Populációs neuronhálózat implementálása PVM környezetre

(szakdolgozat)

Payrits Szabolcs

Eötvös Loránd Tudományegyetem Természettudományi Kar
programozó matematikus szak

Témavezető:

Dr. Érdi Péter

MTA KFKI Részecske és Magfizikai Kutatóintézet
Biofizika osztály

1. BEVEZETÉS

A neurális modellek az emberi agy megértésére irányuló kísérletek fontos eszközei.

A számítógépes idegtudományban *bottom-up* modellekkel foglalkozunk. Ezek olyan modellek, ahol az agyról, mint hierarchikus struktúráról, elemi alkotórészekből álló modellt készítünk.

Napjainkban, a felhasználható számítógépes kapacitás gyors növekedése miatt ezek a modellek egyre realisztikusabbak lehetnek. Ugyanakkor az agy bonyolultsága miatt a szükséges számítási kapacitás még mindig sok nagyságrenddel a lehetőségek alatt marad, és a köveljövőben nem is várható, hogy a teljes agy minden egyes sejtje, axonja és szinapszisa biológiailag releváns módon szimulálható lenne.

A számítási kapacitás korlátozottsága miatt a bottom-up modellek többsége a két lehetséges véglet valamelyikét célozza meg. Ez a két véglet a bonyolult egysejtmodellek illetve a egyszerű elemekből álló hálózatok.

1.1. Részletes egysejtmodellek

Ebben a megközelítésben minden egyes idegsejtről részletes morfológiai és biofizikai modellt készítünk, ahol is a sejt *szómáját*, *axonját* és *dendritjét* is több száz vagy több ezer *sejtmembránrészre*, úgynevezett *rekeszre* (compartment) osztjuk, és egy ilyen rekeszt valamint a rekeszek közti kapcsolatokat részletesen modellezzük valamilyen *Hodgkin-Huxley-szerű* modellt alkalmazva (lásd 2.1 Fejezet). A sejtek közötti kapcsolatokat részletes *szinapszismodellekkel* valósítjuk meg. llymódon részletes és fiziológiailag reális modellek készíthetők, azonban korlátozott a módszer méretben (azaz sejtek számában) és a szimulációs időben is (csak rövid időtartamok szimulálhatók).

1.2. Hálózati modellek

A lehetséges *bottom-up* modellezési technikák skálájának másik végén azok a modellek helyezkednek el, amelyeknél igencsak leegyszerűsített sejtmodellből építenek fel nagy hálózatokat, ahol is a hangsúly nem az egyes elemek (sejtek),

hanem az egész hálózat viselkedésén, sőt, időbeli változásán (tanulás) van. Leegyszerűsített egysejtmodellekkel és a a sejtek között egyszerűsített kapcsolatfüggvényekkel dolgozik. A sejtek állapottere gyakran egydimenziós (sőt, véges), az idő gyakran diszkrét, a kapcsolatokat pedig lépcső vagy szigmoidfüggvények reprezentálják. Hangsúlyt helyez azonban a hálózat állapotának nagyobb időléptékű időbeli változására, amit, mint tanulást interpretálunk. Erre a végletre klasszikus példa a McCulloch-Pitts (MCP) modell, az erre alapuló Perceptron, valamint továbbfejlesztései, a Hopfield-modell, Kohonen-háló, back-propagation hálózatok. Ezek a modellek azonban már a mesterséges neuronhálózatok tárgykörébe tartoznak, és inkább konkrét optimalizálási feladatok megoldására, mint idegrendszeri modellezésre szolgálnak, így most részletesen nem foglalkozunk velük.

Ami viszont lényeges, hogy a hálózati modellek nagyobb időskálán dolgoznak és több elemmel képesek dolgozni, mint a részletes egysejtmodellek. Azonban mikroszkopikus, fiziológiai jelenségek nem modellezhetők és nem jelezhetők előre a segítségükkel.

1.3. Populációs modell

Az idegrendszeri modellezés jelentős problémája tehát, hogyan köthető össze az agy makroszkopikus és mikroszkopikus állapota, hogyan készíthető *mezoszkopikus* jelenségeket magyarázó modell. A probléma hasonló az olyan mikroszkopikus fizikai mennyiségek, mint molekulák helye és sebessége, és olyan makroszkopikus jellemzők, mint például a hőmérséklet viszonyához. Felvetődik, hogy a statisztikus fizika eszköztára, de legalábbis alapelvei talán alkalmazhatók az idegrendszeri modellezésre, abból a célból, hogy lemondva minden egyes idegsejt részletes jellemzéséről, az idegsejtpopulációk mezoszkopikus jelenségeit megfigyelhessük, és ez a modellünk mégis egy realisztikus, részletes modellből származtatott legyen.

A KFKI Részecske és Magfizikai Kutatóintézetében kidolgozott *populációs modell* (Grőbler,Barna,Érdi, 1998) egy olyan statisztikus módszereken alapuló modell, amely részletes egysejtmodellből származatott, és biológiailag releváns makroszkopikus mennyiségek mérésére alkalmas. A modell Traub és Miles (Traub és Miles, 1991) valamint Ventriglia (Ventriglia, 1974) munkájára támaszkodik.

1.4. Összehasonlítás

A következő táblázat mutatja a fenti modellezési stratégiák összehasonlítását:

	Részletes egysejtmodellek	Hálózati modellek	Populációs modell
Biológiailag reális ?	Igen	Nem	Igen
Modellezett elemek száma	Kevés (1-100)	Sok	Sok
Szimuláció ideje	Rövid (ezredmásodpercek – másodpercek)	Hosszú (akár órák)	Közepes (percek)
Információ	Mikroszkopikus	Makroszkopikus	Mezoszkopikus
Tanulás modellezhető ?	Nem	Igen	Igen

2. A POPULÁCIÓS MODELL ELŐZMÉNYEI

A populációs modell részletes egysejtmodelből számaztatjuk, azaz sok olyan paramétert és fogalmat használ fel, amit az egysejtmodellezésből ismerünk. Így először ezekkel a fogalmakkal kell megismerkednünk, mindenekelőtt a idegsejtek alapmodelljével, a Hodgkin-Huxley modellel.

2.1. A Hodgkin-Huxley modell

A Hodgkin-Huxley modell egy membránrésznek, egy *rekesznek* a dinamikáját írja le. Egy *rekeszen* belül a dinamikát az *ioncsatornák* működése adja. Az ioncsatornákon keresztül ionok vándorolnak a sejtmembrán két oldala között, az ioncsatornáknak *állapotuk* van, ami leírja, mennyire engedik át az ionokat a sejtmembránon.

A fentiek alapján egy rekesz egy közönséges differenciálegyenletrendszerrel modellezhető. Az első általánosan elfogadott modell, amely valós méréseken alapult, Hodgkin és Huxley nevéhez fűződik (Hodgkin és Huxley, 1952), akik a tintahal axonjának kiváltott *akciós potenciálját* mérték és modellezték. Ezért az eredményükért 1963-ban orvosi Nobel-díjat kaptak. A modellben a szerzők nátrium és káliumcsatornákat alkalmaztak, az általuk felírt differenciálegyenletrendszer pedig a következő :

(1) $$C \cdot \frac{dV(t)}{dt} = I_{Na} + I_K + I_l$$

(2) $$I_{Na} = g_{Na}^{\max} \cdot m^3 \cdot h \cdot (V(t) - E_{Na})$$

(3) $$I_K = g_K^{\max} \cdot n^4 \cdot (V(t) - E_K)$$

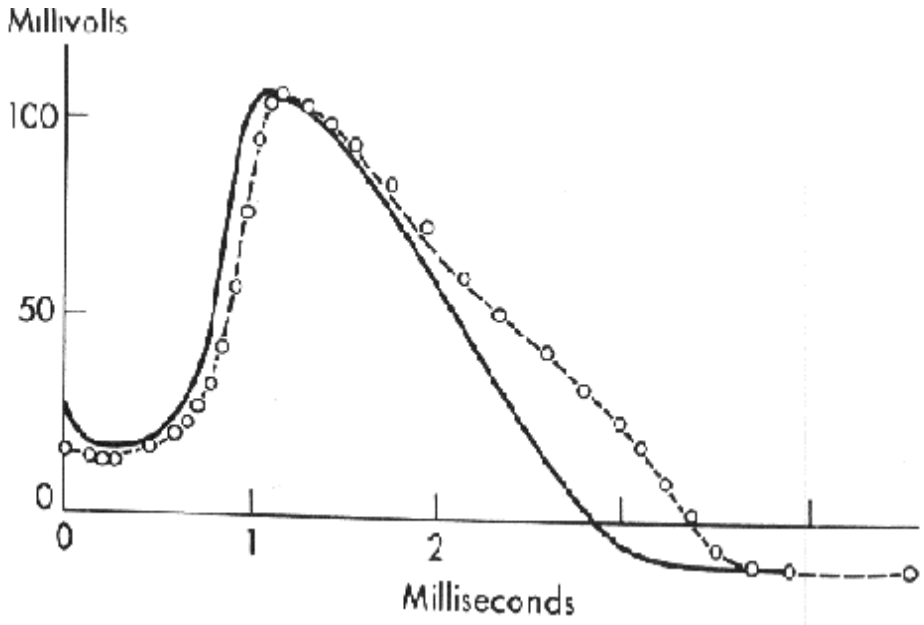
(4) $$I_l = g_l \cdot (V(t) - E_l)$$

Ahol *C* a membrán elektromos kapacitása és I_{Na} a nátriumcsatornán, I_K pedig a káliumcsatornán átfolyó elektromos áram, I_l-t pedig *passzív szivárgási áramnak* nevezzük, $g_{\max}$ az egyes csatornákhoz tartozó maximális vezetőképesség, *E* az egyes ionáramokra jellemző *egyensúlyi potenciál*, konstans érték, *m,n* és *h* pedig a csatornák állapotait leíró úgynevezett *kapuzóváltozók*, amelyekre egyenként a következő differenciálegyenlet vonatkozik:

(5) $$\frac{d\Theta}{dt} = A_\Theta(V(t)) \cdot \Theta + B_\Theta(V(t)) \cdot (1-\Theta) \qquad \Theta \in \{m,n,h\}$$

Ahol A_Θ és B_Θ az egyes kapuzóváltozókhoz tartozó, aktuális *membránpotenciáltól* függő függvények irják le az ioncsatornák kinetikáját.

Az 1.1 ábrán egy ténylegesen mért és egy szimulált kiváltott *akciós* potenciál összehasonlítása látható.



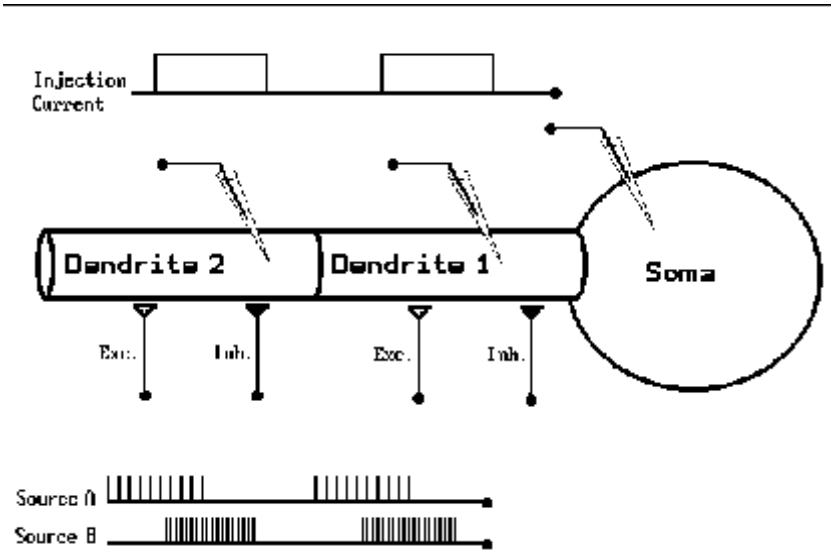
2.1 ábra: Az akciós potenciál alakja Hodgkin és Huxley modelljében (körök) és a valódi axonban (folytonos vonal).

2.2. A Hodgkin-Huxley modell továbbfejlesztései

Az eredeti (Hodgkin-Huxley) modell nem számol semmiféle morfológiával, csupán az idegsejt egyetlen membrándarabjának viselkedésével. Ennél többet szeretnénk, valódi, térbeli kiterjedéssel rendelkező idegsejtek, esetleg idegsejtek hálózatának szimulációját. Ehhez azonban szükségünk van arra, hogy meghatározzuk a rekeszek egymáshoz való térbeli viszonyát, és az azokat összekötő egyenleteket. (1.2 ábra) Feltesszük, hogy a fizikában alkalmazott *kábelegyenlet,* ami egy hosszú, szigetelt, henger alakú elektromos vezetõ feszültségének térbeli és idõbeli változására felírható parciális differenciálegyenlet, alkalmas az idegsejtek nyúlványaiban és sejttestjében a membránpotenciál terjedésének leírására.

A neuroanatómus által készített morfológiai leírásokból tehát meghatározzuk az egyes rekeszek helyét és méretét. Ez többnyire igen bonyolult feladat, hiszen a sejt minden egyes dendritjét, és az axonjának minden elágazását figyelembe kell vennünk. Ezután a fiziológustól kapott adatok alapján pedig meghatározzuk az egyes rekeszeken az ioncsatornák sűrűségét (ami egy rekeszen belül konstans).

Így minden rekeszen belül egy közönséges differenciálegyenletet, a rekeszek között pedig egy annyi változós parciális differenciálegyenletet kell megoldanunk, ahány rekeszre az idegsejtet felosztottuk. Látható, hogy nagyon sok számítást kell minden egyes idegsejtre végeznünk, így a számítási kapacitás többnyire csupán egyetlen sejt vagy legfeljebb néhány sejtből álló hálózat modellezésére elegendő.



2.2 ábra: Az idegsejt rekesz-modellje

Részletes egysejtmodellezésre alkalmas programcsomag több is fellelhető, ilyenek például a Neuron (Hines, 1993) és a Genesis (Bower és Beeman, 1995), amely utóbbi a Debian GNU/Linux operációs rendszer csomagjai között is megtalálható.

2.3. Szinapszisok

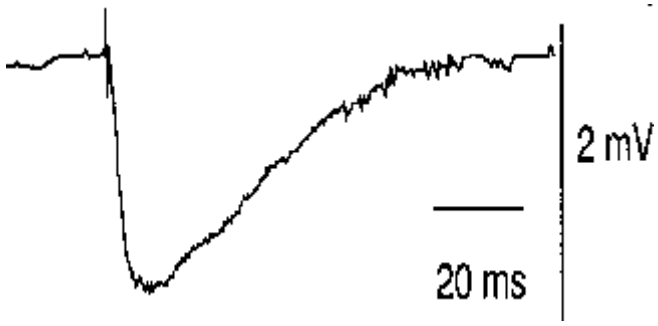
A sejteket összekapcsoló elemek a *szinapszisok.* A legtöbb szinapszis az ingerületet kémiai úton közvetíti, a szinapszis *preszinaptikus* sejt felőli részéből *transzmitter* anyag szabadul fel, ami *posztszinaptikus* sejt membránján a receptorokhoz kötődik, és ott az ioncsatornák állapotát megváltoztatja. Ennek a hatása lehet serkentés (a membránpotenciál növelése) vagy gátlás (a membánpotenciál csökkentése).

A szinapszisok modellezésére a részletes egysejtmodellek esetén használnak differenciálegyenleteket, a hálózati modellekben azonban ennél egyszerűbb módszereket alkalmaznak, és a később tárgyalandó modellben is egy egyszerűbb megközelítést alkalmazunk.

Egyik ilyen lehetséges fenomenológiai megközelítés, ha a mérési eredmények és a részletes szinapszismodellek alapján meghatározunk egy függvényt, ami megadja az

idő függvényében, hogy a szinapszis ingerlése esetén mekkora a *posztszinaptikus* membránon mért, a szinapszis által okozott áram. Ez hivjuk *szinaptikus áramnak*.

Az 1.3 ábra egy gátló szinapszis hatását mutatja a posztszinaptikus sejten. Mivel gátló szinapszisról van szó, itt negatív értékeket látunk.



2.3 ábra: Gátló szinapszis hatása a hippokampusz CA3 régiójában mérve. Az idő függvényében mutatja a posztszinaptikus sejt membránjának potenciálváltozását.

2.3.1 Alfafüggvény

Egy ilyen függvény, amit gyakran alkalmaznak, az ún. a*lfafüggvény*, amely a következő:

(6) $$g_{syn}(t) = g_{\max} \cdot t \cdot \frac{e^{(1-\frac{t}{t_0})}}{t_0}$$

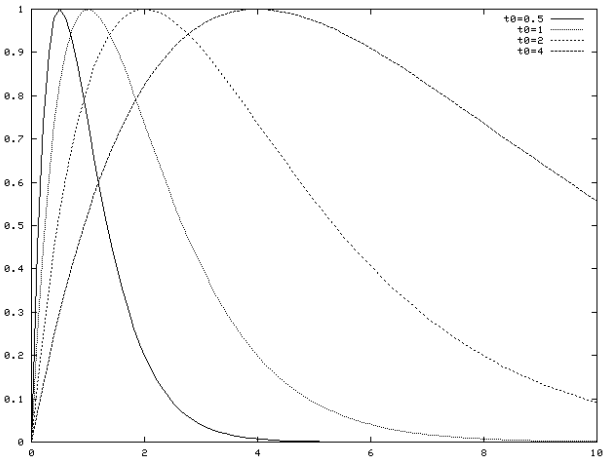
ahol a függvény a maximumát t = t_0 -ban veszi fel, ez az érték $g_{\max}$,

Az alfafüggvény valójában a szinaptikus áttevődés által okozott konduktanciaváltozást adja meg, azaz ha több szinapszis kapcsolódik ugyanarra a posztszinaptikus sejtre, akkor a konduktanciaértékek összegződnek, nem a szinaptikus áramok (ez a későbbiekben fontos lesz). A szinaptikus konduktanciából a szinaptikus áram, a Hodgkin-Huxley egyenletbe beírva a következő módon számolható:

(7) $$I_{syn}(t) = \sum_i g_{syn}^{(i)}(t) \cdot (V(t) - E_{syn})$$

ahol V(t) a Hodgkin-Huxley egyenletben szereplő membránpotenciál, E_{syn} a szinapszis egyensúlyi potenciálja, konstans.

Egy alfafüggvénnyel megadott szinapszis tehát jellemezhető a maximális értékével($g_{\max}$) és sebességével (t_o). A függvény alakját az 2.4 ábra mutatja.



2.4 ábra: Az alfafüggvény alakja különböző t_0 paraméterek esetén

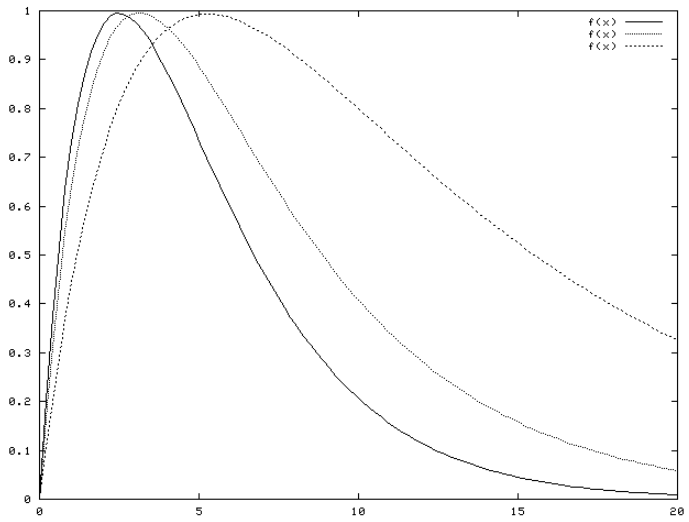
2.3.2 Dupla exponenciális függvény

Egy másik gyakran alkalmazott függvény a dupla exponenciális függvény, amelynek alakja a következő:

(8) $$g_{syn}(t) = C \cdot \frac{g_{\max}}{\tau_1 - \tau_2} \cdot (e^{-\frac{t}{\tau_1}} - e^{-\frac{t}{\tau_2}})$$

ahol két időkonstans, τ_1 és τ_2 szerepel az egyenletben, C pedig skálázóváltozó, amivel elérhető, hogy a függvény $g_{\max}$ -ot vegye fel a maximumában.

A konstansok megfelelő beállításával itt a valódi posztszinaptikus konduktanciaváltozáshoz hasonló görbealakot kaphatunk.



2.5 ábra: Dupla exponenciális függvények

3. POPULÁCIÓS MODELL

3.1. A populációs modell származtatása

A populációs modellt a fenti módszerekből és modellekből származtatjuk olymódon, hogy a rendszer aktuális állapotát nem az egyes sejtek állapotváltozóival jellemezzük, hanem az egyes állapotok együttes eloszlásfüggvényével.

Azaz ha az eredeti modell állapottere :

$A = S^n$ ahol *n* a sejtek száma, S egy sejt állapottere, azaz

$S = \Re^k$ ahol egy sejtet *k* darab folytonos állapotváltozóval írunk le.

Ilyenek például a membránpotenciál, különböző ionkoncentrációk, valamint az ioncsatornák *kapuzó változói.*

Ezt az állapotteret transzformáljuk egy olyan *g(t,**r**)* függvénybe, amely minden időpillanatban(*t)* a sejttér (azaz a tényeges idegszövet) minden pontjához(***r***) egy *k*-dimenziós eloszlásfüggvényt rendel. Fontos, hogy most folytonos sejttérről beszélünk, azaz szabadon választhatjuk meg a megoldási módszerünkben, mennyire részletesen diszkretizáljuk a folytonos sejtteret, attól függően, mennyire részletes képet akarunk kapni a hálózatunk működéséről.

Vegyük észre, hogy a transzformáció igen érzékeny az állapotváltozók számára, hiszen míg egy sejt állapotát *k* darab valós számmal jellemezhetjük, addig a hozzá rendelt *k* dimenziós **eloszlásfüggvény** jellemzése jóval több adatot igényel. Míg az eredeti állapottérben a állapottér tárolásához a sejt állapotváltozóival lineárisan arányos memória és számítási kapacitás szükséges ($C = a \cdot k$, ahol *a* egy konstans), addig a transzformált állapottérben a szükséges memória és számításigény az állapotváltozóknak exponenciális függvénye ($C = a^{'} \cdot b^k$, ahol *a* konstans, *b* pedig megoldási módszertől függő konstans).

Azonban látható, hogy a sejtek száma a transzformált állapottérben, mint lineáris szorzótag az erőforrásigényben, nem jelenik meg, ennyit nyertünk a transzformációval.

Helyette az általunk választott diszkretizációs fok (azaz a sejttér hány pontjában számoljuk a dinamikát) jelentkezik, szintén, mint lineáris szorzótag.

Mindez azt jelenti, hogy a modellünk a következő kritériumok teljesülése esetén lesz hatékony és használható :

- Egy sejtet a lehető legkevesebb állapotváltozó jellemez.
- Sok sejtet tartalmazó sejtteret szimulálunk.
- A sejtek számához képest alacsony felbontással vizsgáljuk a jelenségeket, a sejtek számánál lényegesen kevesebb pontra diszkretizálunk a sejttérben.

Tehát amit valójában transzformálnunk kell, az a dinamika, azaz kérdés, hogy hogyan transzformálható a sejtmembránra felírt differenciálegyenletrendszer egy, a fenti *g* függvényre felírt differenciálegyenletté, úgy, hogy a makroszkopikusan mérhető mennyiségek megfeleltethetők legyenek egymásnak.

Vizsgáljuk meg, milyen egysejtmodellből kell kiindulnunk a fentiek figyelembevételével.

3.2. Az egysejtrész

Az egysejtmodellt meghatározásakor figyelembe kell vennünk, hogy milyen, biológiailag releváns viselkedést várunk el a kiinduló modellünktől, és ezt hogyan tudjuk a lehető legkevesebb állapotváltozó bevezetésével elérni.

A következő tulajdonságokat követeljük meg a kiinduló modellünktől:

- Legyen képes akciós potenciálok előállítására
- Legyen képes ún. *burstre*. A *burst* az akciós potenciálok gyors egymásutánban való megjelenése, ami jellemző az egykéregben és a hippokampuszban található *piramissejtekre*, ami a serkentő idegsejtek egy fontos szerepet játszó fajtája.
- A Hodgkin-Huxley modellből származatott legyen.

A fentiek figyelembevételével Grőbler,Barna és Érdi egy **kétdimenziós** modellt adtak (Grőbler,Barna,Érdi, 1998), amelynek változói az idegsejt membánpotenciáljának

illetve a sejten belüli kálciumtartalomnak felel meg. Bevezettek azonabn egy új állapotváltozót, amely két értéket vehet fel, és azt jelöli, hogy a sejt tüzelő vagy nemtüzelő (normál) fázisban van-e. Ez a – így összesen három – állapotváltozó mégiscsak kétdimenziós eloszlásfüggvénnyé transzformálódik, viszont **két** ilyen eloszlásfüggvénnyé, az egyik a normál állapotban, a másik a tüzelő állapotban levők számára.

A dinamikát a következőképpen kapjuk: elhagyjuk a nátriumcsatornákat leíró áramokat, feltételezve, hogy a nátriumáramok csupán az akciós potenciálok csúcsáért felelősek, és a többi árammal csupán a tüzelések közötti dinamikát számoljuk. Bevezetünk viszont két új áramot az eredeti Hodgkin-Huxley modellhez képest, amelyek a sejt kálciumtartalmától függnek. Jelöljük a membránpotenciált *V*-vel, a sejten belül kálciumkoncentrációt *X*-el. Ekkor a következő differenciálegyenletrendszer írja le a sejtek tüzelések közötti viselkedését :

(1) $$-C \cdot \frac{dV(t)}{dt} = I_{Ca}(V(t)) + I_K(V(t)) + I_{K[Ca]}(V(t), X(t)) + I_L(V(t)) + I_{\exp}(t)$$

(2) $$\frac{dX(t)}{dt} = -\beta \cdot X(t) - B \cdot I_{Ca}(V(t))$$

ahol β és B konstansok, I_{Ca} a kálciumcsatornákon átfolyó áram, I_K a kálumcsatonák árama, $I_{K[Ca]}$ az ún. kálciumfüggő káliumáram, ami olyan káliumcsatornán keresztül folyik, aminek állapota függ a sejt kálciumtartalmától, I_L pedig a *szivárgási* áram (leakage), ami az olyan lassan működő ionpumpák működésének eredménye, amelyek a sejtet a nyugalmi állapotába (E_{rest}) igyekeznek visszavinni, I_{ext} a sejtbe kívülről érkező áram (például szinapszisokon keresztül)

A tüzelő állapotban lévő sejtekre dinamikát nem számolunk, a tüzelő állapotba került sejteket az ún. *refracter* idő (t^{ref}) letelte után (V_{ret}, X_{ret}) állapotba helyezzük:

(3) $$X_{ret} = X + \Delta X$$

(4) $$V_{ret} = \begin{cases} -55 + \dfrac{X_{ret}}{B \cdot C} & ha \quad X_{ret} < 10 \\ -30 - X_{ret} & ha \quad X_{ret} \geq 10 \end{cases}$$

ahol *B* és *C* konstansok.

Meg kell még adnunk annak a definícióját, mikor tekintünk egy sejtet tüzelő állapotban lévőnek. Csak a membránpotenciál áll rendelkezésünkre ennek eldöntésére. A triviális megoldás az lenne, ha egy bizonyos *tüzelési küszöb (Θ)* felett a sejteket tüzelőnek minősítenénk. Biológiailag reálisabb azonban, és a későbbiekben látni fogjuk, hogy a modellünkbe is jobban illeszkedik, ha inkább definiálunk egy P(V_1, V_2) valószínűségi függvényt, ami megadja, hogy ha a sejt membránpotenciálja V_1-ről V_2-re változik, akkor mekkora valószínűséggel tüzel a sejt. Ahhoz, hogy ez konzisztens legyen, a következő feltételeket kell teljesíteni :

- P(V_1, V_2) nem függ attól, mennyi idő alatt jut el a sejt V_1-ből V_2-be.
- Mivel a legtöbb tüzelés a sejt tüzelési küszöbénél történik, így $\lim_{V \to \infty} P(V_1, V) = 1$

  ha $V \leq \Theta$,de nullához tart, ha $V > \Theta, V \to \infty$.

Grőbler,Barna és Érdi megmutatta (Grőbler,Barna,Érdi, 1998), hogy a következő függvény kielégíti a fenti feltételeket:

(5) $$P(V_1, V_2) = 1 - e^{-\int_{V_1}^{V_2} p(v)dv}$$, ahol

(6) $$p(V) = \begin{cases} q \cdot e^{\frac{V-\Theta}{V^*}} & ha \quad V < \Theta \\ q \cdot e^{\frac{-(V-\Theta)}{V^*}} & ha \quad V \geq \Theta \end{cases}$$

ahol V^* és *q* konstansok.

Hogy a modell teljes legyen, definiálnunk kell az (1)-ben szereplő ionáramokat. Mint említettük, az állapotterünk csupán kétdimenziós, így kapuzóváltozóink sincsenek. De mivel a nátriumáramokat elhanyagoltuk, így csupa olyan áramunk marad, amihez ún. lassú csatornák tartoznak. Vegyük hát ezek határértékét $t \to \infty$ esetén, így a következő függvényeket kapjuk :

(7) $$I_{Ca}(V) = \gamma_{Ca} \cdot s^5(V) \cdot (V - E_{Ca})$$

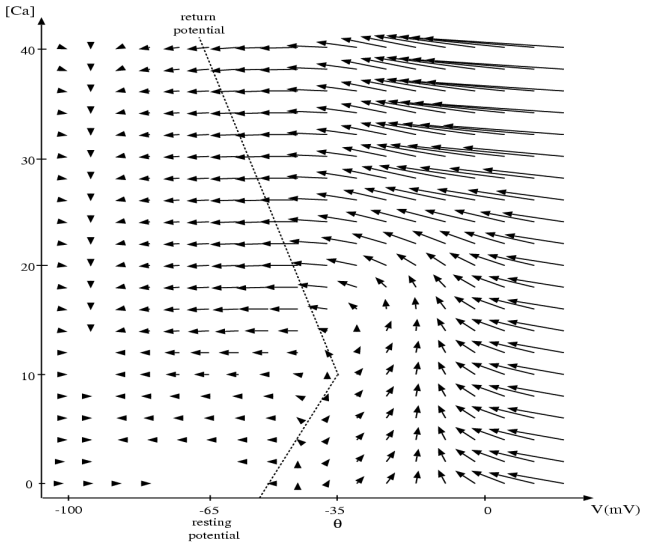
(8) $$I_K(V) = \gamma_K \cdot n^4(V) \cdot (V - E_K)$$

(9) $$I_{K[Ca]}(V, X) = \gamma_{K[Ca]} \cdot q(V, X) \cdot (V - E_K)$$

(10) $$I_L(V) = \gamma_L \cdot (V - E_{rest})$$

ahol E_{rest} a sejt ún. *nyugalmi potenciálja*, E_K, E_{Ca} és $E_{K[Ca]}$ pedig a kálium, kálcium és kálciumfüggő káliumionok *egyensúlyi potenciálja*, mindegyik konstans, a γ-ák pedig vezetőképességet meghatározó konstansok.

Az így kapott egysejtmodell dinamikáját, a konstansok megfelelő megválasztásával, a 3.1 ábra mutatja. A vízszintes tengely a membránpotenciált, a függőleges a sejten belüli kálciumtartalmat mutaja. A nyilak jelzik az adott állapotban az áramot és kálciummozgást, nevezetesen a nyilak vízszintes komponense az elektromos áramot, a függőleges pedig a be- vagy kiáramló kálcium mennyiségét mutatja.



3.1 ábra: Az egysejtmodell dinamikája

3.3. Gátlósejt-modell

A fenti egysejtmodell alkalmazható gátlósejtekre is, mivel azonban gátlósejteken nem figyelhető meg *burst*, így elegendő csupán egyetlen állapotváltozót, a membránpotenciált figyelembe venni, és a tüzelési küszöbértéket is fixre (Θ_{inh}) választjuk, azaz:

(11) $$C \cdot \frac{dV(t)}{dt} = -(I_L(V(t)) + I_{ext})$$

(12) $$P(V_1, V_2) = \begin{cases} 1 & ha\ V_1 < \Theta \wedge V_2 > \Theta \\ 0 & egyebkent \end{cases}$$

Valamint a tüzelésből visszatérés esetén a sejt (V_{ret}) állapotba kerül.

(13) $$V_{ret} = E_{rest}$$

ahol csakúgy, mint fent, E_{rest} a sejt nyugalmi potenciálja.

3.4. A feladat transzformálása

Adott most már az egysejtmodellünk, transzformáljuk a feladatot a 3.1 fejezetben említett alakú *g* függvénnyé. Legyen most *r* rögzített, azaz egy szövettérbeli pontot tekintünk, a transzformált egyenlet ekkor így alakul:

(14)
$$\frac{\partial g_s(r,u,\chi,t)}{\partial t}+\frac{\partial}{\partial u}(\varepsilon_s(r,u,\chi,t)\cdot g_s(r,u,\chi,t))+\frac{\partial}{\partial \chi}(\eta_s(u,\chi)\cdot g_s(r,u,\chi,t))-$$
$$\frac{D_u}{2}\cdot\frac{\partial^2 g_s(r,u,\chi,t)}{\partial u^2}-\frac{D_\chi}{2}\cdot\frac{\partial^2 g_s(r,u,\chi,t)}{\partial \chi^2}=b_s(r,u,\chi,t)-n_s(r,u,\chi,t)$$

ahol is

- *s* a populációt jelzi, több, akár egymáson térben átfedő populációt is figyelembe vehetünk egyidőben.
- g_s az *s* populációhoz tartozó olyan függvény, ami minden időpillanatban (*t*) minden térbeli ponthoz (***r***) egy kétdimenziós, *u* és χ szerinti valószínűségeloszlásfüggvényt rendel, ahol *u* a membránpotenciálnak, χ pedig a sejt kálciumtartalmának felel meg.
- ε_s és η_s a sejtbe folyó elektromos áram, illetve beáramló kálcium mennyiségét adják meg, ε_s-t (1)-ből, η_s-t (2)-ből származtatjuk.
- n_s egy olyan valószínűségi függvény, ami azt adja meg, milyen valószínűséggel mennek tüzelni a sejtek az adott állapotban, b_s pedig, hogy mekkora valószínűséggel térnek vissza a tüzelésből az adott állapot**ba**. Ezeket a függvényeket (3)-(6)-ból származtatjuk.
- D_u és D_χ a diffúzió mértékét meghatározó konstansok.

A fenti egyenlet érdekessége, hogy második deriváltakat is tartalmaz, azaz diffúzió figyelhető meg az eloszlásfüggvény dinamikájában. Ez biztosítja, hogy ha a szövetbeli diszkretizáció léptéke nagy, az idegsejtek közötti lokális kapcsolatokat, mint diffúz összeköttetéseket, figyelembe vesszük.

Adjuk meg most a fenti egyenletben szereplő ismeretlen függvényeket. Kezdjük a legegyszerűbbel, a kálciumbeáramlást leíró egyenlettel, ez teljes egészében átvehető (2)-ből :

(15) $$n_s(u,\chi) = -\beta \cdot \chi - B \cdot I_{Ca}(u)$$

Az elektromos áramot leíró egyenlet is hasonló (1)-hez, de itt bevezetünk egy $I^{syn}_{s,s'}$ tagot, ami azt mutatja, mekkora áram érkezik a szinapszisokon keresztül az *s'* populációból az *s* populációba a megadott állapotban.

(16) $$\varepsilon_s(r,u,\chi,t) = -\frac{1}{C_s} \cdot \sum_i I_s^{(i)}(u,\chi) - \frac{1}{C_s} \cdot \sum_{s'} I^{syn}_{s',s}(r,u,t)$$

$I^{syn}_{s's}$

Fontos, hogy ez az egyetlen tag az egyenletben, ami kapcsolatot teremt nemcsak a különböző populációk között, de egy populáció különböző térbeli pontjai között is, azaz minden térbeli kapcsolat a $I^{syn}_{s',s}$ tagon keresztül jön az egyenletbe. Fejezzük ki tehát $I^{syn}_{s',s}$ -t :

(17) $$I^{syn}_{s',s}(r,u,t) = -\gamma_{s's}(r,t) \cdot (u - E_{s',s})$$

ahol $E_{s',s}$ az *s'* és az *s* populáció közti nyugalmi potenciál, konstans, $\gamma_{s',s}(r,t)$ a posztszinaptikus populáció (azaz *s*) *szinaptikus konduktanciája* az ***r*** pontban és t időpillanatban. (lásd 2.3 fejezet)

A fenti szinaptikus konduktancia meghatározása lesz a következő fejezet témája, hiszen ezen keresztül "kommunikálnak" egymással a sejtek, pontosabban a folytonos sejttér pontjai. Most azonban írjuk még fel a hiányzó egyenleteket, amelyek leírják a tüzelni menés és tüzelésből visszatérés valószínűségeit.

(18) $$n_s(r,u,\chi,t) = \begin{cases} p_s(u)\cdot\varepsilon_s(r,u,\chi,t)\cdot g_s(r,u,\chi,t) & ha\ \varepsilon_s(r,u,\chi,t) > 0 \\ 0 & egyebkent \end{cases}$$

ahol $p_s(u)$ már definiáltuk (6)-ban, s pontosan úgy, ahogy, hogy itt felhasználhassuk.

(19) $$b_s(r,u,\chi,t) = \int_{-\infty}^{t}\int_{-\infty}^{\infty}\int_{0}^{\infty} n_s(r,u',\chi',t')\cdot\delta(u-U_s(u',\chi',t'))\cdot \delta(\chi-\chi_s(u',\chi',t'))\cdot\delta(t-T_s(u',\chi',t'))du'dt'd\chi'$$

ahol δ olyan valós számokon értelmezett függvény, ami nullában 1-et, mindenhol máshol nullát vesz fel, U_s, χ_s, T_s pedig a (3) és (4) egyenletekből származtatható függvények, amelyek megmondják, milyen állapotban jöjjön ki a sejt a tüzelő állapotból, azaz:

(20) $$U_s(u,\chi,t) = V_{ret}(\chi+\Delta\chi)$$

(21) $$\chi_s(u,\chi,t) = \chi+\Delta\chi$$

(22) $$T_s(u,\chi,t) = t + t^{ref}$$

3.5. A kapcsolatfüggvények

Az előző fejezetben a szinaptikus konduktanciát meghatározó tagot nem definiáltuk, ezt most itt fogjuk megtenni. Ezen a ponton eltérünk a Grőbler-Barna-féle eredeti modelltől. Míg a eredeti modellben a szinaptikus konduktanciaváltozást egy térben

terjedő *spike*-okból álló másodlagos populáció (spikepopuláció) terjedése okozta, addig itt a tér pontjai között *kapcsolatfüggvényeket* vezetünk be.

A változtatás fő oka, hogy az eredeti modellben csupán diffúz, lokális és nem specifikus kapcsolatokat lehetett modellezni, azaz kizárólag olyan agyterületek szimulálására volt alkalmas, amelyekben a neuronok kapcsolatai lokálisnak és véletlenszerűnek volt tekinthető, ilyen például a hippokampusz

A kapcsolatfüggvények bevezetésével azonban lehetővé vált más, nem ilyen tulajdonságú agyterületek modellezése, s nem utolsó sorban, változó kapcsolatfüggvényket feltételezve, az agyban lezajló tanulási folyamatok is nyomon követhetőek lesznek.

Írjuk fel tehát a szinaptikus konduktanciát meghatározó függvényt kapcsolatfüggvények segítségével:

(23) $$\gamma_{s',s}(r,t) = \int_{R_{s'}} \Phi_{s'}(r') \cdot \int_0^{\infty} a_{s'}(r', t - t' - d_{s',s}(r', r)) \cdot \alpha_{s',s}(k_{s',s}(r', r), t')\, dt\, dr'$$

- ahol Φ a *sűrűségfüggvény*, ami a tér minden pontjához megadja az ott érvényes sejtsűrűséget,
- $R_{s'}$ az s' populáció teljes szövetterét jelöli
- $a_{s'}(r,t)$ az s' populáció *aktivitásfüggvénye* az *r* pontban *t*-kor.
- $d_{s',s}(r',r)$ az s' és s populáció r' és r pontja közti késleltetést adja meg, ami annak felel meg, mennyi időre van szüksége az ingerületnek, hogy végigfusson a megfelelő axonon,
- $k_{s',s}(r',r)$ a *kapcsolatfüggvény,* az s' és s populáció r' és r pontja közti kapcsolaterősséget adja meg.
- $\alpha_{s',s}$ az *alfafüggvény*, magát a szinapszist modellezi. Lásd 2.3 fejezet.

Az *aktivitásfüggvény* alatt a adott populáció adott pontjában lévő éppen tüzelő sejtek számát értjük, amit a következőképpen definiálhatunk:

(24) $$a_s(r,t) = \int_0^{t^{ref}} \int_{-\infty}^{\infty} \int_0^{\infty} (n_s(r,u',\chi',t-t') - b_s(r,u',\chi',t-t'))d\chi' du' dt'$$

Amit a definíciónál lényegesen egyszerűbben szimulálhatunk :-)

A modellt tehát a fentiekben definiáltuk.

4. A PROGRAM

A populációs modell, mint számítógépes program a 3. Fejezetben leírt modell implementációja. A program dokumentációjában felhasználom a 3. Fejezet definícióit, és megadom a függvények és konstansok programbeli megfelelőit.

4.1. Tervezési szempontok

A populációs modell tervezésekor a következő kritériumoknak kellett megfelelni :

- A kód könnyen érthető és könnyen módosítható, de hatékony legyen.
- A program legyen képes párhuzamos környezetben való működésre
- A programnak legyen könnyen kezelhető felhasználói felülete, és legyen lehetséges a modellben adott konstansok és függvények minél egyszerűbb és szabadabb változtatása
- A programmal nyomon követhetőek legyenek a modell számunkra lényeges paraméterei
- A program adjon olyan kimentet, ami későbbi újrafeldolgozásra alkalmas.
- A program különböző platformokra lehetőleg hordozható legyen.

4.2. Párhuzamosítás

Fő tervezési szempont volt a program párhuzamosíthatósága. A program futásához a labor egy 16 darab Intel alapú PC-ből álló clusterrel rendelkezett, melyeken szinten **Linux** operációs rendszer futott.

4.3. Eredmények

A fenti kritériumoknak az elkészült program a következőképpen igyekszik megfelelni :

- A programot **Linux** operációs rendszer alá fejlesztettük, ugyanis a Részecske és Magfizikai Kutatóintézet biofizikai laborjában ez volt a használt operációs rendszer, és így a hordozhatóság is biztosított, hiszen a legtöbb platformon elérhető a **Linux** operációs rendszer.
- A program C++ nyelvben íródott, így könnyen módosítható, remélhetőleg könnyen meg is érthető, mégis gyors kód fordítható belőle.
- A párhuzamosítást a **PVM** (Parallel Virtual Machine) könyvtár- és toolkittel végeztük. Ez egyszerűen kezelhető felületet biztosít mind a cluster karbantartása, mind a populációs modell párhuzamosításért felelős része számára.
- Grafikus környezetnek így az X11 **Gtk+/Gnome** toolkitjét választottam, ez lévén a program írásának kezdetén az egyetlen viszonylag kiforrott és teljesen ingyenes megoldás. Problémát jelentett azonban, hogy a **Gtk+** C nyelvre írodott, és stabilan működő C++ wrapper nem állt rendelkezésre, így törődni kellett a C++-ba való beágyazás problémájával is. A grafikus felület egyszerű kezelhetőséget és a szimulációs eredmények könnyű áttekinthetőségét biztosítja.

5. FELHASZNÁLÓI DOKUMENTÁCIÓ

Ebben a fejezetben a populációs modell számítógépes implementációjának felhasználói dokumentációja következik. Tárgyalni fogom a grafikus felhasználói felület kezelését és a bemenet formátumát, az implementáció részletes bemutatása viszont a fejlesztői dokumentációban található.

5.1. A program telepítése

5.1.1 Szükséges programok és könyvtárak

A Populációs Modell program Linux operációs rendszer alatt működik. A program futtatásához a következő könyvtárak és az ezektől függő minden könyvtár szükséges:

- **libgnome** (**GNOME** könyvtár)
- **libgtk** (**Gtk+** grafikus widget-könyvtár, legalább 1.2-es verzió)
- **libpvm3** (a **PVM** párhuzamosítási könyvtára)
- **libm** (matematikai függvénykönyvtár)

A fentieken kívül telepítve kell lennie a **pvm** kiegészítő programcsomagnak, ami a **pvm** nevű beállítóprogramot, és a **pvmd** nevű démonprogramot tartlamazza.

Ha a programot forrásból szeretnénk fordítani, akkor a fentieken kívül telepítve kell lennie a fenti könyvtárakhoz tartozó fejlesztői csomagok is, valamit szükségünk van a **gzip** és a **tar** valamit a **lex, yacc, GNU make** és **g++** programokra is.

5.1.2 Fordítás

A program forrása a Popmodell_<Dátum>.tgz nevű fileban található. (például Popmodell_2000_Jun_11.tgz).

A programot a következő módon tömöríthetjük ki :

cd <célkönyvtár>
tar zxvf Popmodell_<Dátum>.tgz

Gondoskodjuk arról, hogy a **PVM** környezet megfelelő módon be van állítva, azaz a *PVM_ROOT* és a *PVM_ARCH* környezeti változók értéke megfelelő, és a *home*könyvtárunkban létezik a ${HOME}/pvm3/bin/${PVM_ARCH} könyvtár.

Ezután a program fordítása a

make

parancs beírásával történik. Ez lefordítja a programot, létrehozza a futtatható binárisokat, és elhelyezi a őket a **PVM** számára megfelelő könyvtárba.

5.1.3 Futtatás

A program futtatásához szükséges, hogy a gépünkön fusson X11 kiszolgáló, valamint, hogy a **PVM** környezet már rendelkezésre álljon, fusson a **PVM** kiszolgáló démon. Ekkor az aktuális könyvtárban lévő **pop** nevű bináris futtatásával lehetséges a program főmenüjébe jutni.

5.2. Háttérinformáció

A program használatához az impmentációról a követekező információk elengedhetetlenül szükségesek:

- A program diszkretizálja az időt, azaz a $g(r,u,\chi,t)$ függvényben a t nem a valós számok halmazán oldjuk meg, hanem annak egy diszkrét részhalamzának eleme. Ez gyakorlatilag azt jelenti, hogy $t \in T$, ahol $T = \{k \cdot \Delta t \mid k \in N\}$. A Δt az időtengely mentén a diszkretizáció foka.

- A program diszkretizál a folytonos sejttérben is, ami azt jelenti, hogy ***r*** nem R^2 eleme, hanem $\boldsymbol{r} \in \{a \cdot \Delta x + b \cdot \Delta y \mid a, b \in Z\}$. Δx és Δy a sejttérbeli diszkretizáció foka. Így kapjuk a *sejttérbeli rácspontokat.*
- A program diszkretizál az egy szövetpontbeli ketdimenziós eloszlásfüggvényen is. A folytonos eloszlásfüggvényt tehát a program egy diszkrét eloszlásfüggvénnyel helyettesíti, azaz a *u* és χ valószínűségi változók a következő diszkrét értékeket vehetik fel:

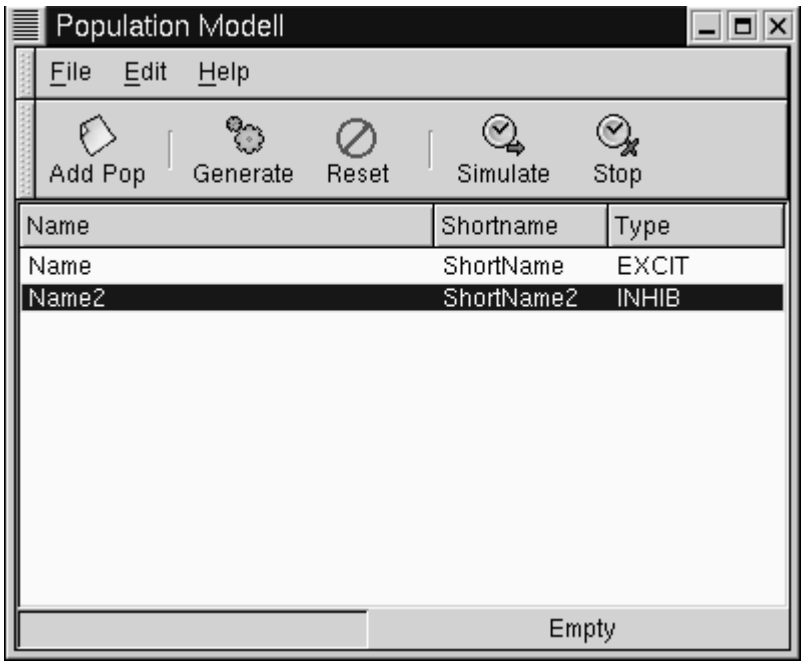
  $u \in \{U_{\min} + k \cdot \Delta u \mid U_{\min} + k \cdot \Delta u \le U_{\max}, k \in N\}$, és

  $\chi \in \{\chi_{\min} + k \cdot \Delta\chi \mid \chi_{\min} + k \cdot \Delta\chi \le \chi_{\max}, k \in N\}$

  Így kapjuk az *állapotérbeli rácspontokat.*

5.3. A program használata

A program indítása után a program főablakába jutunk. A program angol nyelvű.



5.1. ábra: A PopModell program főmenüje

A program menüsorában a következő menüket találjuk:

5.3.1 File menü

A file menüben a következő menüpontok találhatók:

5.3.2 Add Pop

Új populáció hozzáadása. Erre e menüpontra kattintva egy új ablakot kapunk, ahol beállíthatjuk az új populáció jellemzőit. Ennek leírása a **Add Pop** gomb leírásánál található.

5.3.3 Open

Már létező szimuláció megnyitása. A gombra kattintva egy ablakot kapunk, ahol kiválaszthatjuk a betöltendő fájt. Sikertelen betöltés esetén *"Error while loading file"* hibaüzenetet kapunk.

5.3.4 Save

Aktuális szimuláció elmentése az aktuális néven. Ha nincs aktuális név, akkor a program egy fájlválasztó ablakban kér egyet. Így menthetjük el az általunk beállított paramétereit a szimulációnak.

5.3.5 Save as

A szimuláció elmentése más néven.

5.3.6 Close

Az aktuális szimuláció bezárása és a populációk törlése. A program a menüpont választása esetén alaphelyzetbe áll.

5.3.7 Exit

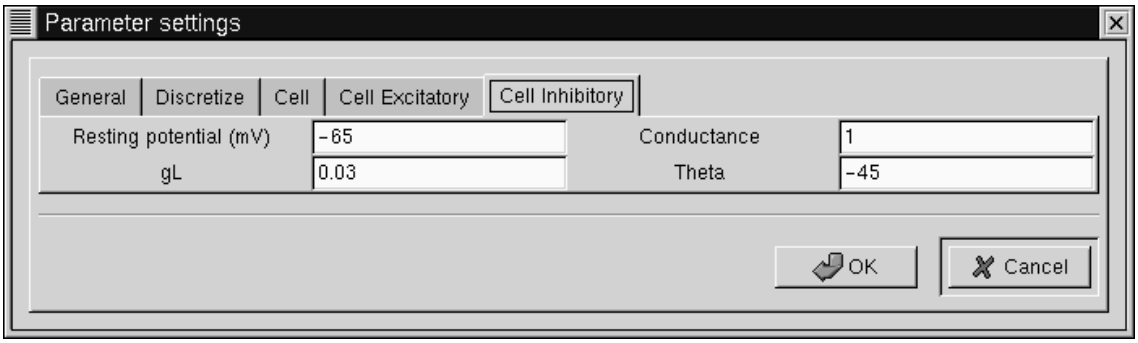
A program a menüpont választása esetén befejezi tevékenységét.

5.3.8 Edit menü

Az Edit menüben a paraméterek szerkesztésével, változtatásával kapcsolatos menüpontok találhatók. Ezek a következők:

5.3.9 Parameters

A menüpont választása esetén egy új ablakot kapunk, ahol a modell konstans paramétereit állíthatjuk be.



5.2. ábra: A paraméterbeállítások ablaka

A konstansok további csoportokra bomlanak, ezek a következők:

General: A modell általános paraméterei. Kezdetben ezek alapértékeit találhatjuk itt, amelyek megfelelnek a Gőbler és Barna (1998) modell paramétereinek. Ezek a következők:

- **Timestep**: Az idődiszkretizáció mértéke, a szimuláció időfelbontása ezredmásodpercekben.
- **Dendritic delay:** A dendriten végighaladó ingerület késését adja meg ezredmásodpercben, ez hozzáadódik minden a 3.fejezet (24) egyenletében szereplő késleltetéshez.

Discretize: A sejttér egyetlen pontjában lévő eloszlásfüggvény megoldásakor használt diszkretizációs konstansok

- **V_min:** A membránpotenciálértékek alsó korlátja (lásd 5.2. fejezet)
- **V_max:** A memránpotenciálértékek felső korlátja (lásd 5.2. fejezet)
- **V_number:** A membránpotenciáltengely mentén ennyi pontra diszkretizálunk.

- **Diff_V:** A 3.fejezet (14) egyenletében szereplő D_u konstans.
- **Ca_min:** A sejten belüli kálciumtartalom alsó korlátja. (lásd 5.2. fejezet)
- **Ca_max:** A kálciumtartalom felső korlátja (lásd 5.2. fejezet)
- **Ca_number:** A kálciumtengely mentén ennyi pontra diszkretizálunk.
- **Diff_Ca:** A 3.fejezet (14) egyenletében szereplő D_{Ca} konstans.

Cell: Az egysejtmodellre vonatkozó általános paraméterek.

- **Stable U:** Az egysejtmodell fixpontjának membránpotenciálértéke.
- **Stable Ca:** Az egysejtmodell fixpontjának kálciumtartalom-értéke.

A fenti értékeket a szimuláció kezdetén, mint kezdetiértéket használjuk.

Cell excitatory: A teljes (serkentő) egysejtmodellre vonatkozó konstansok.

- **C:** A sejtmembrán kapacitása. A 3.fejezet (1), (4) illetve (16) egyenletében szereplő *C* konstans.
- **gCa:** A 3. Fejezet (7) egyenletében szereplő γ_{Ca} konstans.
- **gK:** A 3. Fejezet (8) egyenletében szereplő γ_K konstans.
- **gK[Ca]** : A 3. Fejezet (9) egyenletében szereplő $\gamma_{K[Ca]}$ konstans.
- **gL**: A 3. Fejezet (10) egyenletében szereplő γ_L konstans.
- **ECa:** A 3. Fejezet (7) egyenetében szereplő E_{Ca} konstans.
- **EK:** A 3. Fejezet (8) és (9) egyenletében szereplő E_K konstans.
- **Erest:** A 3. Fejezet (10) egyenletében szereplő E_{rest} konstans.
- **beta:** A 3. Fejezet (14) egyenletében szereplő β konstans.
- **B:** A 3. Fejezet (14) és (4) egyenletében szereplő *B* konstans.

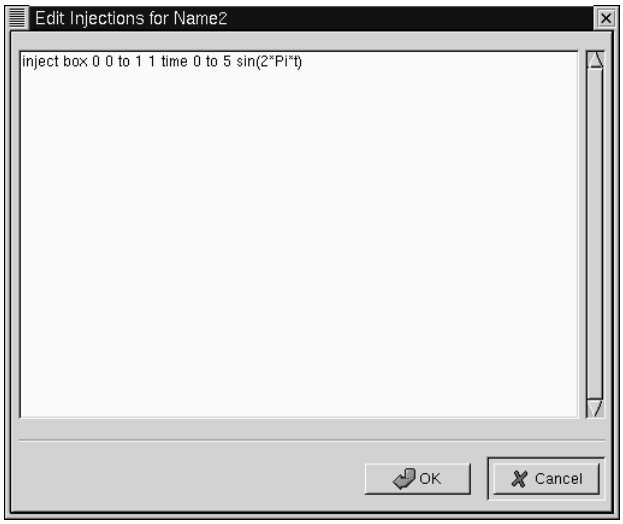
- **Theta:** A 3. Fejezet (6) egyenletében szereplő Θ konstans.
- **V*:** A 3. Fejezet (6) egyenletében szereplő V^* konstans.
- **Q:** A 3. Fejezet (6) egyenletében szereplő *q* konstans.

Cell inhibitory: A gátló (egyszerűsített) sejtekre vonatkozó konstansok.

- **Erest:** A 3. Fejezet (13) egyenletében szereplő E_{rest} konstans.
- **gL:** A 3. Fejezet (10) egyenletének γ_L konstansa gátlósejt esetén.
- **C:** A 3. Fejezet (11) és (16) egyenletének megfelelő *C* konstansa.
- **Theta:** A 3.fejezet (12) egyenletének Θ konstansa.

5.3.10 Inputs

Ezt a menüpontot csak akkor választhatjuk, ha van aktuálisan kiválasztott populációnk a főablakban. Ekkor egy új ablakot kapunk, amiben az aktuális populációhoz tartozó inputfájlt szerkeszthetjük. Az inputfájban a kívülről beadott külső áramot szimulálhatjuk, ami a 3. Fejezet (1) és (11) egyenletében, mint I_{ext} szerepel.



5.3. ábra: Az inputfájl szerkesztő ablak

Használható szimbólumok :

x,y : a tér megfelelő koordinátái

t : idő

Használható konstansok :

Egész alakban (pl. 22) vagy lebegőpontos alakban (pl 3.45 vagy 4.0).

A következőkben megadom az inputfile egy **BNF**-szerű leírását. A szögeletes zárójelek között szereplő kifejezések opcionálisak. A kulcsszavak tetszőlegesen lehetnek kis és nagybetűsek.

Az inputfájl egymás után következő sorok sorozata, egy sor formátuma pedig:

sor	=	Inject [<hely>] [<idő>] <parancs> <érték>
hely	=	Box <kifejezés> <kifejezés> To <kifejezés> <kifejezés> | Everywhere
Idő	=	Time <kifejezés> To <kifejezés>
parancs	=	Add | Sub | Be
érték	=	<kifejezés>
kifejezés	=	<konstans> | <változó> | (kifejezés) | <kifejezés> + <kifejezés> | <kifejezés> – <kifejezés> | <kifejezés> * <kifejezés> | <kifejezés> / <kifejezés> | exp(<kifejezés>) | sin (<kifejezés>) | cos (<kifejezés>) | <string> = kifejezés | <string>
változó	=	x | y | t
konstans	=	<egész> | <lebegőpontos> | Pi

Példák :

Inject Everywhere Time 1 To 15.5 Add 5

InJEct Box 0 0 to 2 2 Be sin(2*t)

Jelentés :

A <hely> megadja, hol érvényes az adott sor. **Everywhere** esetén mindenhol, **box** esetén a magadott téglalapon belül, ahol az első két szám jelöli ki a téglalap bal felső, a **to** utáni két szám a jobb alsó sarkát.

Az <idő> megadja, hogy mikor érvényes az adott sor. Az első kifejezés az idő-intervallum alsó, a **to** utáni kifejezés az intervallum felső határát adja meg.

A <parancs> jelentése, hogy hogyan értékelődjön ki az utána lévő kifejezés. Ha ez **Be** akkor az Input kiszámításánál az input értéke annyi lesz, amennyi a **Be** mögötti kifejezés, és a kiértékelő több sort nem vizsgál.

Ha ez **Add**, akkor a input értékéhez hozzáadódik az adott sorban levő kifejezés értéke, ha **Sub**, akkor levonódik.

Ha van **Be**-t tartalmazó sor, akkor az első ilyen értéke adódik vissza.

Az <érték> az fejezi ki, mi az az érték, szám az adott sorban, amit figyelembe kell venni. Ez lehet egy kifejezés, amiben az egyszerű aritmetikai műveletek, az exponenciális és a sinus, consinus függvény is szerepelhet.

Az <kifejezés> kifejtésének utolsó sorában a <string> tetszőleges, betűvel kezdődő és betűkből és számokból álló sorozat lehet. Az utolsó két szabály jelentése, hogy tetszőleges változót lehet a fenti módon definiálni, és definiálás után az értékére a nevével lehet hivatkozni. Ezzel definiálhatunk saját konstansokat.

5.3.11 Connections

A connections menüpontban az *inputs* menüponthoz hasonlóan lehet megadni a kapcsolatfüggvényeket, lásd a 3. Fejezet (23) egyenletében lévő $k_{s';s}(r', r)$ függvényt.

A program jelenlegi verziójában ez a funkció még nem támogatott, jelenleg egy, a nem specifikus diffúz kapcsolatokat modellező exponenciálisan lecsengő erősségű kapcsolatfüggvény van alapértelmezésben érvényben minden populációpárra.

5.3.12 Help menü

5.3.13 About

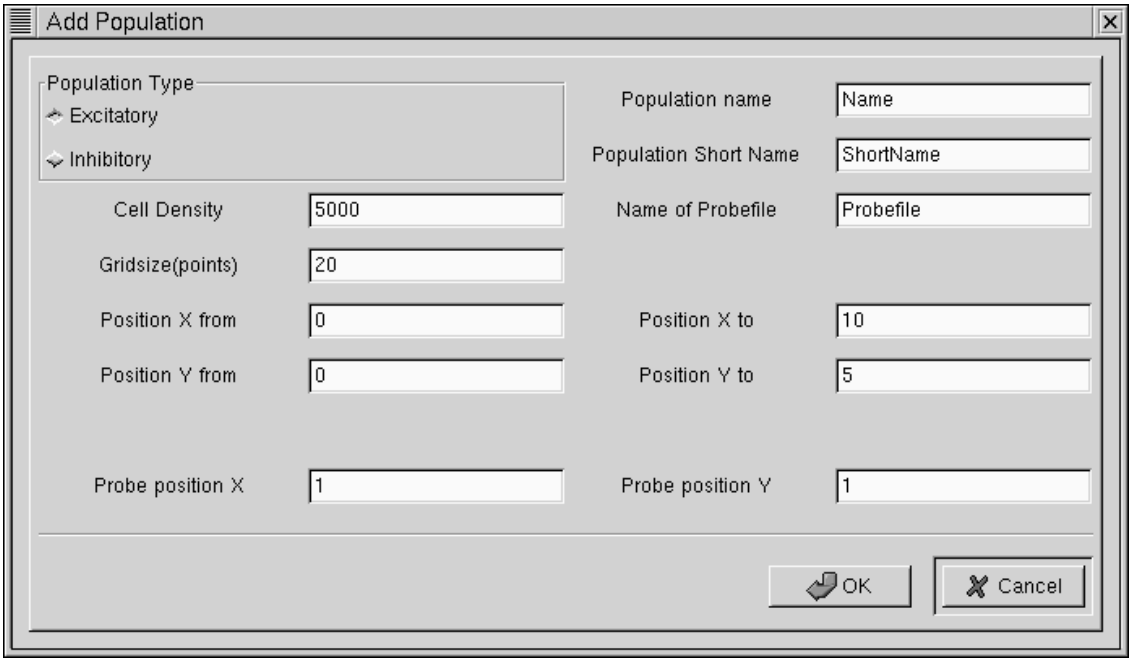
A programról rövid összefoglaló.

5.4. A toolbar

A toolbaron öt gombot találhatunk, ezek a következők:

5.4.1 Add Pop

Új populáció létrehozása lehetséges ezzel a gombbal. A gomb megnyomásakor egy dialógusablakot kapunk, amin beállíthatjuk a populáció adatait.



5.4.ábra: A populácószerkesztő ablak

- Population Type: A populáció típusa. Jelenleg kétféle lehet, a 3. Fejezetben leírt teljes, serkentő sejt modell (excitatory), és az egyszerűsített, gátló sejt (inhibitory).
- Population Name: A populáció neve. Ez a név jelenik majd meg a populáció ablakán és a populációk listájában. Egyéb fontos szerepe nincsen.
- Population Shortname: A populáció rövid neve. Ezt a nevet fogja a program fájlprefixként használni a kimeneti adatokat tartalmazó fájlok gyártásához, illetve a input és kapcsolatfüggvényeket leíró bemeneti fájlokhoz. A kimeneti fájlok neve <shortname>.out.<idő> , ahol az idő ezrendmásodpercekben van megadva. Az inputfájl neve <shortname>.inj, a kapcsolatfüggvényt leíró fájl neve <shortname1> és <shortname2> rövidített nevű populációk között pedig <shortname1>_<shortname2>.con
- Name of probefile: Minden populációhoz tartozik egy *szonda*, a populáció egy adott pozicióján az idő függvényében az aktuális aktivitást követi nyomon, és ezt egy fájlba tárolja. Az aktivitás definícióját lásd 3. Fejezet (24) egyenletét. Itt megadható ennek a fájlnak a neve.

- Cell Density: A sejtsűrűséget adja meg a populációra vonatkozóan. Jelenleg a sejtsűrűség (lásd 3. Fejezet (23)-beli Φ függvényt) konstans egy populáción belül, itt ezt a konstans értéket adhatjuk meg.
- Gridsize(points): Ebben a mezőben adhatjuk meg, hány pontra szeretnénk diszkretizálni a adott populációhoz tartozó sejtteret.
- Position (X,Y) from és to : Megadhatjuk a populáció bal felső és jobb alsó sarkát kijelölő téglalapot (milliméterben)
- Probe Position: A *szonda* pozicióját adhatjuk meg.

Az **OK** gomb megnyomásával létrhozzuk a populációt, a **Cancel** gombbal változtatás nelkül léphetünk vissza a főablakba.

5.4.2 Generate

A parméterek megfelelő beállítása után a program mintavételez a megadott függvényekből, és elkészíti a saját belső reprezentációját a modellről, azaz a modellt minden paraméterváltoztatás után újra kell **generálni**. Ezt végzi el ez a gomb.

A modell generálásakor minden populációhoz megjelenik egy ablak, amiben a populáció egyes pozicióihoz tartozó **aktivitás**értékek követhetők figyelemmel. Az aktivitás az éppen tüzelő állapotban lévő sejtek számát jelenti. Ez egyrészt grafikusan (színskálával), másrészt az egérkurzor adott pozicióra helyezésével numerikusan is leolvasható. A **SHIFT** gomb nyomva tartásával a kijelölt pozició rögzíthető.

5.4.3 Reset

A **Reset** gomb megnyomásával az éppen folyó szimuláció az aktuális modell megváltoztatása nélkül alapállapotba hozható, az idő ismét nullára áll, az állapottér a popkezdeti értékekre tér vissza.

5.4.4 Simulate

A **Simulate** gomb megnyomásával a szimuláció elindítható.

5.4.5 Stop

A **Stop** gomb megnyomásával a szimuláció ideiglenesen megállítható, majd a **Simulate** gomb újbóli megnyomásával tovább folytatható.

5.5. A populációk listája

A főablak közepét a populációk listája foglalja el. Itt felsorolva találhatjuk az éppen aktuális populációkat, és innen választhatjuk ki az éppen aktuálisat.

Láthatjuk alapvető adataikat, úgy mint a nevüket (**Name**), rövidített nevüket (**ShortName**), valamint típusukat (**Type)**, ami lehet serknető (**EXCIT**) vagy gátló (**INHIB**).

Valamelyik populációra duplán kattintva az adott populáció paramétereit szerkeszthetjük az **Add Pop** menüponthoz hasonlóan.

Valamely populáción a DEL gombot lenyomva az adott populáció törlődik.

5.6. A kimenet formátuma

A program a képernyőn látható aktivitási állapoton túl egyéb, szöveges kimenetet is gyárt, egyrészt a *szonda* kimenetét a megadott szondafájlba, másrészt minden populáció aktuális állapotát ezredmásodpercenként elmenti.

Fő szempontok voltak a kimeneti fájlok formátumának meghatározásakor, hogy azok ember számára is olvashatóak legyene, és a **GNUPlot** program számára is közvetlenül feldolgozhatóak. Ezért a kimenetek szöveges fájlok és sorokra tagolódnak.

5.6.1 A szondafájl

A szondafájl nevét a populáció paramétereinél állíthatjuk be, csakúgy, mint a *szonda* pozicióját. A szondafájlba ezután a következő formátumú sorok kerülnek:

<idő> <aktivitás>

Ezután a **GnuPlot** program **plot** parancsával a kimeneti fájl követlenül megjeleníthető.

5.6.2 A populációs fájlok

A program egy ezredmásodpercenként elmenti minden populáció aktuális állapotát a <shortname>.out.<idő> nevű fájlba, ahol <shortname> a populáció paramétereinél beállított érték, <idő> pedig az aktuális szimulációs idő. A populáció állapotához az egyes sejttérbeli diszkretizált pontok aktivitásértékei tartoznak. A fájl sorokból áll, és az egyes sorok formátuma a következő :

<x-pozició> <y-pozició> <aktivitás>

ahol az első két értékben a populációhoz tartozó diszkretizált pontok koordinátáit találjuk, az aktivitásértékben pedig a tüzelő állapotban lévő sejtek arányát.

6. FEJLESZTŐI DOKUMENTÁCIÓ

Ebben a fejezetben a program fejlesztésével kapcsolatos információk következnek, amelyek a program működésének megértéséhez és a program módosításához illetve továbbfejlesztéséhez szükségesek.

Először áttekintem a program alapvető struktúráját, majd külön tárgyalom az egysejtmodell, majd a transzformált modell egy sejttérbeli rácspontjának implementálását. Itt külön ki kell térnem a differenciálegyenletrendszer diszkretizált megoldásának problémájára. Ezután pedig a teljes, szinapszisokkal összekötött modell implementációját mutatom be, itt a párhuzamosítás problémája merül fel. Legvégül a grafikus felhasználói felület megvalósításáról szólok.

6.1. Áttekintés

A program számolásokat végző része eredetileg úgy lettt megtervezve, hogy tetszőleges dimenziószámmal képes legyen dolgozni (praktikusan 2 vagy 3 a gyakorlatban használható), így igyekeztünk a teret úgy absztrahálni, hogy az a lehető legáltalánosabb legyen. Így született a **Space** osztály, ennek belső osztályaként a **Position** osztály, amely egy térbeli pozíciót reprezentál. A **Position** osztályt a program, a bemenet és a kimenet meghatározásának kivételével átlátszatlan típusként kezeli, és rá csak a **Space** osztály műveleteit alkalmazza. A **Space** osztály és általában az osztályok pedig *Standard Template Library* **vector** osztályát használja olyankor, amikor a paraméterben megadott vagy visszatérési értékben visszaadott **Position**ok száma függ a dimenziótól.

A **Space** osztály fontosabb műveletei a következők: (mind virtuálisak, nem írom ki külön most és a továbbiakban sem)

- **double Space::dist(Position& a, Position &b)**

Két pozíció távolságát adja vissza.

- **bool Space::insideBrick(vector<Position>& corners, Position p)**

Megadja, hogy a **p** pont a **corners** által megadott n dimenziós téglalapba esik-e, ahol a téglalap definiálása a **Space** osztály feladata.

- **vector< pair<Position,double> >& Space:: GenerateMesh(vector<Position>& corners, int& n)**

A **corners** által meghatározott n dimenziós téglalapon belül generál **n** darab pontot, és visszadja ezek koordinátáit, valamint a hozzájuk tartozó, az aktuális **Space** osztály által definiált metrikával megkapott terület jellegű mennyiséget.

A **Space** osztály jelenleg definiált leszármazottai a következők:

- **Space2D**

Ez egy kétdimenziós tér euklideszi metrikával, és Lebesque-mértékkel, mint területtel. Mindkét oldalt azonos számú pontja osztja fel.

- **Space2DKeepRatio**

A fentihez hasonló, azzal a különbséggel, hogy a rácspontokat mindkét tengely mentén ugyanakkora távolságra helyezi el egymástól, ekkor nem mindig sikerül pontosan **n** darab rácspontot létrehozni. Ilyen esetben a **GenerateMesh** függvény megváltoztatja **n** értékét.

Természetesen mindezen általánosítások csak addig jók, amíg a bemeneti illetve kimeneti funkciók implementálásáig el nem jutunk, mert ekkor vagy a **Space** osztályba helyezzük ezeket a rutinokat is (ez nem túl jó ötlet), vagy kénytelenes vagyunk belenyúlni ezekbe a struktúrákba (ezt tesszük a grafikus megjelenítés és az be/kimenet esetén). A általánosság annyiban marad meg, hogy csak ezeket a be/kimenetet kezelő osztályokat kell átírni (vagy leszármaztatni), ha a a dimenziószám megváltozik.

6.2. Az egysejtmodell implementációja

A populációs modell legalsó szintjén az egysejtmodell, annak is a tüzelések közötti dinamikáját megvalósító osztály áll. Ennek ősosztálya a **Cell** osztály. Leszármazottai a

CellPyramid és a CellInhib oszállyok. Fontosabb műveleteik a következők (természetesen virtuális műveletek)

- **void step**(**double** dt, **int** injno, **int*** inj_type, **double*** inj_val);

  Végrehajt egy időlépést. A bemenő paraméterek az aktuális időlépés(dt), és a sejtbe adott injekciók, azaz valójában a konduktanciaváltozások összege, inj_no az injekciók száma, inj_type az egyes injekciók típusa, ami lehet CELL_PYRAMIDAL, CELL_INHIB vagy CELL_STIMULUS a preszinaptikus sejt típusától függően. A CELL_STIMULUS speciális sejttípus, a kívülről, a kísérletező által beinjektált áramot jelöli, az inj_no a injekció erőssége. (3. Fejezet (1), (2) és (7)-(10) implementációja)

- **double prob_fire()**

  Ez a függvény megadja, mekkora valószínűséggel tüzelt a sejt a legutóbbi lépés alkalmával. (a 3. Fejezet (5),(6) implementációja)

- **double v**

  Az aktuális membránpotenciál.

- **double ca**

  Az aktuális kálciumtartalom.

Említésre méltó még a **CellPyramid** osztály, ami a serkentő sejtek implementációja. Ugyanis itt a modell generálásakor már előre ismerünk minden lényeges paramétert, így nem kell az (1),(2),(7)-(10) egyenleteket minden lépésben végigszámolni, hanem az első alkalommal kell csupán egy mátrixot feltölteni, ahol is minden állapottérbeli rácsponthoz letároljuk, hogy a **step** művelet milyen állapotba viszi át. Ezt a mátrixot készíti el a CellPyramid osztály **init_lookup_table**() művelete. Természetesen a modell minden paraméterének megváltozásakor ezt a táblát újra kell generálni.

6.3. Egy rácspont implementációja

Egy sejttérbeli rácspontot a **Disc** osztály és annak leszármazottai, a **DiscPyramid és DiscInhib** osztályok implementálnak. A megkülönböztetés optimalizációs szempontból szükséges, hiszen a gátlósejteket csupán egyetlen paraméter jellemzi, így csupán egyetlen dimenziós eloszlásfüggvénnyel kell dolgoznunk, ami kisebb memória és számításigényt jelent.

Az osztály fontos műveletei a következők:

- **void Disc(int mode)**

A konstruktorban megadható, hogy milyen legyen az eloszlásfüggvény a kezdeti időpillanatban. Ez jelenleg lehet **MODE_STABLE**, ami az egysejtnek felhasználói dokumentációban említett stabil fixpontjába helyez egy Dirac-deltát, azaz minden sejtet ilyen állapotból indít, illetve **MODE_UNIFORM,** ami az egyenletes eloszlásból indítja a rendszert.

- **void step(int injno, int* inj_type, double* inj_val)**

Az egysejtmodell implementálásának leírásánál megadotthoz hasonlóan a rácspontba érkező injekciókat adhatjuk meg.

- **void dumpstate(FILE f, int injno, int* inj_type, double* inj_val)**

Kimenti az aktuális rácspont értékét egy szöveges fájlba. Csak debuggolási célra.

- **double get_firgs()**

A rácspont aktuális *aktivitásértékét* adja vissza. (lásd 3.5. Fejezet, (24) egyenlet).

A függvény neve történeti okokból **get_firgs**.

- **double get_ltp() és get_ltd()**

A programmal lehetséges nyomon követni a tanulási folyamatokat. Az **LTP** jelentése Long Term Potentiation, ami a szinapszisok hosszú távú erősítését jelenti. Az **LTD** Long Term Despression, ami a szinapszisok hosszú távú gyengülésének neve.

Megfigyelték, hogy LTP a szinapszis a pre- és posztszinaptikus sejtek nagy frekvenciájú, LTD pedig alacsonyabb frekvenciájú együttes periodikus ingerlésére alakul ki. Így ezeket, mint az állapottér nagy, illetve közepes kálciumértékű tartományait vesszük figyelembe, hiszen tüzeléskor kálcium áramlik a sejtekbe. A fenti

két függvény az ebből származtatott értéket adja vissza, amire a pre- és posztszinaptikus rácspont figyelembevételével tanulási algoritmus alkalmazható. Jelenleg ez a modell felsőbb rétegeiben még nem implementált.

Szót kell ejtenünk még az alkalmazott diszkretizációs módszerről is, hiszen a fenti osztály valójában egy parciális differenciálegyenletet hivatott megoldani.

6.4. A differenciálegyenlet megoldásának közelítése

A 3.4. Fejezet (14) egyenletének numerikus közelítését a programban a következőképpen oldottuk meg.

Látható, hogy ez az egyenlet egy másodfokú parciális differenciálegyenlet, amelyben forrás és nyelő tagok is vannak. Ez hasonlít a fizikában ismert Fokker-Plack típusú egyenletekre, de itt a tagokra vajmi kevés megszorítást tehetünk. Analitikus megoldást nyilván nem várhatunk, tehát valamilyen numerikus közelítő módszert keresünk. Az irodalomban a fenti problémára alkalmazható módszert nem találtunk, így a Grőbler, Barna és Érdi (1998) cikkben találhatóhoz hasonló módszerrel, de annak optimalizált változatával közelítjük a megoldást. Mindenekelőtt tisztázni szeretném, hogy a módszer több jelentős elhanyagolást vezet be:

- Feltesszük, hogy a folytonos eloszlásfüggvény diszkrét pontokban vett diszkrét értékekkel (Dirac-delta függvények összegével) approximálható.
- Feltesszük, hogy konstans diszkrét időlépésenként approximálhatjuk a megoldást.
- Feltesszük, hogy a Dirac-delták összegére alkalmazva a dinamikát egy lépésben, az így kapott új eloszlásfüggvény közelíthető azzal az eloszlásfüggvénnyek, amit úgy kapunk, hogy az egyes Dirac-deltákra alkalmazzuk a dinamikát, majd az így kapott eloszlásfüggvényeket összegezzük, és diszkrét pontokban mintavételezzük, hogy ismét diszkét eloszlást kapjunk.
- Tudjuk, hogy egy Dirac-delta kiinduló eloszlásra ($g(u,\chi,0) \equiv \delta(u,\chi)$) alkalmazva, csak a 3. Fejezet (14) egyenletének térbeli diffúziós, azaz csak másodrendű térbeli deriváltakat tartalmazó változatát, akkor annak megoldása analitikusan lehetséges:

(1) $$g(u,\chi,t) = \frac{1}{2\cdot\pi\cdot D_u \cdot D\ \cdot(\Delta t)^2}\cdot e^{-\frac{1}{2}\cdot\left(\frac{u^2}{(D_u\Delta t)^2}+\frac{\chi^2}{(D_\chi\cdot\Delta t)^2}\right)}$$

Feltételezve, hogy az eredeti egyenlet közelíthető mint független diffúziós és sodródási (elsőrendű tagokból származó) folyamatok összege, akkor a megoldás a következő egyenlettel közelíthető a ($g(u,\chi,0)\equiv\delta(u-u_0,\chi-\chi_0)$) kezdeti állapotból induló rendszer:

(2) $$g(u,\chi,t) = \frac{1}{2\cdot\pi\cdot D_u \cdot D\ \cdot(\Delta t)^2}\cdot e^{-\frac{1}{2}\cdot\left(\frac{(u-u\ -u_d)^2}{(D_u\Delta t)^2}+\frac{(\chi-\chi_0-\chi_d)^2}{(D_\chi\cdot\Delta t)^2}\right)}$$

Ahol u_d és χ_d az elsőrendú térbeli deriváltak, a sordródás (*drift*) által meghatározott elmozdulás Δt idő alatt.

Mit jelent ez valójában ? Azt, hogy a (u_0, χ_0) kezdeti állapotból indítva egy Dirac-deltát, ami *m* darab sejtet reprezeltál, ezt a dinamika egy (u_0+u_d, $\chi_0+\chi_d$) várható értékű és ($D_u\Delta t$, $D_\chi\Delta t$) szórásnégyzetű Gauss-eloszlásba viszi át.

- Feltesszük azt is, hogy a fenti (2) egyenletben a Gauss-eloszlást közelíthetjük minden rácspontra és minden időlépésre egy olyan eloszlással, ami megtartja a annak első két momentumát, azaz a várható értékét és a szórásnégyzetét.
- Figyelnünk kell arra is, hogy az eloszlásfüggvény nem vehet fel negatív értékeket.

Találni akarunk tehát egy olyan módszert, amivel minden lépésben a lehető legkevesebb rácspont módosításával a fenti kritériumoknak meg akarunk felelni.

Ezt úgy tesszük, hogy minden rácspontra meghatározzuk, hogy sodródási tagok, ami megfelel az egysejtmodell dinamikájának, milyen állapotba, azaz melyik rácspontba vinné át az adott rácspontot. Ezután az általunk nyilvántartott diszkrét eloszlásfüggvény abban a kiinduló récspontban felvett értékét szétosztjuk a pont és annak nyolc szomszédja között úgy, hogy a fenti kritériumoknak megfeleljünk. Először oldjuk meg a feladatot egy dimenzióra, azaz tegyük fel, hogy egy egydimenziós normális eloszlást kell közelítenünk tehát az egységet szétosztanunk -Δv, 0 és Δv pontok között, azaz

hozzájuk az u_1, u_2, u_3 súlyokat rendelnünk úgy, hogy *a* legyen a várható érték és $D_u \Delta$t legyen a szórásnégyzet, azaz:

(3) $u_1 + u_2 + u_3 = 1$

(4) $-\Delta v u_1 + \Delta v u_3 = a$

(5) $u_1(a+\Delta v)^2 + u_2 a^2 + u_3(a-\Delta v)^2 = D_u \Delta t$

Amely egyenleteket megoldva kapjuk, hogy:

(6) $u_1 = \frac{D_u \Delta t + a^2 - a\Delta v}{2(\Delta v)^2}$ ezzel kifejezve:

(7) $u_2 = 1 - \frac{a}{\Delta v} - 2u_1$

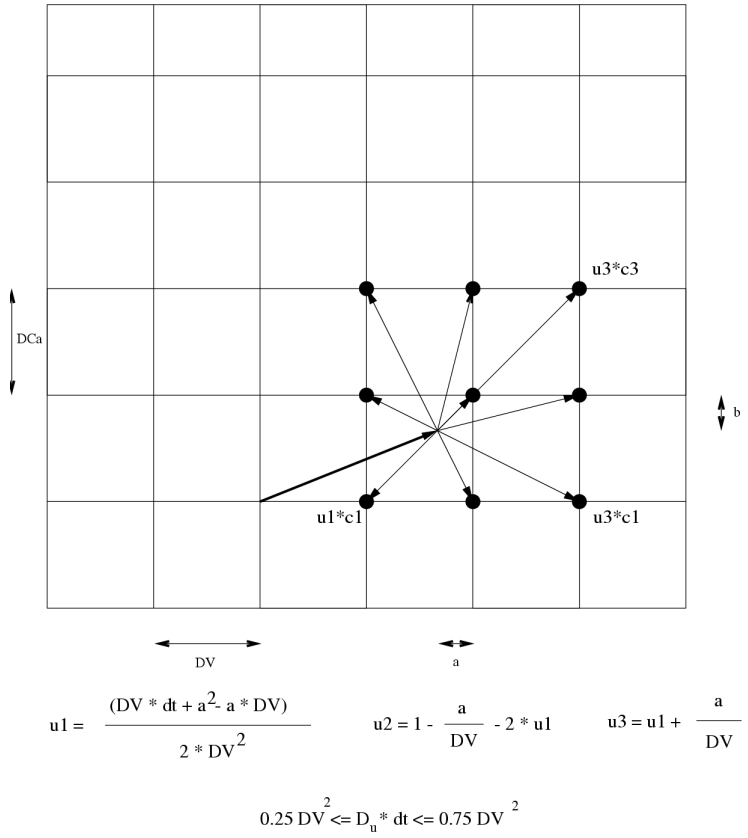
(8) $u_3 = u_1 + \frac{a}{\Delta v}$

Meg kell oldanunk még, hogy az így kapott u_1, u_2, u_3 súlyok ne vehessenek fel negatív értékeket, hiszen eloszlásfüggvénnyel dolgozunk. Meg tudunk adni azonban korlátozást Δv-re, hogy ez teljesüljön. Ez a korlátozás a következő:

(9) $\frac{1}{4}(\Delta v)^2 \le D_u \Delta t \le \frac{3}{4}(\Delta v)^2$

Ezt alkalmazva a megfelelő diffúziós konstanssal a kálciumtengely mentén is, az így kapott kilenc pontba való eloszlás megfelel a fenti kritériumoknak.

A **Disc** osztály **step** művelete ezt végzi.



6.1.ábra: A **Step** osztály által végzett szétosztás

6.5. A sejttér implementációja

Az alábbiakban felsorolom a sejjtér implementációjához tartozó osztályokat, es ahol szükséges, további magyarázatot így fűzök hozzájuk:

6.5.1 AlphaFunction

Ez az osztály implementálja a 2.3. Fejezetben leírt szinaptikus áttevődést leíró függvényeket. Műveletei:

- **AlphaFunction(int from, int to)**

  A konstruktornak paraméterkéne megadandó a forrás és a célpopuláció típusának azonosítója.

- **double Value(double tbegin, double tend, double str)**

  Megadja a tbegin és a tend időpillanat közti integrálját az str erősségű alfafüggvénynek.

- **double Length()**

  Megadja az alfafüggvény figyelembeveendő hosszát. Elméletileg az alfafüggvények a teljes valós számegyenesen értelmezettek, de szimulációs célokra ez nyilván megfelelő, így megadunk egy felső korlátot, ameddig még figyelembe akarjuk venni az adott szinapszis hatását.

- **operator==**

  Definiáljuk az egyenlőség operátort. Ez egyenlő két azonos típusú populáció közötti alfafüggvények esetén.

6.5.2 StandardAlphaFunction

Ez az osztály az **AlphaFunction** leszármazottja, a 2.3.1 Fejezetben definiált alfafüggvény-típus. Műveletei megegyeznek az **AlphaFunction** műveleteivel.

6.5.3 DualExpAlphaFunction

Ez az osztály az **AlphaFunction** leszármazottja, a 2.3.2 Fejezetben definiált dupla exponenciális alfafüggvény-típus. Műveletei a konstruktor kivételével megegyeznek ősének műveleteivel

- **DualExpAlphaFunction(int from, int to, double gx, double t1, double t2, double strfc)**

  Ahol a *gx* a maximális konduktancia a szinapszison, *t1* és *t2* a 2.3.2 Fejezetben szereplő τ_1 és τ_2 konstansoknak felelnek meg, *strfc* pedig egy skálázó faktor, értéke általában 1. A programban ezt az alfafüggvény-típust használjuk mindenütt.

6.5.4 DiracAlphaFunction

Ez egy csak tesztelési céllal betett, Dirac-delta függvénynek megfelelő alfafüggvény. Műveletei, mint az AlphaFunction osztálynak.

6.5.5 ConnectionFunction

Ez az osztály implementálja a 3.5. Fejezetben említett kapcsolatfüggvényt. Műveletei:

- **ConnectionFunction(Population& from, Population& to)**

  Létrehoz egy, a *from* és a *to* populáció között érvényes kapcsolatfüggvényt.

- **Strength(Position& a, Position& b)**

  Megadja az *a* és *b* pozíció közti kapcsolaterősséget.

- **Delay(Position& a, Position& b)**

  Megadja az *a* és *b* pozíció közti axonikus késleltetést.

- **Population& PopulationFrom()**

  Megadja a kapcsolatfüggvényhez tartozó forráspopulációt.

- **Population& PopulationTo()**

  Megadja a kapcsolatfüggvényhez tartozó célpopulációt.

6.5.6 ExponentialCF

Ez az osztály a **ConnectionFunction** osztály a programban jelenleg használt leszármazottja. Az általa felüldefiniált műveletek:

- **ExponentialCF(Population& from, Population& to, double parameter, double speed)**

  Létrehoz egy exponenciális lecsengésű kapcsolatfüggvényt, a *parameter* az exponenciális lecsengés paramétere (a kitevő szorzója), a *speed* az ingerületterjedés sebessége, ami csak a két pont távolságától függ, ahol a távolságot a **Space** osztály definiálja.

6.5.7 DensityFunction

A sűrűségfüggvényt reprezentáló osztály. Minden populációhoz tartozik egy sűrűségfiggvény, ami megadja az adott populáció sűrűségét minden pontban.

Lehetséges nem-konstans sűrűségfüggvényeket is definiálni, de jelenleg csupán konstans sűrűségfüggvény van implementálva.

- **DensityFunction(double h)**

  Minden pontban a *h* sűrűséget visszaadó sűrűségfüggvényt hoz létre.

- **DensityAt(Position& p)**

  Megadja a *p* pozícióban a populáció sejtsűrűségét.

6.5.8 StimulusFunction

Ez az osztály kezeli a modellnek adott inputokat. Lényeges művelete a **Value** függvény.

- **double StimulusFunction(Population& p)**

  Létrehoz egy *p* populációhoz tartozó stimulusfüggvényt.

- **double Value(double time, Population& pop, Element& elem)**

  A **time** az aktuális időt adja át. A függvény az külső inger értékével tér vissza a megadott pozícióban a megadott időpillanatban. Az **Element** osztály leírását lásd később, most elég annyit tudni róla, hogy meghatároz a **pop** populációban egy pozíciót.

- **void Refresh()**

  Ezt a függvényt akkor kell meghívni, ha a modell paraméterei megváltoznak.

6.5.9 ConstantStimulus

Ez a leszármazottja a StimulusFunction osztálynak egy konstans téglalapon belül egy konstans ingert reprezentál.

- **ConstantStimulus(Population& p, vector<Position> corners, double str)**

Ez egy, a *p* populációhoz tartozó, *corners* által meghatározott n dimenziós téglatesten belül értelmezett *str* erősségű stimulusfüggvényt hoz létre.

6.5.10 FileStimulus

A StimulusFunction ezen leszármazottja a stimulusfüggvény értékeit egy külső, ún. inputfájlból veszi. Az osztály általunk használt leszármazottja (**FileStimulus**) valójában meghívja a **yacc** grammatikai elemzőt, és általa adja vissza az értéket, a nyelvtan és a lexikális egységek leírása az **input.y és input.l** fájlokban találhatók, a hozzájuk tartozó dokumentáció a felhasználói dokumentáció Edit menüjének Input menüpontjának leírásánál található.

6.6. Az Element osztály

A sejttér implementációjának lealsó szintje az **Element** osztály. Minden sejttérbeli rácsponthoz tartozik egy **Element** osztály, ami a a sejtérhez tartozó információkat adminisztrálja, és másik fontos feladata, hogy a párhuzamosítást is kezeli, ez utóbbiról bővebben a következő fejezet szól.

Az Element osztály fontosabb műveletei:

- **Element(Position& a, double area_value, double density_value, int celltype)**

  Létrehoz egy, az *a* pozícióban lévő, *area_value* területet reprezentáló, *density_value* sejtsűrűséggel rendelkező, *celltype* sejttípusú (lásd Cell.h) Elementet, ami a rács egy pontját reprezentálja.

  A konstruktor meghívása egyben a kliensgépekkel való kapcsolatfelvételt, és a **PVM** környezetbe való belépést is jelenti. Azaz a konstuktor hívása sikertelen lesz, ha a **PVM** környezet nincs megfelelőel beállítva.

- **Element(const Element& e)**

  Az Element osztály copy-konstruktora.

- **DumpState()**

Ez hibakeresési céllal van a programban, kiírja egy fájlba az Element aktuális állapotát.

- **SetDensity(double density)**

  Beállítható az Elementhez tartozó sejtsűrűségérték.

- **void Step()**

  Az aktuális kapott adatokkal elvégeztet egy lépést a **Disc** osztállyal.

- **void Invalidate()**

  Mivel a párhuzamosítást az Element osztály végzi, és a számolás aktuális adatai egy távoli gépen vannak, így szükségünk van annak nyilvántartására, hogy a lokális gépen lévő adat mikor tekinthető frissnek, és mikor van szükség az aktuális adatok lekérésére a távoli (kliens) géptől. Ez a művelet azt jelzi az Element osztálynak, hogy az általa tárolt adat elavult, és a következő állapotlekérdező művelet (GetAcitvity()) esetén a kliensgéphez kell fordulni az aktuális adatokért.

- **void Add(int poplabel, double delay, double strength, double factor)**

  A *poplabel* által mutatott populációból kap egy *delay* késéssel, *strength* erősségű kapcsolaton keresztül érkező, *factor*-ral szorzott gerjesztést. A *factor*-ban adjuk át a forráspopuláció megfelelő pozícióján az aktivitásértéket.

- **double GetActivity()**

  Az aktuális időlépésbeli aktivitásértéket adja vissza a hozzá tartozó rácsponton.

- **double GetPrevActivity()**

  Az előző lépésbeli aktivitásértéket adja vissza. Erre a majd később említendő párhuzamos optimalizálás miatt van szükség.

- **double GetSpikes()**

  Ez a függvény az aktivitásértéket adja vissza, megszorozva az Elementhez tartozó területtel, ami valójában a pontból kibocsájtott spike-ok "mennyiségét" reprezentálja.

- **Position& GetPosition()**

Visszaadja az Element pozícióját a sejttérben.

- **void ClearEffects()**

  Törli az Add művelettel beállított gerjesztéseket.

- **void SetMaxDelay(int poplabel, double maxdelay)**

  Beállítja a *poplabel*-hez tartozó populáció Pipe-jainak maximális hosszát. Erre azért van szükség, mert a Pipe osztályok a kliensgépeken futó programban implementálódnak, és azoknak szükségük van erre az adatra.

6.7. Párhuzamosítás

A programban a párhuzamosítás, illetve a **PVM** könyvtár használata az **Element** osztály szintjén jelentik meg. A programban centralizált architektúrát használtunk, ami azt jelenti, hogy egy központi (*frontend*) gépen fut a grafikus kezelői felület és a főprogram a legfelső szinttől egészen az Element osztályig. Az Element osztályhoz tartozó interfész a kliensgépeken a **client.cpp** fájlban található.

Minden, a hierarchiában az **Element** osztály alatt található osztály implentációja a kliensgépeken fut, ilyen módon az Element osztály alatt és felett lévő osztályoknak nem kell törődniük a párhuzamosítás kérdésével, számukra a párhuzmosítás teljesen *átlátszatlan*.

A párhuzamosítással kapcsolatos definíciók a **mypvm.h** fájlban találhatók.

A következő *tag*eket definiáljuk a **PVM** üzenetek számára:

- TAG_INIT

  Ezt a tagot a főgép küldi a klienseknek, a program kezdetén kettőt is, az elsőben utazik a populáció típusa, és hogy milyen kezdeti feltétellel (egyenletes vagy Dirac-delta eloszlással) induljon a szimuláció. A második INIT tagban utaznak az aktuálisan érvényes paraméterei a modellnek.

- TAG_DESTROY

Ezt az üzenetet a főgép küldi a klienseknek, és arra utasítja azokat, hogy fejezzék be tevékenységüket.

- TAG_STEP

  Ezt az üzenetet a főgép küldi a kliensgépeknek, és benne az aktuális, a célrácsponthoz tartozó injekcióértékekkel.

- TAG_TELL_FIRGS

  Ezt az üzenetet a kliensgépek küldik a főgépnek, amikor befejezték egy időlépés számolását, és benne küldik az aktuális aktivitás és LTP valamit LTD értékeket. A tag neve történeti okokból az, ami.

- TAG_DESTROY_ANSWER

  Ezt az üzenetet a kliensgép küldi a főgépnek, és azt jelzi, hogy megkapta a DESTROY üzenetet és kilép.

- TAG_DUMP_STATE

  Ezt az üzenetet a főgép küldi a kliensgépeknek, és arra utasítja őket, hogy írják ki az aktuális állapotukat egy fájlba. (lásd a Disc osztály dumpstate műveletét)

- TAG_SETMAXDELAY

  Ezen üzenet adata a kliensgépeket lévő Pipe maximális hosszát határozza meg.

6.7.1 A párhuzamosítás hatékonysága

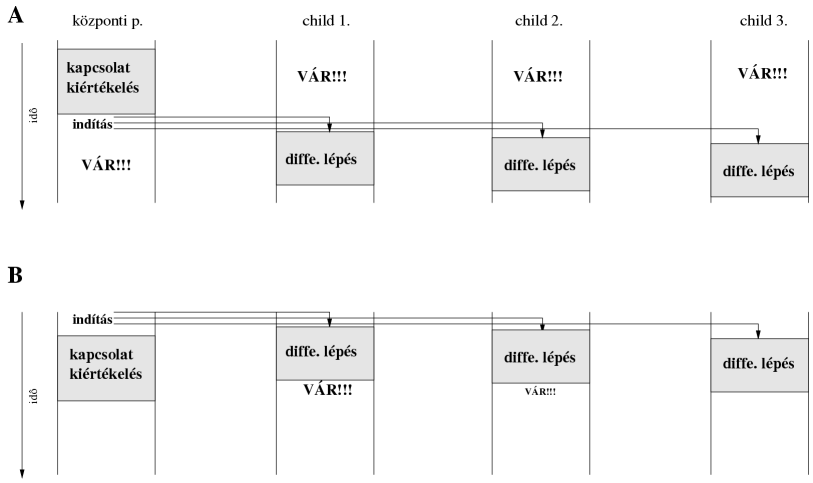
A függvényhívások egyszerű átültetése PVM kommunikációvá nem hatékony, a clustergép teljes rendelkezésre álló számítási kapacitásának 60-70%-a marad így kihasználatlan. Ahhoz, hogy hatékony eredményt érjünk el, a következő módszereket alkalmaztuk:

- Nem a frontend gépen számoljuk a alfafüggvények hatását a rácspontokra, hanem minden rácspont minden gerjesztését elküldjük neki, mint PVM üzenetet, és a **child.cpp** számítja így ki minden időpillanatban az aktuális bemenetet. Ez nagyobb

kommunikációval jár, de a főgép kisebb terheleésével, ami centralizált architektúránál lényeges szempont.

- *Aszinkron* architektúrát választottunk, ami azt jelenti, hogy a kliensgépek mindig a következő időlépésben érvényes aktivitást számolják, és az **Element** osztály eltárolja az aktuálisat. Ilymódon a kliensgépeknek nem kell várniuk, amíg a főgép szállítja számukra a szükséges adatokat az aktuális gerjesztésekről. Azonban ezt csak azzal a megszorítással tudjuk megtenni, hogy a szinaptikus késés (**dentritic_delay)**, amit a Felhasználói dokumentáció **Edit** menüjének **Parameters** részében állíthatunk, legalább egy időlépés hosszú kell, hogy legyen. Ez egyébként biológiailag reális feltevés.

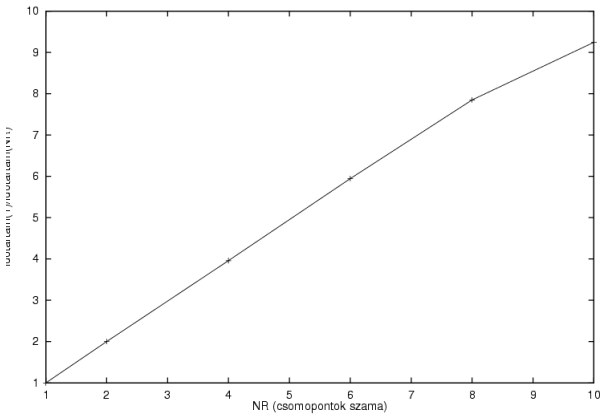
  Az aszinkron architektúra működését a következő ábra mutatja:



6.2.ábra: A szinkron(A) és aszinkron (B) működés összehasonlítása.

A fenti optimalizációkkal a teljesítmény az alkalmazott kliensgépek számával majdnem lineárisan növekszik, és 16 gép esetén a kommunikáció még mindig elégge alacsony, a hálózati forgalom kihasználtsága kb. 1.5%-os.

A párhuzamosítási algoritmus skálázhatóságát a következő ábra mutatja:



6.3.ábra: A számítási teljesítmény (futási idő reciproka)
változása a clusterbe kapcsolt gépek számának függvényében.

6.8. A Population osztály

A **Population** osztály egy absztrakt osztály, ami a populáció adatait foglalja magában, ami már térben kiterjedt. Egy Population sok elemet tartalmaz, amik a rácspontokat reprezentálják. Ezek az Elementek lesznek a leszármazottakban, de itt, az absztrakt osztály szintjén még nincs definiálva. Alapvető műveletei:

- **Population(vector<Position> corners, int gridsize, DensityFunction& df, int celltype, string popname, string short_popname)**

  Létrehoz egy *corners* sarokpontok közötti térben létező, *gridsize* darab rácspontra felbontott, *df* sűrűségfüggvényű, *celltype* típusú sejteket tartalmazó (lásd Cell.h), *popname* nevű és *short_popname* rövid-nevű (lásd a Felhasználói dokumentáció **Add Pop** menüjét)

- **Population::iterator begin()**

  Visszaadja a populációhoz tartozó első elemre mutató iterátort (lásd C++ Standard Template Library)

- **Population::iterator end()**

Visszaadja a populációhoz tartozó, utolsó elem után mutató iterátort (lásd C++ Standard Template Library)

- **Element& back()**

  Megadja a populációhoz tartozó utolsó elemet.

- **void Output()**

  Kimenti a populációhoz tartozó *rövid-név* által megadott fájlba a megfelelő adatokat (lásd 5.6.2. Fejezet)

- **int CellType()**

  Megadja a populáció sejttípusát.

- **void Generate()**

  Létrehozza a populációhoz tartozó elemeket.

- **void UnGenerate()**

  Törli a populációhoz tartozó elemeket.

6.8.1 A PopulationDirect osztály

A **PopulationDirect** osztály a **Population** osztály leszármazottja. Amit megváltoztat, csupán annyi, hogy specifikálja, hogy a Populationhoz tartozó elemeket Elementek reprezentálják.

6.8.2 A PopulationProbe osztály

Ez az osztály a **PopulationDirect** osztályból származtatható. Egy szondapozíció adható meg, aminek állapotát a program folyamatosan nyomon követ, és a konfigurációban megadott szondafájlba az aktivitásértékeket kiírja az Output műveletre. A műveletek szemantikája egyébként azonos, a konstruktor szignatúrája változott:

- **PopulationProbe(vector<Position> corners, int gridsize, DensityFunction& df, int celltype, string popname, string short_popname, Position probeposition, string probefilename)**

  Ahol új a *probeposition*, ami a szonda helyzetét, és a *probefilename,* ami a szondafájl nevét adja meg.

6.8.3 A Slice osztály

Az populációk grafikus reprezentációját a Slice osztály végzi. Ez az osztály az X11 kiszolgálón hoz létre egy ablakot, és itt nyomon követhetők a populáció elemeinek aktivitásértékei. Közvetlenül nem használja a populáció adatait, a populáció használja őt, mégis itt említem meg, mert logikailag ide tartozik. Műveletei a következők:

- **void Slice(const char* name, int nox, int noy)**

  Létrehoz egy Slice példányt, s ezzel együtt egy ablakot, aminek neve *name* lesz, és egy nox doboznyi széles és noy doboznyi magas rács-grafikát helyez bele.

- **void Color(int x, int y, double val)**

  Az *x,y* koordináták által meghatározott doboz színét változtatja *val* színűre, ahol *val* egy 0 és 1 közötti szám, és a folytonos színskálán egy színt reprezentál.

- **void Color(int index, val)**

  Az *index* indexű doboz színét változtatja *val* színűre.

- **void Update()**

  Ezt a műveletet akkor kell meghívni, ha azt akarjuk, hogy az ablak alában lévő pro*gressbar* mozogjon, hogy a felhasználó lássa, hogy a számítás folyamatban van.

- **void SetStatusBarText(const char* text)**

  Az ablak *statusbar*-jának szövegét állíthatjuk be.

- **void SetWatched(int x, int y)**

Beállíthatjuk, milyen pozíción lévő aktivitásértékek kerüljenek kijelzésre az ablak alján.

- **double GetStoredColor(int x, int y)**

  Visszaadja a megadott koordinátára utoljára beírt szín-értéket.

- **double GetStoredColor(int index)**

  Visszaadja a megadott indexre az utoljára beírt szín-értéket.

- **int Index(int x, int y)**

  Visszaadja a megfelelő koordinátákhoz tartozó indexértéket. Belső konverzióhoz használatos.

- **void GetCoords(int x, int y, int& goodx, int &goody)**

  Konverziót végez a az általunk definiált koordináták és a **GNOME/Gtk+**-ban érvényesek között.

- **int Width()**

  Megadja a dobozokban mért szélességét az ablaknak.

- **int Height()**

  Megadja a dobozokban mért magasságát az ablaknak.

6.9. A PopModell osztály

A teljes modell implementációjáért a **PopModell** osztály felelős. Rajta keresztül lehet új populációkat, inputfüggvényeket és kapcsolatfüggvényeket a rendszerhez rendelni, illetve onnan elvenni. Magának az osztálynak az adataihoz tartozik minden, a modellhez tartozó osztály és adat. Azaz több modell futtatása is lehetséges egyidőben, több PopModell osztályt példányosítva.

- **PopModell()**

  Létrehoz egy modellt.

A **PopModell** osztály két jelentős függvénye még a modell generálásával kapcsolatos.

- **void PopModell::Generate()**

  Ezzel lehetséges a modell ideiglenes belső adatstruktúráit generálni.

- **void PopModell::UnGenerate()**

  Az ideiglenes adatstruktúrák felszabadítása lehetséges ezzel a művelettel.

- **void Step()**

  Végrehajt egy szimulációs lépést.

- **void AddPopulation(vector<Position> corners, int gridsize, DensityFunction density, int celltype, string popname, string pop_shortname, Position probeposition, string probefilename)**

  A függvény pontosan a megadott paraméterekkel létrehoz egy Population példányt (azaz meghívja annak konstruktorát)

- **void ClearPopulations()**

  Ezzel a művelettel törölhetjük ki a modellben lévő populációkat.

- **void DeletePopulation(Population& p)**

  Ezzek a művelettel lehet egy konkrét populációt törölni a modellből.

- **void PopModell::AddStimulus(Population* pop)**

  Ezzel a művelettel rendelhetünk egy populációhoz egy stimulusfüggvényt.

- **AddExponentialCF(int pop_from, int pop_to, double param, double speed)**

  Létrehoz egy exponenciális kapcsolatfüggvényt a *pop_from* és a *pop_to* populációk között *param,speed* paraméterekkel meghívva annak konstruktorát.

- **bool Empty()**

  Visszaadja, hogy a modell üres-e, ami akkor igaz, ha nincs benne populáció.

6.10. Az App osztály

Az App osztály implementálja a grafikus felhasználói felületet.

A kezelői felület implementációja legnagyobbrészt az **App.cpp** fájlban történik, de részei találhatók még az **input.cpp** és **clist.cpp** fájlokban is. Előbbi a stimulusfájl szerkesztéséért felelős, utóbbi a populációk főablakban megjelenő listájának kezeléséért. Ezen kívül a **signal.c** fájl a szignálok elkapásával foglalkozik, azért, hogy a *CTRL+C* billentyűkombináció lenyomása esetén biztosítsa, hogy a kliensgépeken is leálljanak a kiszolgálóprogramok.

Az **App** oszály műveletei:

- **App()**

  Létrehoz egy applikációt, ami a **GNOME** terminológia szerint egy főablakot jelent.

- **GtkWidget* Widget()**

  Megadja applikációhoz tartozó GNOME widgetet.

- **PopModell* Modell()**

  Visszaadja az applikációhoz tartozó modellt.

- **AppStatus GetStatus()**

  Visszaadja az applikáció állapotát. Ez lehet Inakív, Generált, Futó. Ezek az állapotok az App.h-ban, mint **enum** definiáltak.

- **void SetStatus(AppStatus status)**

  Beállítható az applikáció állapota.

- **void SetSaveFileName(const char* filename)**

  Beállítható, mi legyen az alapértelmezett file a Save műveletkor.

- **Void SetLoadFilename()**

  Beállítható, mi legyen az alapértelmezett file a Load műveletkor.

- **GtkWidget* Selection()**

Visszaadja, melyik az éppen kiválasztott populációhoz tartozó widget az ablakban.

- **void SetSelection(GtkWidget* w)**

  Beállítható, melyik legyen a kiválasztott populáció a főablakban.

- **GtkWidget* Clist()**

  Visszadja az applikációhoz tartozó listát, ami egy Gtk+ widget. Ez a lista tartalmazza az aktuális populációkat, és ez látható a főablak közepén.

- **void Save()**

  Elmenti a modell paramétereit az alapértelmezett néven. Ha ilyen nincs, akkor ekvivalens a SaveAs() művelettel.

- **void SaveAs()**

  Elmenti a modellt egy, a feljövő fájlválasztó-ablakban a felhasználó által kiválasztott névre. Az így kiválasztott név lesz az alapértelmezett név.

- **void Load()**

  Betölt egy régebben kimentett modellt, a fájl egy feljövő fájlválasztó-ablakban választható ki.

Mivel a GNOME/Gtk+ alapvetően esemény-, szignál- és callback-orientált, így az **App** oszály műveletei mellett érdemes még megemlíteni a felhasznált callbackeket is.

- **void clist_key_press_event_cb(GtkWidget* w, GdkEventAny* e, gpointer data)**

  Ez a callback akkor hívódik meg, ha a felhasználó a főablakban kiválaszt egy populációt, és megnyom egy gombot. Ha ez a DEL gomb, akkor az adott populáció törlődik mind a applikáció listájából, mint a modellből.

- **void clist_selected_cb(GtkWidget* widget, int row, int column, GdkEventButton* e, gpointer data)**

  Ez a callback meghívódik, ha a felhasználó kiválaszt egy populációt a főablak listájából. Ha a felhasználó kétszer kattintva választotta ki a populációt, akkor ez létrehoz egy dialogusablakot, ahol a populáció adatai szerkeszthetők.

- **void input_cb(GtkWidget* w, gpointer data)**

  Ez a callback az Parameters / input menüpont választásakor hívódik meg, ha az applikáció nincs Futó állapotban. Létrehoz egy dialógusablakot, amibe egy GtkText widgetet helyez, amiben a populációhoz tartozó injekciós fájl szerkeszthető. Ha az OK gombra kattintunk, akkor az applikáció Inaktív állapotba kerül.

- **void new_pop_cb(GtkWidget* w, gpointer data)**

  Ez a callback a File menü Add Pop menüpontjának választásakor és az Add Pop gomb megnyomásakor hívódik meg. Ha az applikáció nincs Futó állapotban, akkor egy GnomeDialogot készít, amiben a létrehozandó populáció paraméterei beállíthatók. Az OK gomb megnyomásakor a populáció elkészül. Az applikáció ezután Inaktív állapotba kerül.

- **void play_cb(GtkWidget* w, gpointer data)**

  A Play gomb mgnyomásakor hívódik meg. Regisztrálja a modell Step() műveletét, hogy a gtk+ minden üres ciklusában hívódjon meg, feltéve, hogy a modell Generált állapotban van. A program Futó állapotba kerül.

- **void stop_cb(GtkWidget* w, gpointer data)**

  A Stop gomb megnyomásakor hívódik meg, és kiveszi a modell Step() műveletét az üresjáratban végrehajtandó műveletek közül, feltéve, hogy a modell éppen Futó állapotban van. A program így Generált állapotba kerül.

- **void close_cb(GtkWidget* w, gpointer data)**

  A File menü close pontjának választásakor hívódik meg. Ha az applikáció nincs Futó állapotban, akkor kitörli a populációkat a modellből, kitörli a kapcsolatfüggvényeket és a sűrűségfüggvényeket is. A program így alaphelyzetbe kerül, az applikáció pedig Inaktív állapotba.

- **void exit_cb(GtkWidget* w, gpointer data)**

  A File menü exit menüpontjának választásakor hívódik meg. Mindazt megteszi, mint a close, de ki is lép a programból.

- **void about_cb(GtkWidget* w, gpointer data)**

A Help menü About pontjának választásakor hívódik meg. Egy kis dialógusablakot hoz létre, ami a programról tartalmaz alapvető információkat. Nem változtatja az applikáció állapotát.

- **void generate_cb(GtkWidget* w, gpointer data)**

  A Generate gomb megnyomásakor hívódik meg. Leellenőrzi, hogy az applikáció Inaktív állapotban van-e, majd meghívja a modell Generate műveletét. Az appliláció ezután Generált állapotba kerül.

- **void ungenerate_cb(GtkWidget* w, gpointer data)**

  A Reset gomb megnyomásakor hívódik meg. Ha az applikáció Generált állapotban van, akkor meghívja a modell UnGenerate műveletét, és az applikáció Inaktív állapotba kerül.

- **void edit_parameters_cb(GtkWidget* w, gpointer data)**

  Az Edit menü Parameters pontjának választásakor hívódik meg. Ha az applikáció nincs Futó állapotban, akkor egy dialógusablakot hoz létre, ahol a modell paraméterei szerkeszthetők. Az OK gomb megnyomása esetén az applikáció Inaktív állapotba kerül.

- **void save_cb(GtkWidget* w, gpointer data)**

  A File menü Save pontjának választásakor hívódik meg. Elmenti a modell paramétereit (lásd store.c), és a populációk adatait az alapértelmezett fájlba.

- **void save_as_cb(GtkWidget* w, gpointer data)**

  Mint fent, de a fájlnevet dialógusablakból választjuk.

A programban a **GNOME/Gtk+** standard callback mechanizmusát használtam. A C++-ba való integrálást olymódon oldottam meg, hogy a **Gtk+** minden callback függvényében megadható egy userdata, ami a könyvtár használója számára tetszőlegesen beállítható. Én itt mindig az **App** osztályra mutató mutatót adtam át, amely **App** osztály tartalmaz minden, a grafikus felülettel kapcsolatos információt és ezen kívül a **PopModell** osztály egy példányát.

7. KÖSZÖNETNYILVÁNÍTÁS

Köszönetet szeretnék mondani Szatmáry Zoltánnak, akivel együtt végeztük mind az eredeti Grőbler-Barna féle modell módosításait, mind a program implementációját.

A jelenleg a programot alkotó osztályok közül az **AlphaFunction, ConnectionSet, Connectiom, Pipe, Population** osztályok, valamint a **Popmodell** osztály egy része eredetileg az ő munkája. Tőle származik továbbá a 6.2. ábra is.

Köszönet illeti továbbá a KFKI Részecske és Magfizikai Kutatóintézet minden munkatársát, akik segítettek, hogy a modellt és annak biológiai hátterét megérthessem.

8. SZÓMAGYARÁZAT

Akciós potenciál: A *membránpotenciálnak* inger hatására bekövetkezõ ugrásszerû megnövekedése, majd lecsengése. Az *akciós potenciál* küszöbjelenség, vagyis nagysága és alakja független az inger nagyságától, "minden vagy semmi" módon jön létre. Az éppen *akciós potenciált* generáló sejtre mondjuk, hogy tüzel.

Axon: Az idegsejt kitüntetett nyúlványa, amely akciós potenciál generálására, gyengítetlen továbbítására, és - speciális végzõdései segítségével - átadására képes.

Burst: Gyors akcióspotenciál-sorozat, amelyet hosszabb "csend" követ. Az egyes akciós potenciálok között a membránpotenciál nem tér vissza a nyugalmi értékre. Az egyedi akciós potenciáloknál hatékonyabb információ-átadást és szinkronizációt tesz lehetõvé.

Dendrit: Az idegsejtnek azon nyúlványainak neve, amelyek akciós potenciál fogadására alkalmas *szinapszisokat* tartalmaznak, és a szinapszison át kapott ingerületet passzív módon a sejt *szómájához* továbbítja. A *dendritek* összessége a dentritfa, amely jellemzően sok dendritből áll és igen szerteágazó.

Egyensúlyi potenciál: Adott ionra jellemzõ feszültségérték, amelynél az elektromos, és a membrán két oldala közötti koncentrációkülönbségbõl adódó diffúziós hajtóerõ éppen kioltja egymást.

Hippocampus: Az agykéreg fejlõdéstanilag legrégebbi részéhez tartozó agyterület. A neurobiológiai kutatások egyik fõ célpontja, mivel alapvetõ szerepet játszik az új emléknyomok rögzítésében, vagyis a memória kialakulásában.

Ioncsatorna: A *sejtmembránba* ágyazott fehérjemolekula, amely bizonyos ionok szelektív áteresztésére képes. A feszültségfüggõ (aktív) *ioncsatornák* a membránpotenciál értékétõl függõen, alakjuk változtatásával képesek nyitni-zárni.

Ionpumpa: A sejtmembránba ágyazott fehérjemolekula, amely koncentrációgradiens ellenében képes ionokat a membránon keresztül mozgatni, és így fenntartja a membrán két oldala közötti koncentrációkülönbséget.

Membrán (sejthártya): A sejtet határoló lipid kettõsréteg, amely elválasztja a sejt anyagát a külsõ tértõl. A különféle molekulák és ionok speciális transzportfolyamatok

segítségével (pl. ioncsatornák és ionpumpák) közlekedhetnek a *membrán* két oldala között.

Membránpotenciál: A membrán két oldala között fennálló potenciálkülönbség, megegyezés szerint a külsõ tér potenciálját nullának tekintve. A *membránpotenciált* a különféle ionoknak a membrán két oldala közötti egyenlõtlen megoszlása hozza létre. Az ionmegoszlás kialakításában az ioncsatornáknak és az ionpumpáknak van

meghatározó szerepük.

Posztszinaptikus: A szinapszis utáni sejt, aminek a *dendritjére* az ingerület érkezik.

Preszinaptikus: A szinapszis előtti sejt, aminek az *axonjáról* az ingerület továbbterjed.

Receptor: Olyan (általában fehérje-)molekula, amely agy adott anyag (ligandum) specifikus megkötésére képes, és ez a kötés mint jel, más folyamatok beindulását eredményezi.

Rekesz(Compartment): Az idegsejtnek a modellezés szempontjából egységesnek tekintett darabkája. A *rekesz*-modellek feltételezik, hogy az egyes rekeszeken belül a különféle *ioncsatornák* sûrûsége állandó.

Szinapszis: Az idegsejtek közötti specializált funkcionális kapcsolat. A pre- és posztszinaptikus sejt membránja nem ér egészen össze, hanem keskeny szinaptikus rést formál, amely elektronmikroszkóppal jól látható. A központi idegrendszerben legelterjedtebb kémiai szinapszison kívül az idegrendszerben elektromos szinapszis és más típusú sejtkapcsolatok is találhatók.

Szóma: A sejttest, amelyből a *axonok* és *dentritek* elágaznak.

Számítógépes idegtudomány (Computational neuroscience): Az a diszciplína, amely az idegrendszer valós szerkezetét és mûködési elveit figyelembe vevõ, kísérleti alapokon nyugvó matematikai modellek készítését, és ezáltal e folyamatok jobb megértését tûzte ki céljául. (vö. *mesterséges neuronhálózat*)

9. HIVATKOZÁSOK

(1) Bower JM, Beeman D (1995): The Book of GENESIS. TELOS, The Electronic Library of Science, Springer-Verlag Publ.

(2) Érdi P, Aradi I, Grõbler T, Barna Gy (1996): Matematikai modellek az idegrendszer kutatásában. Fizikai Szemle 1996/6 (201-207)

(3) Grőbler T., Barna Gy., Érdi, P.: (1998), Statistical model of the hippocampal CA3 region, I. The single cell module: Bursting model of the Pyramidal Cell, *Biol.Cybernetics* (1998), **79**, pp. 301-308

(4) Grőbler T., Barna Gy., Érdi, P.: (1998), Statistical model of the hippocampal CA3 region, II. The population framework: model of rhytmic activity in the CA3 slice. Biol. Cybernetics, **79**, pp. 309-321

(5) Hines M (1993): The NEURON simulation program. In:Neural network simulation environments, Skrzypek (ed.), Kluwer Acad. Publ., Norwell, MA

(6) Hodgkin A, Huxley A (1952): A quantitative description of membrane current and its application to conduction and excitation in nerve. J. Physiology (London) 117, 500-544

(7) Payrits, Sz., Szatmáry, Z., Zalányi, L, Érdi, P. (2000): Use of Parallel Computers in Neurocomputing, LNCS Proceedings, in progress

(8) Traub, R.D., Wong, R.K.S., Miles, R. and Michelson,H. (1991): A model of a CA3 hippocampal pyramidal neuron incorporating voltage-clamp data on intrinsic conductances. *J. Neurophysiol.,* **66**, 534-544

(9) Ventriglia, F.: Kinetic approach to neural systems I. *Bull. Math. Biol.*, **36**, 534-544