FRIEDRICH PÉTER

Idegsejtek biofizikai tulajdonságainak meghatározása valószínűségi keretben, számítógépes modellek paraméter-becslése segítségével

Pázmány Péter Katolikus Egyetem

Információs Technológiai és Bionikai Kar

Info-bionika

Témavezető: Káli Szabolcs

2014.01.01.

Nyilatkozat

Alulírott Friedrich Péter, a Pázmány Péter Katolikus Egyetem Információs Technológiai és Bionikai Karának hallgatója kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, és a szakdolgozatban csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen a forrás megadásával megjelöltem. Ezt a Szakdolgozatot más szakon még nem nyújtottam be.

……

Friedrich Péter

Tartalomjegyzék

Tartalmi összefoglaló

Kísérleti adatok elemzéséhez és biológiai relevanciával rendelkező idegsejt modellek létrehozásához egyaránt szükség van a modellek finomhangolására oly módon, hogy a kapott modell viselkedése valamilyen előre definiált kritérium szerint hasonlítson valós idegsejtek működésére. A megfelelő modell létrehozása, azaz a szabad paraméterek optimális értékének egyszerre történő meghatározása nagyon számításigényes feladat. Ezt a problémát tovább nehezítik olyan technikai részletek, mint a megfelelő költség-függvény vagy az ideális algoritmus megválasztása, vagy ezek helyes, hatékony implementálása. Talán éppen ezen nehézségek miatt nem található olyan szoftver az irodalomban, mely hatékonyan képes megoldani ezeket a problémákat és mindezek mellett áttekinthetően, lépésről lépésre vezeti végig a felhasználót a folyamaton.

Az általunk fejlesztett, Optimizer névre keresztelt program egy olyan moduláris keretrendszer, mely grafikus felületen keresztül segíti a modell optimalizáció problémájának megoldását. A program flexibilitásának köszönhetően a felhasználó könnyedén kibővítheti, testreszabhatja az alkalmazást, így igazodva a konkrét probléma részleteihez. A szoftvert Python programnyelven implementáltuk, részben a nyelv egyszerűsége miatt, részben pedig azért, hogy kihasználjuk azokat a nyílt forráskódú Python csomagokat, melyek már rendelkezésre állnak hasonló problémák megoldására. A Python nyelv használatának köszönhetően a program közvetlenül képes a NEURON szimulátor használatára, egyéb szimulátorok használatát külön interfésszel biztosítottuk.

A szoftvert különböző komplexitású eseteken keresztül teszteltük, több modell osztály és különböző jellegű bemenetek felhasználásával. Az Optimizer felhasználásával sikeresen meghatároztunk passzív paramétereket és konduktancia sűrűségeket többkompartmentumos modellekben, illetve sikeresen illesztettünk absztrakt, adaptive exponential integrate-and-fire típusú modelleket valós biológiai adatokhoz. Ezen tesztek mellett részletesen összevetettük a szoftver teljesítményét már létező modell optimalizációs szoftverekkel, és azt tapasztaltuk, hogy az Optimizer minden esetben azonos vagy jobb eredményeket produkált (1).

Egy ilyen rugalmas keretrendszer azonban nem csak neuron modellek paraméter illesztésére alkalmas, hanem egyéb, modell optimalizációhoz csak lazán kapcsolódó feladat megoldására is. Egy ilyen probléma annak vizsgálata, hogy egy adott kísérleti protokoll mennyi információt képes szolgáltatni biofizikai paraméterekről. A konkrét feladat annak elemzése, hogy különböző típusú elektrofiziológiai elvezetések milyen mértékben alkalmasak a sejtmembrán passzív paraméterei sejten belüli eloszlásának becslésére. Ennek a problémának a szisztematikus kezelésére jól használhatóak a valószínűségi modellezés eszközei.

Munkánk során felépítettünk egy olyan valószínűségi modellt, melynek keretein belül sikerült adott modell osztályokban eltérő paraméter eloszlásokat felismerni és ezek alapján esetszétválasztást végezni, illetve sikerült különböző komplexitású modellek passzív paramétereinek eloszlására becslést adni.

Abstract

The construction of biologically relevant neuronal models as well as model-based analysis of experimental data often requires the simultaneous fitting of multiple model parameters, so that the behavior of the model in a certain paradigm matches (as closely as possible) the corresponding output of a real neuron according to some predefined criterion. Although the task of model optimization is often computationally hard, and the quality of the results depends heavily on technical issues such as the appropriate choice (and implementation) of cost functions and optimization algorithms, no existing program provides access to the best available methods while also guiding the user through the process effectively.

Our software, called Optimizer, implements a modular and extensible framework for the optimization of neuronal models, and also features a graphical interface which makes it easy for even non-expert users to handle many commonly occurring scenarios. Meanwhile, educated users can extend the capabilities of the program and customize it according to their needs with relatively little effort. Optimizer has been developed in Python, takes advantage of open-source Python modules for nonlinear optimization, and interfaces directly with the NEURON simulator to run the models. Other simulators are supported through an external interface. We have tested the program on several different types of problems of varying complexity, using different model classes. As targets, we used simulated traces from the same or a more complex model class, as well as experimental data. We successfully used Optimizer to determine passive parameters and conductance densities in compartmental models, and to fit simple (adaptive exponential integrate-and-fire) neuronal models to complex biological data. Our detailed comparisons show that Optimizer can handle a wider range of problems, and delivers equally good or better performance than any other existing neuronal model fitting tool.

Creating such a flexible framework also allows the user to solve problems not closely related to parameter fitting. One such problem is to determine the amount of information obtained from an electrophysiological experiment using a given protocol. To demonstrate the flexibility of our software and to solve the aforementioned task, we created a probabilistic framework to estimate the distribution of passive membrane parameters in a variety of cell models using different experimental setups.

During our work, we created a probabilistic framework, with which we managed to differentiate the different parameter distributions in a given cell type and, by using this framework, we also obtained an approximate distribution for many passive parameters in neuron models with varying complexities.

Bevezetés

A manapság rendelkezésre álló kísérleti adatok lehetővé teszik egyre komplexebb többkompartmentumos, konduktancia alapú idegsejt modellek létrehozását, melyek képesek nagy pontossággal utánozni valós idegsejtek viselkedését (2,3). Ezen modellek azonban a legtöbb esetben több paraméterrel rendelkeznek, melyek meghatározása gyakorta nagy kihívást jelent.

Egy lehetséges megoldás, melyet főleg hálózati szimulációk esetében szokás alkalmazni, hogy részletes rekonstrukción alapuló modellek helyett kevés paraméterrel rendelkező absztrakt modelleket alkalmazunk (4,5,6). Az ilyen modellek esetében viszont nem támaszkodhatunk közvetlenül a biofizikai ismereteinkre és mérési adatokra a paraméterek helyes értékét illetően. A legtöbb esetben a modell paraméterei és a modell viselkedése között komplex, nemlineáris kapcsolat áll fenn, éppen ezért a megfelelő paraméterek meghatározása nem-triviális feladat, ugyanakkor fontossága miatt igen kiterjedt irodalommal rendelkezik (6-15).

Az irodalom áttekintését követően két kulcsfontosságú lépést sikerült meghatároznunk a modell illesztés folyamatában, az első a modell kimenetének és a cél adat összevetésének módja (azaz a költség-függvény meghatározása), a második pedig a lehetséges megoldások előállításának módja (azaz az optimalizációs algoritmus). A probléma fontosságát jól mutatja, hogy több kifejezetten modellillesztésre tervezett szoftver is létezik, ezek közül érdemes kiemelni a NEURON (16) illetve a GENESIS (17) szimulátorok beépített optimalizációs moduljait, illetve a Neurofitter (15) és Neurotune szoftvereket. Sajnos ezek a szoftverek sok esetben kevés költség-függvényt és/vagy optimalizációs algoritmust tartalmaznak, így új problémák megoldására gyakran nem alkalmasak.

Munkánk során ezért kifejlesztettünk egy Optimizer nevű szoftvert, mely egy intuitív grafikus felület segítségével kalauzolja végig a felhasználót a modell optimalizáció főbb lépésein. A program tartalmazza a leggyakoribb költség-függvényeket és algoritmusokat, ugyanakkor moduláris felépítése lehetővé teszi, hogy a tapasztaltabb felhasználók továbbfejlesszék, kiterjesszék a programot, így igazodva az adott feladat jelentette kihívásokhoz.

Egy ilyen flexibilis keretrendszer azonban jóval több lehetőséget hordoz magában, hiszen a felhasznált építőelemek más területeken felmerülő problémák megoldása során is használhatóak. Ennek demonstrálására kidolgoztunk egy egyszerű valószínűségi modellt, melynek keretein belül a különböző elektrofiziológiai mérések információtartalmát próbáltuk elemzés alá vetni. Maga az elképzelés, azaz idegsejt modelleket használni elektrofiziológiai mérések kiegészítéseként, nem újkeletű ötlet, a megközelítés – azaz valószínűségi keret alkalmazása mérési eredmények információ tartalmának előrejelzésére – azonban merőben új, így szolgáltathat az eddigieknél hatékonyabb, vagy pontosabb megoldásokat.

A következőkben részletesen bemutatjuk a fenti témákban végzett munkát és az elért eredményeket. Elsőként a modell optimalizáció problémáját tekintjük át, bemutatva az Optimizer megtervezésének menetét, illetve a program implementációjának részleteit. Ezt követően megvizsgáljuk az Optimizerrel elért eredményeket és összevetjük a szoftvert más, hasonló funkciójú programokkal.

Második lépésként felvázoljuk a kidolgozott valószínűségi modellt, majd röviden bemutatjuk a modell alkalmazásával kapott eredményeket.

Optimizer

Tervezés

A modell optimalizáció feladata 5 lépésre bontható:

1. az optimalizálni kívánt modell meghatározása
    a. absztrakt szinten (pl. modell osztály)
    b. konkrét implementációs szinten
2. az optimalizálni kívánt szabad paraméterek meghatározása
3. a szimulációs protokoll kiválasztása
4. a cél adat megválasztása (lehet kísérleti, vagy mesterségesen generált)
5. a költség-függvény megadása

Ezt az 5 lépést köti össze a kiválasztott optimalizációs algoritmus, mely megpróbálja megtalálni azt a paraméter-halmazt, mely minimalizálja a költség-függvényt.

Az egyszerűbb modellek esetében a legtöbb hagyományos algoritmus képes megtalálni a helyes megoldást megfelelő pontossággal, elfogadható idő alatt. Komplexebb problémák esetében azonban még az összetettebb algoritmusok sem garantálják a globális optimum megtalálását.

Mivel az összes fenti elem lényegében tetszőleges kombinációja felmerülhet, mint megoldandó probléma, igen nehéz jól használható kezelőfelületet tervezni egy ilyen programhoz. Szerencsére a gyakorlatban nem kell minden lehetséges problémára felkészülni, mivel például a legtöbb elektrofiziológiai mérés során current clamp vagy voltage clamp konfigurációt használnak valamilyen step protokollal kombinálva. Ez lehetővé teszi, hogy olyan interfészt készítsünk, melyen keresztül a leggyakoribb esetek kényelmesen kezelhetők. Az idegsejt modellek, melyek a legnagyobb variabilitást mutatják, könnyedén kezelhetők, köszönhetően a különböző modell leíró nyelveknek. Ezek a leíró nyelvek olyan kódot jelentenek, melyet az optimalizációs szoftverek közvetlenül manipulálhatnak, és a szimulátorok közvetlenül képesek futtatni.

Mindezeket figyelembe véve kettős célt tűztünk ki; elsőként szerettünk volna a lehető legrugalmasabb programot létrehozni, mely képes a problémák legszélesebb körű megoldására, azaz képes a lehető legtöbb modelltípus, szimulációs környezet, stimulációs protokoll, költség-függvény és optimalizációs algoritmus kezelésére. Második célként egy egyszerű és intuitív interfész létrehozását tűztük ki, mely segítségével akár egy tapasztalatlan felhasználó is képes lenne megoldani a gyakoribb optimalizációs feladatokat.

Mint látható a két cél némiképp ellentmond egymásnak, éppen ezért a programot két különálló szintre bontottuk. Az alsó szinten azok a komponensek találhatók, melyek a fent bemutatott 5 feladatot végrehajtják, illetve azon részek, melyek a komponensek interakcióját megvalósítják. Látható, hogy felépítéséből adódóan ez egy nagyon moduláris megvalósítást jelent, melyben bármelyik modul lecserélhető, vagy kibővíthető a többi módosítása nélkül.

A felső szinten található maga a grafikus felület, illetve a parancssoros interfész. A felhasználó a grafikus felületen keresztül könnyedén beállíthatja az alsó szinteken elhelyezkedő komponensek paramétereit, ezen felül pedig áttekintheti, analizálhatja a kapott eredményeket. A grafikus felület elkészítésekor a wxPython programcsomagra támaszkodtunk, mely a közismert C++ keretrendszer, a wxWidget Python nyelvű implementációja.

Implementáció

A Python programnyelv választása kézenfekvőnek tűnt, mivel a nyelv igen egyszerű és flexibilis, ugyanakkor nagyobb léptékű problémák megoldására is jól használható.

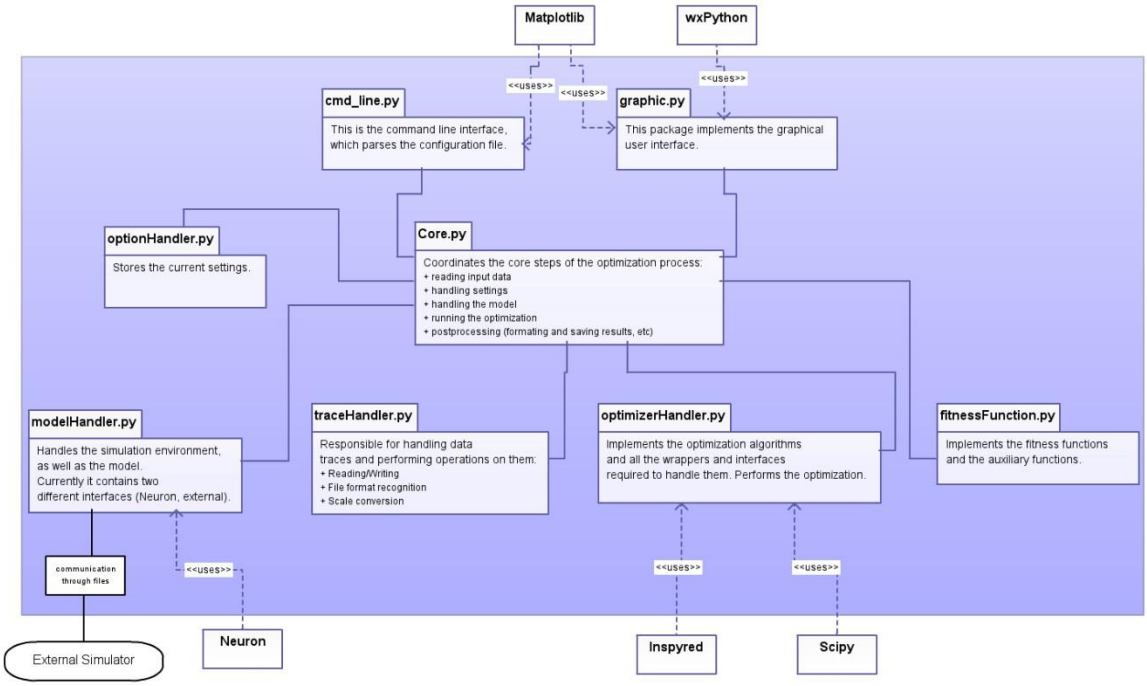
Ezen felül nagyon sok olyan nyílt forráskódú Python programcsomag létezik, melyek kész megoldást kínálnak a modell optimalizáció egyes lépéseinek végrehajtására, mint például az adatok kezelése és vizualizációja, vagy a nem-lineáris optimalizáció. A Python nyelv mellett szólt az a tény is, hogy a legtöbb modern szimulátor rendelkezik Python interfésszel.

A Python nyelv használatának köszönhetően a program közvetlenül interakcióba tud lépni a NEURON szimulátorral, ami lehetővé teszi, hogy külön interfész nélkül kezelhessük a modelleket, illetve futtassuk a szimulációkat. Természetesen a program támogat más szimulációs szoftvereket is, ezekkel az Optimizer indirekt kapcsolatban áll, és black-box módon kezeli őket. Ez a black-box módszer jelen esetben azt jelenti, hogy az Optimizer input-output fájlok formájában kommunikál a szimulátorral, ez pedig lehetővé teszi, hogy lényegében tetszőleges futtatható alkalmazás használható szimulátorként, amennyiben képes a megfelelő fájl-műveletek végrehajtására.

A konkrét optimalizáció elvégzésére a program mind lokális, mind globális algoritmusokat is kínál, melyek az inspyred és scipy programcsomagokból származnak.

A költség-függvények legtöbbjét az Optimizer implementációja tartalmazza, ez alól

kivétel a Phase Plane Trajectory Density (PPTD) módszer, mely a pyelectro csomagból származik.



1. ábra Az Optimizer belső felépítése

A program létrehozása során egy moduláris struktúra kialakítását tartottuk szem előtt, a kész programba a következő modulok kerültek be (1. ábra):

1) traceHandler modul:

   Ez a modul tartalmazza a Data nevű osztályt, mely a program központi adathordozó struktúrája, ez az osztály felel az adattárolás mellett az input fájlok kezeléséért is. Ez a struktúra tartalmazza továbbá a Trace osztályt, melynek feladata az egyes adatsorok kezelése. A modul ezen felül tartalmazza még az egyéb, adatkezeléshez kapcsolódó alfeladatokat ellátó függvényeket is, mint például a mértékegység átváltás. A program jelenlegi verziója csak egyetlen adatsor szettet képes kezelni, melyben az adatsorok azonos típusú adatokat kell hogy tartalmazzanak (hossz, mintavételezési frekvencia, mérési protokoll, stb.). Természetesen tervezzük a modul kibővítését, mivel szükség lehet több különböző jellegű adatok (akár absztrakt, akár kevert mérési eredmények) felhasználására. Egy ilyen lehetséges eset, amit a program részben már képes támogatni, amikor a feszültséggörbe mellett explicit akcióspotenciál időket is fel kívánunk használni.

2) modelHandler modul:

   Két fő osztályt tartalmaz, melyek az idegsejt modellek kezeléséért felelnek. Ezek

közül az első, a modelHandlerNeuron osztály, mely a beépített NEURON szimulációs környezet kezelését valósítja meg, ezen az osztályon keresztül érhetőek el a NEURON mérési, és szimulációs protokolljai, illetve a NEURONban implementált idegsejt modellek is. A második osztály az externalHandler, mely minden egyéb, a felhasználó által definiált, külső szimulátor kezelését megvalósítja.

3) optionHandler modul:

   Ez a modul egy egyszerű adathordozó struktúrát tartalmaz, melynek feladata a modell optimalizációs probléma összes paraméterének és beállításának rögzítése. Ez a modul felel a konfigurációs fájl kezeléséért (írás és olvasás).

4) optimizerHandler modul:

   Ez a modul tartalmazza a programban implementált optimalizációs algoritmusokat, mindet külön osztályban, illetve itt találhatóak azok a kiegészítő függvények (paraméter normalizáció, paraméter határok kezelése), melyek minden optimalizációs algoritmus számára nélkülözhetetlenek. A felhasználó a modul kibővítésével további algoritmusokat adhat a programhoz, ehhez mindössze az algoritmust megvalósító osztályt kell létrehozni, illetve regisztrálnia kell az új algoritmust a Core modulban és a grafikus felületen.

Jelen pillanatban 7 különböző algoritmus implementációját tartalmazza a program, ezek közt találhatunk egyaránt lokális és globális algoritmusokat illetve egy főként tesztelésre szolgáló véletlen bolyongáson alapuló módszert is (lásd részletesen a következő szekcióban).

5) fitnessFunctions modul:

   Ebben a modulban található a különböző költség-függvények (fitness függvények) implementációja. Itt található még egy osztály, melynek feladata az akciós potenciálok kezelése. A felhasználó itt is könnyedén kibővítheti a függvények listáját egy tagfüggvény létrehozásával és regisztrálásával.

Jelenleg a következő függvények érhetők el a programban (részletesen lásd később):

- Mean squared error
- Mean squared error excluding spikes
- Spike count
- Spike count during stimulus
- ISI differences
- Latency to first spike
- AP overshoot
- AP width
- AHP depth
- Derivative difference
- Phase plane trajectory density (PPTD)

A program ezen függvények tetszőlegesen súlyozott kombinációját képes költség-függvényként használni, erre a célra szolgál az osztály fő metódusa a combineFeatures függvény, mely létrehozza a szimulált adatsort, valamint meghatározza az aktuális szimuláció és a cél adatsor költség értékét a megadott költség-függvények és súlytényezők felhasználásával.

6) Core modul:

Mint a neve is mutatja, ez a szoftver központi modulja, mely elvégzi az optimalizációs probléma megoldásának egyes lépéseit a megfelelő modulok segítségével:

1. bemeneti adatok feldolgozása
2. modell fájl betöltése és az optimalizálandó paraméterek meghatározása
3. szimulációs és mérési protokoll meghatározása
4. költség-függvény megadása
5. algoritmus megválasztása és beállítása
6. az optimalizáció elvégzése
7. a beállítások elmentése a konfigurációs fájlba illetve az eredmények eltárolása HTML formátumban

A program használata

A felhasználó szakértelmétől, illetve a megoldandó probléma komplexitásától függően három különböző felhasználói szintet különböztethetünk meg a programban. A legalapvetőbb szinten a felhasználó elvégezheti a modell optimalizációt a grafikus felület segítségével, vagy a

parancssoros interfész használatával, a szoftver beépített eszközeire támaszkodva. A középső szinten a felhasználó bizonyos területeken kiterjesztheti a program lehetőségeit a grafikus felület és külső fájlok segítségével (lásd lejjebb). A legfelső, egyben legkomplexebb szinten a felhasználó új keretrendszert hozhat létre, építőkövekként használva az Optimizer már implementált moduljait, vagy kibővítheti azokat az adott problémához igazítva ezzel a program funkcióit.

Mint korábban említettük, bizonyos szimulátorok (NEURON) modell implementációit a program közvetlenül képes kezelni és értelmezni, ezeket a szimulátorokat nevezzük belső szimulátornak. Természetesen lehetetlen lenne minden lehetséges szimulátorhoz külön interfészt biztosítani, ezért építettünk a programba egy olyan interfészt, amely minden, közvetlenül nem támogatott, külső szimulátort kezelni képes. A külső szimulátorokat a program fekete doboz szerűen kezeli, azaz csak bemenet-kimenet párokat használ. Ezzel a módszerrel lehetővé tettük, hogy a felhasználó minden olyan futtatható állományt szimulátorként használjon, mely képes az Optimizer által küldött adatok beolvasására és ezekre megfelelő formátumú kimenettel képes reagálni. Az így megvalósított külső szimulátorok egyik hátránya, hogy a modell és szimuláció kezelést a felhasználónak az Optimizertől függetlenül kell megvalósítania, viszont a szoftverünk kínálta algoritmusokra és költség-függvényekre továbbra is támaszkodhat. Ezzel szemben a belső szimulátorok használatakor a felhasználó magas szinten képes interakcióba lépni a szimulációs környezettel a grafikus felületen keresztül.

A grafikus felület használata nagyban megkönnyíti a különböző modell optimalizációs problémák megoldását, ugyanakkor bizonyos feladatok elvégzéséhez nem elég rugalmas. Egy ilyen eset, amikor a megoldandó probléma megköveteli, hogy a modell több paraméterét valamilyen absztrakt paraméter értéke alapján határozzunk meg. Egy konkrét példa a fenti esetre, amikor egy sok kompartmentumos, realisztikus modell esetében próbáljuk meghatározni a passzív biofizikai paramétereket, egy áram injekcióra adott válasz alapján. Ebben az esetben nem szerencsés a membrán ellenállást minden dendrit szekcióra különálló paraméternek tekinteni (mivel ez nagyon sok paramétert jelentene), ehelyett praktikusabb egy paramétert alkalmazni, amelyből aztán minden szekcióban a lokális geometria alapján kiszámolható a tényleges membrán ellenállás. Hogy az ilyen jellegű paraméter leképzéseket lehetővé tegyünk absztrakt és tényleges modell paraméterek között, beépítettünk egy felhasználói függvény opciót a szoftverbe. Ebben a felhasználói függvényben lehetőség nyílik arra, hogy absztrakt paramétereket definiáljunk, és meghatározzuk ezek viszonyát a modell valós paramétereivel. Ez

a megoldás lehetővé teszi, hogy olyan paramétereket is optimalizáljunk, melyek szigorúan véve nem részei a modellnek (ilyenek például a bejövő szinaptikus kapcsolatok paraméterei).

Egy másik helyzet, amikor a grafikus felület önmagában nem elegendő a probléma megoldására az olyan problémák esete, amikor nem egyszerű, úgynevezett step stimulust szeretnénk használni. Az ilyen áram vagy feszültség step-ek használata igen gyakori, és néhány paraméterrel jól definiálható, ezért a grafikus felületről vagy a konfigurációs fájl használatával könnyen kezelhető. A szimulációk és különösen kísérleti mérések során azonban gyakran alkalmazunk összetettebb stimulációs protokollt (lásd később, 4. teszt eset), melyek kezeléséhez intuitív felületet tervezni igen nagy kihívás, ezért egy egyszerűbb megoldás mellett döntöttünk: a program képes egy megadott fájlból olyan idősor beolvasására, mellyel stimulálni képes az adott modellt.

Hibafüggvények

A modell optimalizáció talán legkritikusabb lépése a költség- vagy fitness függvény megválasztása, mivel ez fogja meghatározni, hogy az algoritmusunk milyen szempontból fogja az optimális megoldást megtalálni, azaz a modellünk milyen módon fog hasonulni a cél neuronhoz. Éppen ezért fektettünk nagy hangsúlyt minél több, az irodalomban bemutatott költség-függvény implementálására. Az Optimizer jelenleg a következő függvényeket képes használni:

Mean squared error:

A két adatsor között vett pontonkénti átlagos, négyzetes hiba, mely a kísérleti adat értéktartományával van normálva. A mennyiben a felhasználó meghatároz egy kovariancia mátrixot, akkor a függvény a kovarianciával súlyozott különbséget (lényegében a Mahalanobis távolság négyzetét) adja vissza.

Mean squared error excluding spikes:

A fent leírt mennyiséget adja meg, de csak az adatsorok küszöb alatti részét figyelembe véve, azaz, az akciós potenciálokat egy adott ablakolással kihagyja az adatsorokból.

Derivative difference:

A két adatsor deriváltjainak átlagos, négyzetes különbségét adja meg, normálva.

Spike count:

Az akcióspotenciálok számának abszolút különbségét adja meg, normalizálva a két adatsor

akciópotenciáljainak összegével

Spike count during stimulus:

Ugyanaz, mint fent, de csak a stimulus időtartama alatt bekövetkező akciópotenciálokat figyelembe véve.

ISI differences:

A két adatsor összetartozó akciópotenciáljai közt eltelt időintervallumok abszolút különbségének összege, mely normálva van az adatsorok hosszával.

Latency to 1st spike:

Az első akciópotenciálig eltelt négyzetes időkülönbség, mely normalizálva van az adatsorok hosszának négyzetével.

AP overshoot:

Az akciópotenciálok átlagos, négyzetes amplitúdó különbsége, normalizálva a kísérleti adatsor amplitúdó maximumának négyzetével. Az akciópotenciál amplitúdóját az AP csúcs értéke és az AP küszöbértéke közti feszültség különbségként definiáljuk.

AP width:

Az akciópotenciálok szélességének átlagos, négyzetes különbsége, normalizálva a kísérleti adatsor átlagos AP szélességének négyzetével

AHP depth:

Az utó-hiperpolarizáció mélységének átlagos, négyzetes eltérése a két adatsor közt, mely normalizálva van a kísérleti adatsor küszöb alatti feszültségtartományával.

Phase plane trajectory density (PPTD):

A két adatsort a fázistérben hasonlítja össze (14), a pyelectro modul pptd_error függvényének felhasználásával.

A legtöbb költség-függvény rendelkezik egy vagy több szabadon állítható paraméterrel. Ilyen paraméter például az akciópotenciálok detektálásához használt küszöbérték, vagy a Mean squared error excluding spikes függvény által használt akciópotenciál ablak szélessége.

A szoftver képes a fent felsorolt függvények tetszőleges lineáris kombinációját felhasználni az optimalizáció során. Minden függvény a 0-1 tartományra van normalizálva, így garantálva, hogy az adott függvény a megadott súlynak megfelelő arányban járuljon hozzá az összesített költség értékéhez. Több bemenő adatsor esetén a végső költség a bemenő és szimulált adatsorokra számolt költség-érékek összegeként adódik.

Mivel fent felsorolt függvények implementációja minden esetben pontonkénti összehasonlításon alapul, valamilyen módon garantálnunk kellett, hogy a megfelelő pont párokat hasonlítjuk össze. Ez a módszer akkor válik problémássá, ha a felhasználó eltérő frekvenciával mintavételezett adatsorokat próbál összehasonlítani, hiszen ilyenkor ugyanolyan hosszúságú adatsorokat kapnánk más időskálával (vagy éppen fordítva: eltérő hosszúságú adatsorok, melyek ugyanahhoz az időskálához tartoznak). A problémát igyekeztünk a lehető legegyszerűbben megoldani, ezért két szabályt fektettünk le:

1. Ha a bejövő adatsor mintavételezési frekvenciája magasabb, mint a modell mintavételi frekvenciája, akkor a szimulációban használt mérési frekvenciát a bejövő adatsorhoz igazítjuk, így garantálva a pontonkénti összehasonlítás elvégezhetőségét. Ez egyben azt is jelenti, hogy az input adatsor felbontásánál gyengébb felbontás használatát a program nem engedélyezi.

2. Ha a bejövő adatsorunk felbontása kisebb, mint amire szükségünk van a szimulált esetben (ami gyakran előfordulhat, ha valós biológiai adatra próbálunk integrate-and-fire típusú modellt illeszteni), akkor a program lineáris interpolációt alkalmazva előállít egy új input adatsort, mely felülírja az eredetit.

Algoritmusok

Az irodalomban nagyon sok olyan algoritmust találhatunk, melyeket nemlineáris optimalizációs problémák megoldására fejlesztettek ki. Az Optimizer fejlesztése során elsősorban olyan algoritmusok implementációjára koncentráltunk, melyeket már korábban is sikeresen alkalmaztak idegsejt modellek paraméter illesztésére. Két algoritmus csoportot is alkalmaztak szép eredményekkel különböző kutatások során: az egyik az evolúciós (genetikus) algoritmusok családja, a másik a szimulált hőkezelés alapú algoritmusok csoportja, mindkét csoport bizonyítottan alkalmas lehet több tucat paraméter illesztésére (7).

Mivel a konkrét algoritmusok teljesítménye és hatékonysága nagyban függ az implementáció részleteitől, ezért például a szimulált hőkezelés algoritmus két eltérő változatát is a program részévé tettünk (egyet az inspyred egyet pedig a scipy programcsomagból). Ezek mellett implementáltunk egy testreszabott genetikus algoritmust, illetve egy differenciál evolúciós

algoritmust is. A fenti globális algoritmusokon felül megtalálható néhány lokális algoritmus a programban: egy klasszikus szimplex módszeren alapuló algoritmus, illetve az L-BFGS-B (18) algoritmus, mely az egyik legmodernebb lokális módszer.

Az általunk vizsgált esetek mindegyikét képesek voltunk a fenti algoritmusok valamelyikével megoldani, de természetesen törekedtünk arra, hogy a felhasználók egyszerűen bővíthessék az algoritmusok listáját.

Mivel a legtöbb optimalizációs algoritmusnak problémát jelenthet, ha a paraméterek eltérő nagyságrendbe esnek (és az idegsejt modellek paramétereire ez szinte mindig igaz), a szoftver az optimalizáció során normalizált, 0-1 intervallumba eső paramétereket használ. Az algoritmusok által generált értékeket ezért csak egy re-normalizáció után adjuk át a szimulátornak. Ahhoz hogy a normalizáció és re-normalizáció elvégezhető legyen, mindenképpen szükség van az egyes paraméterek értéktartományának megadására (mivel a legtöbb modell illesztési probléma esetében ezek a tartományok adottak), illetve arra, hogy az algoritmusok támogassák az ilyen jelegű optimalizációt.

Az algoritmusok alapesetben egy véletlenszerű pontból, vagy egy véletlenszerű pont-populációból indulnak ki (a megadott paraméter tartományon belül), de természetesen a felhasználónak lehetősége van kezdeti pont(ok) megadására is.

Globális algoritmusok

Evolúciós algoritmus: Egy olyan, testreszabott genetikus algoritmus, mely generációs cserélési technikát (generational replacement) alkalmaz, gyenge elitizmussal (azaz a legjobb megoldás mindig megmarad), illetve gaussi mutációt és “blend” cross-over-t a költség minimalizációjára (a részleteket lásd az inspyred dokumentációjában).

Szimulált hőkezelés: Az inspyred programcsomag evolúciós keretrendszerét használja fel egy szimulált hőkezelés alapú cserélési technikát alkalmazva (a részleteket lásd az inspyred dokumentációjában).

Basinhopping:

Egy olyan sztochasztikus algoritmus, mely nagyban hasonlít a scipy programcsomag korábbi kanonikus szimulált hőkezelés algoritmusához (19). A mi implementációnk a lokális minimalizáció fázisában az L-BFGS-B algoritmust használja (a részleteket lásd a scipy dokumentációjában).

Differenciál evolúció:

A Storn és Price által leírt differenciál evolúció algoritmusát valósítja meg az inspyred programcsomag DEA osztályán keresztül. Az algoritmust kis részben módosítottuk, hogy a populáció mérete állandó maradjon az optimalizáció során.

A globális algoritmusok közé sorolható még a véletlen bolyongás (Random search) algoritmus is, ezt azonban főként tesztelésre használtuk, azzal a céllal, hogy bemutassuk, az általunk felvázolt problémák komplexitásukból eredően nem oldhatók meg egyszerűbb algoritmusokkal.

Lokális algoritmusok

Szimplex módszer: A Nelder-Mead szimplex algoritmust használja lokális minimum keresésre (20).

L-BFGS-B: Egy limitált memóriájú, korlátos Broyden-Fletcher-Goldfarb-Shanno algoritmust használ a minimalizációhoz (18).

A grafikus interfész

Ahogy a korábbiakban már érintőlegesen említettük, az Optimizer összes alapvető, és néhány magas szintű funkciója is kényelmesen elérhető egy grafikus felületen keresztül. Ez a GUI hét, úgynevezett lapból áll, ezek a lapok felelnek az optimalizációs folyamat egy-egy lépésének végrehajtásáért.

Az első lapon a felhasználó kiválaszthatja a bemenő adatsor(oka)t tartalmazó fájlt, és minden, az adatsorhoz kapcsolódó adatot (mérés típusa, hossza, alkalmazott mértékegység, stb.). Ezen felül meg kell határoznia egy mappát a számítógépen, ahová a program a futás során keletkezett fájlokat és eredményeket összegyűjtheti. Az összes adat megadása után a felhasználó betöltheti az adatsor(oka)t, és a beolvasás eredményét egy ploton ellenőrizheti. A program az egyszerűbb (adatsor, idő-és-adatsor) szöveges formátumok mellett a PyNN szimulátor által generált fájlokat képes kezelni. Több adatsor használata esetén ezeket az ábrán összefűzve jeleníti meg a program, de valójában külön-külön kezeli őket.

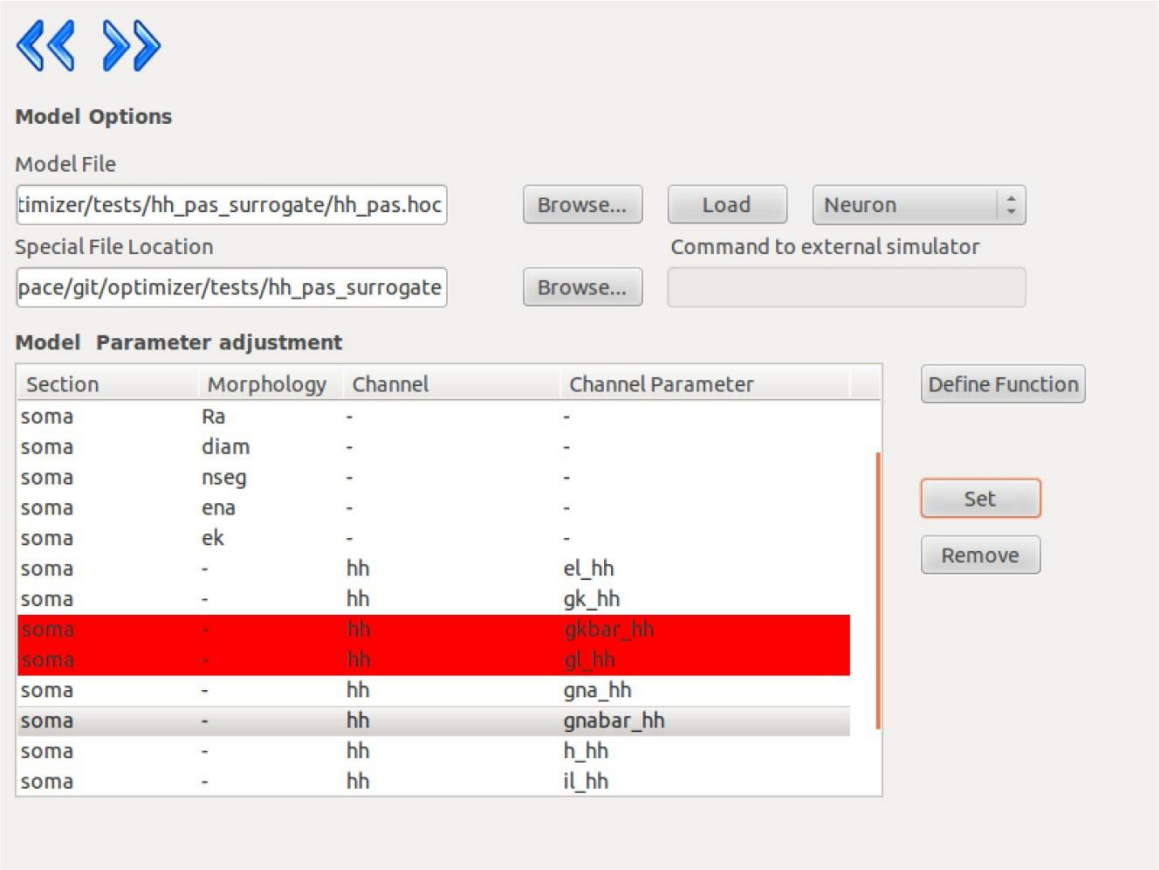
A második lapon a felhasználó kiválaszthatja a használni kívánt szimulátort. Jelenleg csak a NEURON és az external lehetőségek közül lehet választani, mivel a NEURON az egyetlen belső szimulátor, minden más programot külső szimulátorként tud csak kezelni az Optimizer. A külső szimulátor használatához egy parancssori utasítást kell megadni, ami a következőkből épül fel:

- a szimulátort futtató parancs
- a modell fájl neve
- szimulátor paraméterek (opcionális)
- utolsó paraméterként az optimalizálandó paraméterek száma

A külső szimulátor használatához nem elegendő csak a modellt definiálni a modell fájlban, annak tartalmaznia kell a stimulációs és mérési protokollokat, illetve képesnek kell lennie fájlból modell paramétereket fogadnia, valamint a futtatási eredményt egy szöveges fájlba elmenteni (minden adatsort külön oszlopban tabulálva).

Ha a felhasználó a NEURON-t kívánja használni, akkor meg kell adnia a modellt tartalmazó .hoc kiterjesztésű fájlt, mely (csak) a modell definíciót tartalmazza. Ha a modell tartalmaz olyan speciális csatornákat, melyek külső fájlokban találhatók, akkor ezeket a .mod fájlokat tartalmazó mappát is meg kell adni. A program jelenlegi verziója sajnos nem képes a modell helyes betöltésére, ha a hozzátartozó mechanizmusokat tartalmazó fájlokat nem sikerült megtalálnia, ezért érdemes többször ellenőrizni, hogy helyes-e a megadott elérési útvonal, mivel a programot ilyenkor újra kell indítani.

Ha a modellt sikerült betölteni, akkor a tartalma automatikusan megjelenik az oldalon található táblázatban, mely segítségével a felhasználó kiválaszthatja az optimalizálandó paramétereket (2. ábra).



2. ábra Az Optimizer modell kezelő felülete

A korábban említett felhasználói függvény használatára is itt nyílik lehetőség. A függvény létrehozását egy előre megadott minta segíti, de lehetőség van egy már megírt függvény betöltésére is. A függvénynek be kell tartania a Python nyelv szintaktikai megkötéseit és ezen felül helyesen kell használni a NEURON Pythonba épülő programcsomagját is. Ha a függvényt helyesen definiáltuk, akkor a választható paramétereket a program elrejti, mivel a modell paraméterek direkt felhasználása nem megengedett a továbbiakban. A felhasználó által megadott függvényt a program szintaktikai szempontból ellenőrzi, de érdemes többször is ellenőriznünk mindent, mert az esetleges hibák csak futás közben fedezhetők fel.

A következő lapon a stimulációs protokollt, illetve a szimulációt és a szimulált mérést érintő beállításokat adhatjuk meg. A felhasználó szabadon választhat current clamp és voltage clamp konfigurációk közül, ezekhez tetszőleges típusú stimulust választhat (step protokoll vagy szabad hullámforma megadásával). A stimulus típus megválasztását követően annak paramétereit kell megadni, azaz vagy a szabad hullámformát tartalmazó fájl nevét és elérési útját, vagy a step protokoll késleltetését, hosszát és amplitúdóját. Több adatsor felhasználásakor lehetőség van különböző stimulus amplitúdók megadására, ezek száma a grafikus felületen tízben maximalizálva

van (parancssoros futtatáskor nincsen limitáció). Ezt követően a felhasználónak meg kell adnia a stimulált szekció nevét és a szekción belüli pozíciót.

Ez után a második oszlopban ki kell tölteni a szimuláció paramétereit. A felhasználónak meg kell adnia egy kezdeti feszültség értéket, a szimuláció hosszát, és az integrálás lépéseinek méretét (a program jelenlegi verziója csak fix lépésközzel képes az integrálás elvégzésére). Ezeken felül meg kell még határozni a mérés helyét (szekció és pozíció), illetve a mérni kívánt paramétert (áram vagy feszültség).

A következő lapon a költség-függvények kiválasztása és beállítása végezhető el. Itt tudunk súlyokat rendelni a kiválasztott függvényekhez, illetve lehetőség van a súlyok normalizálására is, valamint finomíthatjuk az adott függvények viselkedését.

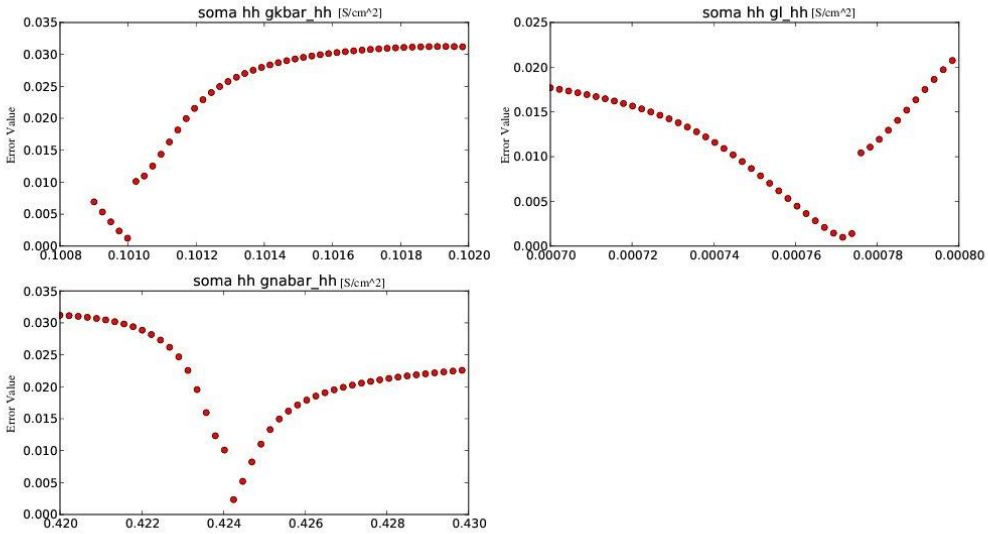
A következő lapon az optimalizációs algoritmust választhatjuk ki és állíthatjuk be. Itt kell megadnunk továbbá az optimalizálandó paraméterek értéktartományát, és itt határozhatunk meg kezdeti ponto(ka)t is az algoritmusok számára. Ha lokális algoritmust választunk és megadtunk egy kezdeti pontot, akkor az algoritmus ebből a kiválasztott pontból fog indulni, ellenben a globális algoritmusok esetén csak az garantált, hogy az általunk választott pont szerepelni fog a kezdeti populációban (éppen ezért van lehetőség teljes populáció megadására is).

Ezen paraméterek megadásával az optimalizációs problémát sikeresen specifikáltuk, a program a Run gomb megnyomásával elkezdi megoldani a problémát.

Miután a program befejezte az optimalizációs folyamatot, a kapott eredmények megtekinthetőek, mind numerikus formában, mint pedig grafikusan összevetve a cél adatsorral. Ez a grafikus eredmény elmentésre kerül kép formájában (több formátumban is), illetve az optimális paraméterekkel generált adatsor kiíródik egy szöveges fájlba. A kapott eredményekről, illetve a futtatás lényegesebb beállításairól egy html formátumú összefoglalót generál a program. A szoftver ezen felül készít egy konfigurációs .xml kiterjesztésű fájlt, melyben az összes beállítás kiírásra kerül, így téve reprodukálhatóvá a futtatást, mivel ezt a konfigurációs fájlt a parancssoros interfész segítségével bármikor újra lefuttathatjuk.

Az utolsó lapon a felhasználónak lehetősége van korlátozott mértékben elemeznie a kapott eredményeket. Néhány alapvető statisztika megtekintésén túl lehetőség van költség-függvény részletes elemzésére is, illetve az inspyred algoritmusok használatakor a generation plot (a költség-függvény változását mutatja generációról-generációra) és allel plot (főként egydimenziós problémák esetében használható) vizsgálatára. Ezen felül található még egy analízist szolgáló eszköz az Optimizerben, ez pedig a grid plot nevű ábra, mely a költség-függvényt ábrázolja egy

bizonyos paraméterhalmazon kiértékelve és a paraméterek mentén vetítve. Ezzel a módszerrel a felhasználó megvizsgálhatja az optimum körül a keresési teret, mely sok, lényeges információval szolgálhat (3. ábra).



3. ábra A Grid plot segítségével megvizsgálhatjuk a paraméterteret a program által talált optimum közelében

A paraméterhalmazt a program úgy állítja elő, hogy minden paramétert rögzít az optimumban, kivéve egyet, mely szabadon változhat. Ezt a módszert iterálva az összes paraméterre egydimenziós szeleteket kapunk az úgynevezett fitness landscape-ről az optimum körül.

Parancssoros interfész

A parancssoros interfész használatához a felhasználónak rendelkeznie kell egy konfigurációs XML fájllal, mely xml tag-ek formájában tartalmazza az összes releváns beállítást az adott futtatáshoz. Minden lehetséges beállítás rendelkezik tag-gel, a tag-ben tárolt értéknek a program által értelmezhető formátumban kell lenni.

Ezt az interfészt azért építettük bele az Optimizerbe, hogy támogatni tudjunk olyan rendszereket is, ahol a grafikus felületre nincs szükség, vagy nem elérhető, azaz az optimalizációnak felhasználói interakció nélkül kell lefutnia. Mivel ez csak kiegészítő funkció, az implementáció nem "bolondbiztos", nincsen semmilyen hibadetekció, vagy hibajavító eljárás, azaz egy hibásan megadott paraméter csak futás közben kerül detektálásra. Éppen ezért javasoljuk, hogy a konfigurációs fájlt a GUI-n keresztül generálják a felhasználók, és ezt követően az így kapott mintát módosítsák az aktuális problémának megfelelően.

Eredmények

Teszt esetek

Igyekeztünk az Optimizert úgy megtervezni, hogy a problémák lehető legszélesebb skáláját tudja kezelni, és éppen ezért több különböző jellegű, komplexitású eseten teszteltük. A modell optimalizáció problémája igen sok formában megjelenhet, hiszen a használt modell típusa, annak paraméterei, a szimulációs protokoll, a céladat, vagy a költség-függvény szabadon változhat, és éppen ezért több különböző optimalizációs módszerrel is megoldható.

A probléma egy érdekes aspektusa maga a cél adatsor jellege. A modell optimalizációs módszerek tesztelésének egyik leggyakoribb módja a mesterséges adatok használata. Ebben az esetben a cél adatot egy meglévő modellel generáljuk, majd a kiválasztott paramétereket elállítjuk, és azt vizsgáljuk, hogy az alkalmazott módszer képes-e visszaállítani őket. Ez a módszer főként tesztelésre és debuggolásra alkalmas, illetve az adott feladat komplexitásának megbecsülésére, valamint a kiválasztott algoritmus teljesítményének meghatározására. Láthatjuk azonban, hogy ez a fajta tesztelés nem feltétlenül ad hiteles képet a tesztelt módszer teljesítményéről, hiszen ebben az esetben tudjuk, hogy létezik tökéletes (0 hibájú) megoldás, míg ez valós adatok esetében nem feltétlenül igaz, ezért nem jelenthető ki, hogy egy algoritmus, mely az első esetben jól teljesít, az ugyanilyen jó eredményeket ad valódi adatokra is (10). Éppen ezért létrehoztunk olyan teszt eseteket is, melyekben tökéletes megoldás valószínűleg nem létezik: a cél adatsort vagy egy magasabb komplexitású modellből nyertük ki (az így elvégzett illesztés a szisztematikus modell egyszerűsítés egyik legfontosabb lépése), vagy valódi, biofizikai mérésekből származó adatokat

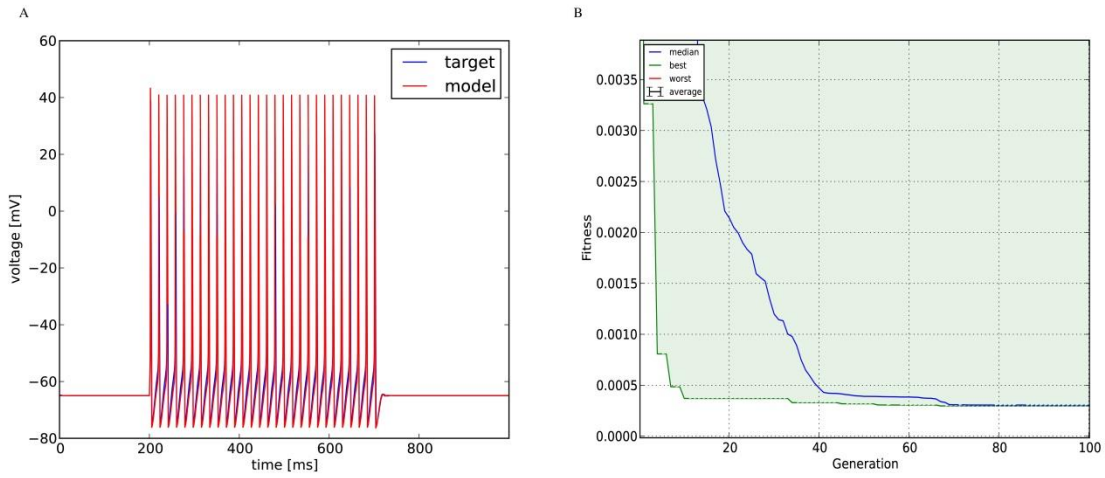
használtunk fel.

A következőkben bemutatjuk, hogyan teljesített az Optimizer a kiválasztott 5 teszt esetben. A teszt eseteket igyekeztünk úgy megválasztani, hogy a fenti kritériumoknak megfeleljen, illetve jól bemutassa az Optimizer teljesítményét a különböző problémák megoldásában. Minden teszt esetet hagyományos asztali PC-n vagy laptopon futtatunk különböző Linux alapú operációsrendszereken. Az optimalizációk időigénye igen széles skálán mozgott, a néhány perctől a néhány napig, a probléma komplexitásától függően.

1. Egykompartmentumos Hodgkin-Huxley modell konduktanciáinak visszaállítása egyszerű step stimulusra adott feszültségválasz alapján.

NEURON modellező környezet segítségével létrehoztunk egy egykompartmentumos modellt, mely a klasszikus Hodgkin-Huxley modell nátrium, kálium és szivárgási konduktanciáit tartalmazta. A modell egyetlen szekciójának átmérőjét 10 μm-re állítottuk, miközben a hosszát és a modell passzív paramétereit változatlanul hagytuk. Ezt követően áramot injektáltunk a kompartmentumba amplitúdó=200 pA, késleltetés=200 ms, időtartam=500 ms), hogy létrehozzuk a mesterséges céladatunkat. Ezt követően megváltoztattuk a konduktancia értékeket, és az így kapott új modellt próbáltuk az előbb létrehozott adatsorhoz illeszteni. Az eredeti paraméterek megtalálásához az Optimizer genetikus algoritmusát használtuk 100-as populáció mérettel, 100 generáción keresztül futtatva. Hibafüggvényként a Mean squared error excluding spikes-t, illetve a Spike Count függvények 1:1 arányú kombinációját használtuk.

Az eredeti és az algoritmus által talált paraméterek, azok értéktartományával együtt az 1. táblázatban látható, míg a két adatsor összevetése a 4. ábrán látható, a legalacsonyabb hibaérték generációról generációra mért változásával együtt. Érdekes megjegyezni, hogy az algoritmus igen jól illeszkedő eredményt adott, a kapott paraméterek ellenben nem esnek egybe az eredeti értékekkel. Feltételezhető tehát, hogy különböző paraméter kombinációk képesek egymáshoz nagyon hasonló mérési eredményeket produkálni az általunk használt protokoll alkalmazása mellett. Hogy igazoljuk ezt a feltételezést, elvégeztük az optimalizációt különböző, véletlenszerű kiindulási populációk felhasználásával. Az így kapott eredmények a hibát tekintve lényegében azonosnak tekinthetőek, ellenben a program által meghatározott paraméterek igen különbözőek voltak (lásd később). Ezek az eredmények alátámasztják azt a következtetést, mely szerint a Hodgkin-Huxley modell paraméterei nem határozhatóak meg egyértelműen, egyszerű áraminjekcióra adott válaszok alapján.



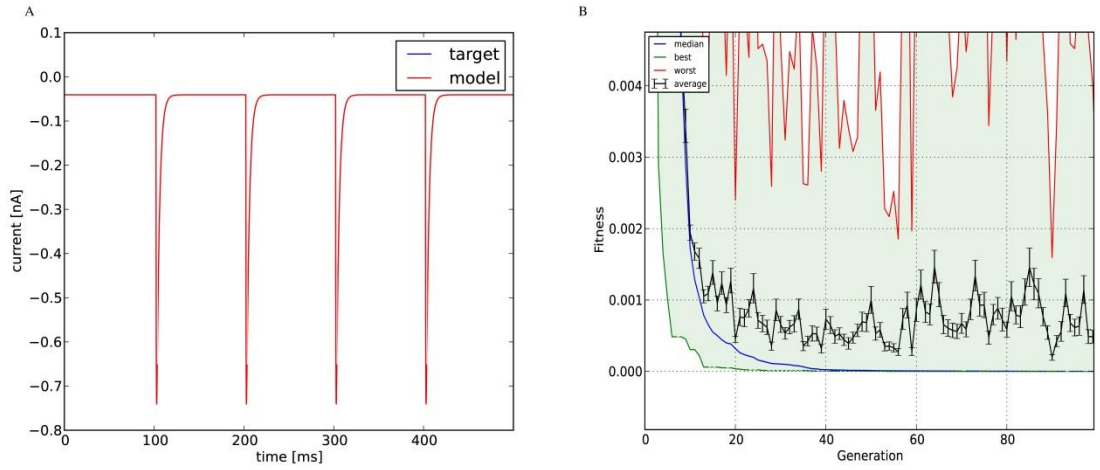
4. ábra Az első teszt eset eredményei. Az A ábrán a mesterségesen generált céladatsor (kék) és a rá legjobban illeszkedő eredmény (piros) látható. A B a hiba minimumának és mediánjának változása látható az optimalizáció alatt.

2. Egykompartmentumos modellben alapvető szinaptikus paraméterek visszaállítása voltage clamp mérés alapján.

Céladatként ismét mesterséges adatot használtunk, egy az előző esethez nagyon hasonló modellből kiindulva, kibővítve azt egy konduktancia alapú dupla exponenciális időbeli lefutással rendelkező szinapszissal (felfutási idő=0.3 ms, lecsengési idő=3 ms, maximális konduktancia=10 nS, késleltetés=2 ms). Ezt a modellt ezután egy, a szinapszison keresztül érkező AP sorozattal ingereltünk, mely 4 AP-t tartalmazott minden 100 ms-os szakaszban. A feladat a szinapszis 4 paraméterének visszaállítása volt.

Mivel a szinaptikus paraméterek a NEURON implementációjában nem az idegsejt modell részei (egy belső NetCon objektumba tartozik), az Optimizer által használt heurisztika pedig csak a modell részét képező paramétereket képes automatikusan detektálni, így egy egyszerű felhasználói függvényt használtunk a paraméterek kezelésére.

Az optimalizációt a Mean Squared Error függvény felhasználásával végeztük, ismét a genetikus algoritmust használva 100-as populáció mérettel és 100 generációval. Ezekkel a beállításokkal az Optimizer nagy pontossággal meghatározta a keresett szinaptikus paramétereket.(2. táblázat; 5. ábra).



5. ábra A második teszt eset eredményei. Az A ábrán a mesterségesen generált céladatsor (kék) és a rá legjobban illeszkedő eredmény (piros) látható. A B ábrán a hiba statisztikáinak változása látható az optimalizáció alatt.

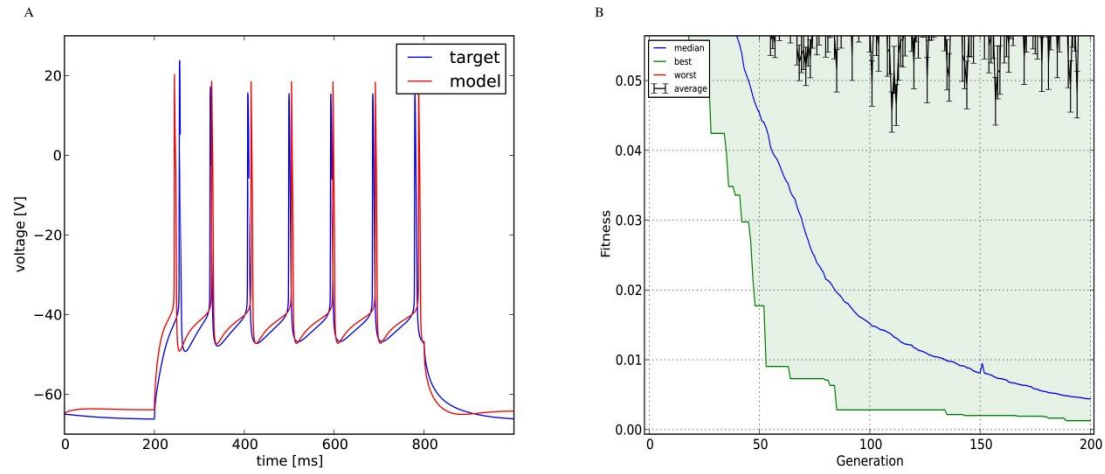
3. Egy hat kompartmentumból álló CA1 piramissejt modellben a szomatikus, feszültségvezérelt csatornaparaméterek optimalizációja egy morfológiailag és biofizikailag rekonstruált CA1 piramissejt modellből származó adatsor alapján.

A céladatsort egy biofizikailag pontos, részletes morfológiával rendelkező hipokampális CA1 piramissejt modellből (11) nyertük, a szómára adott 200 pA-es stimulus hatására. A kísérlet 1000 ms-ig tartott, a stimulus a 200 ms-ban indult és 600 ms-ig tartott.

Az egyszerűsített modell struktúráját az optimalizáció előtt, attól függetlenül hoztuk létre a komplex modell ágainak klaszterezésével és összevonásával. A klaszterezést a modell küszöb alatti stimulusra adott válaszainak amplitúdói alapján végeztük, az egy klaszterbe eső szakaszokat pedig összevontuk. Ezzel a módszerrel egy olyan modellt kaptunk, mely mindössze hat kompartmentumból (egy szomatikus; egy, a bazális dendrithez tartozó; és négy, az apikális dendrit fából származó) épült fel. Az így kapott kompartmentumok csatornaparamétereit a részletes modell paramétereinek átlagolásával kaptuk meg. Optimalizálandó paramétereknek a szóma csatornaparamétereit választottuk, míg a többi kompartmentum csatornaparamétereit és a passzív membrán karakterisztikát a geometriai jellemzőkkel egyetemben változatlanul hagytunk.

Mivel a probléma meglehetősen komplex, a megoldáshoz hat különböző hibafüggvény kombinációját használtuk fel: Mean Squared Error excluding spikes (0.2-es súly), Spike Count (0.4-es súly), ISI differences, AP amplitude, latency to first spike, és AHP depth (mind 0.1-es súlyozással). Algoritmusnak ismét a genetikus algoritmust választottuk, 200 generáció és generációnként 300 egyed felhasználásával. Ezekkel a beállításokkal az algoritmus a keresési tér nagyobb részét volt képes bejárni, melyre a probléma összetettsége és magas dimenziója miatt

volt szükség. Az algoritmusnak így a feladat komplexitásának ellenére sikerült elfogadható megoldást adnia, mely megoldás jó illeszkedik a céladatra. (6. ábra; illetve 11. ábra).



6. ábra A harmadik teszt eset eredményei. Az A ábrán a mesterségesen generált céladatsor (kék) és a rá legjobban illeszkedő eredmény (piros) látható. A B ábrán a hiba statisztikáinak változása látható az optimalizáció alatt.

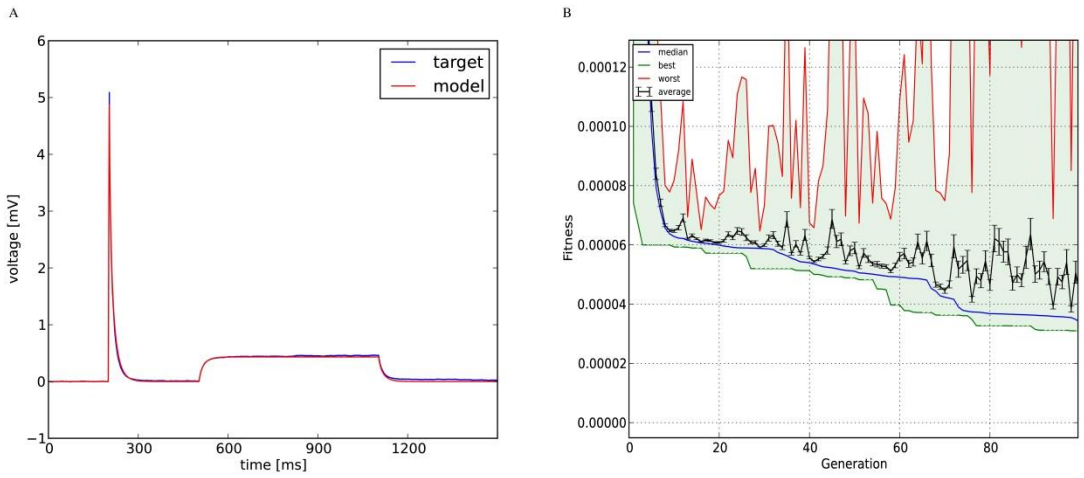
4. Egy morfológiailag részletes CA1 piramissejt passzív paramétereinek illesztése komplex stimulust alkalmazva, kísérleti adatok alapján.

Ebben az esetben egy morfológiailag részletes, passzív hipokampális CA1 piramissejt modell passzív paramétereit próbáltuk illeszteni a rekonstrukció alapjául szolgáló sejten végzett mérések alapján.

A sejtet egy rövid (3 ms, 500 pA) és egy hosszabb áram impulzussal (600 ms, 10 pA) stimulálták, ennek reprodukálásához az Optimizer “Szabad hullámforma” funkcióját kellett használnunk.

Az optimalizációt három paraméteren, a membrán kapacitáson és ellenálláson, valamint az axiális ellenálláson kívántuk elvégezni. Mivel ez a gyakorlatban azt jelentette, hogy az összes szekcióban optimalizáljuk a cm, Ra és g_pas paramétereket, illetve az e_pas paramétert mindenhol 0-nak kellett tekintenünk (mivel az alapvonali feszültséget kivontuk a mérési adatokból), létrehoztunk egy felhasználói függvényt, mely elvégezte számunkra a paraméter megfeleltetést a globális és lokális paraméterek közt.

A probléma megoldása során a Mean Squared Error függvényt használtuk, illetve a genetikus algoritmust 100 generáció és 100-as populáció méret felhasználásával. Az ábrákon jól látható, hogy a program nagyon pontos illeszkedést mutató eredményt adott.(7. ábra; illetve 12. ábra).



7. ábra A negyedik teszt eset eredményei. Az A ábrán a mesterségesen generált céladatsor (kék) és a rá legjobban illeszkedő eredmény (piros) látható. A B ábrán a hiba érték statisztikáinak változása látható az optimalizáció alatt.

5. Egy absztrakt adaptive exponential integrate-and-fire típusú idegsejt modell illesztése egy valódi CA3 piramissejtből származó, több, különböző amplitúdójú stimulusra adott szomatikus feszültségválasz alapján.

Ebben az esetben egy AdExpIF modellt próbáltunk illeszteni négy feszültségsorra, melyet egy valódi CA3 piramissejtből kaptunk. A mérések 1100 ms hosszúak voltak, mindet 5 kHz frekvenciával mintavételeztük. Az áraminjekciók amplitúdói 0.30 nA, 0.35 nA, 0.40 nA és 0.45 nA volt. A modell következő paramétereit próbáltuk optimalizálni:

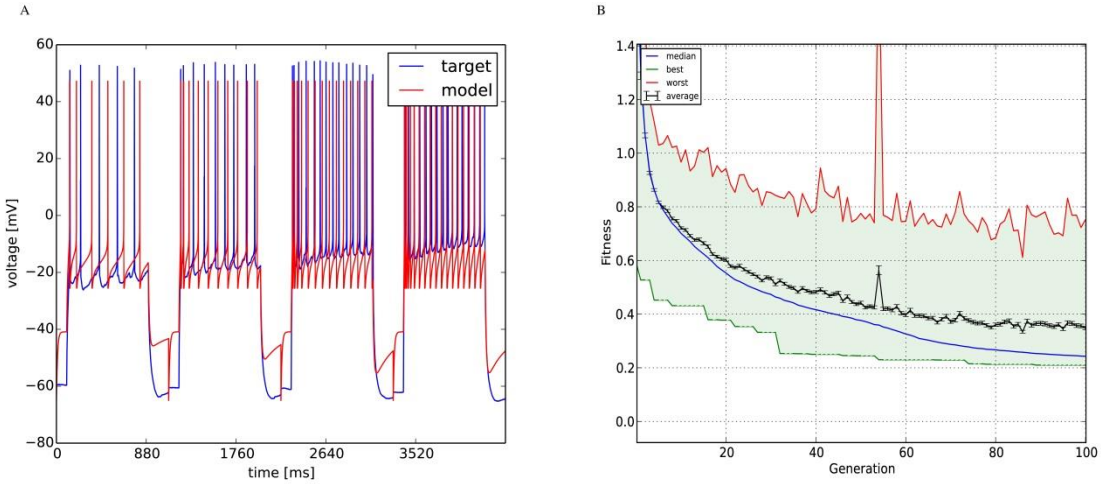
- membrán kapacitás (capacitance)
- szivárgási konduktancia (leak conductance)
- szivárgási reverz potenciál (leak reversal potential)
- feszültség küszöb (threshold voltage)
- reset feszültség (reset voltage)
- refrakter periódus (refractory period)
- i-v görbe meredeksége az exponenciális szakaszon (steepness of exponential part of the current-voltage relation)
- küszöb alatti adaptációs konduktancia (subthreshold adaptation conductance)
- akciópotenciál adaptációs áram (spike adaptation current)
- adaptációs időállandó (adaptation time constant)

Mivel a modell a paramétertartomány egy részén numerikus instabilitást mutat, bizonyos megkötéseket kellett tennünk a paraméterek értékét illetően, ezért ismét definiáltunk egy felhasználói függvényt.

Az optimalizáció végrehajtásához a következő függvények kombinációját használtuk (azonos súlyozással):

- Spike Count
- Mean Square Error II (sub-threshold MSE)
- Latency to 1st Spike
- AP Overshoot
- AHP Depth

A modell illesztését 500 egyed/generáció és 100 generáció felhasználásával végeztük (8. ábra). Látható, hogy a kapott modell elfogadhatóan képes utánozni a valódi idegsejt tüzelését, azonban a küszöbalatti viselkedés esetében már nem ez a helyzet (ez nagyrészt a modell jellegéből adódik, nem pedig az illesztési módszer jellegéből).



8. ábra Az ötödik teszt eset eredményei. Az A ábrán a mesterségesen generált céladatsor (kék) és a rá legjobban illeszkedő eredmény (piros) látható. A B ábrán a hiba statisztikáinak változása látható az optimalizáció alatt.

Összevetés más szoftverekkel

Az irodalomban leggyakrabban idegsejt modell optimalizációra használt szoftvereszközök a NEURON a GENESIS és a Neurofitter. A továbbiakban röviden összehasonlítjuk ezeket a programokat az általunk fejlesztett Optimizerrel.

A fenti programok közül az Optimizer mellett egyedül a NEURON rendelkezik olyan grafikus felülettel, mely közvetlenül integrálódna egy gyakran használt szimulátorba (NEURON). A NEURON grafikus felülete rendelkezik néhány nagyon hasznos funkcióval, például lehetőséget ad arra, hogy tetszőleges szimulációk eredményeit kombináljuk, vagy, hogy intuitív módon határozzuk meg a bemenő adatsor számunkra lényeges részét. Ezek a funkciók jelenleg nem elérhetők az Optimizerben. Mivel a NEURON grafikus felülete igen jól felépített, több

kutatócsoport is sikerrel alkalmazta modell illesztési problémák megoldására, ellenben ezek általában egyszerű problémák voltak, melyek kevés paramétert tartalmaztak. Mivel a program csupán egy, lokális algoritmust tartalmaz (21), komplex problémák megoldására csak korlátozottan alkalmas. Mivel a NEURON az egyik legtöbbet használt szimulátor, több kísérlet is történt a modell optimalizációs modul kibővítésére. Az egyik ilyen kiegészítő egy genetikus algoritmussal bővíti a NEURON-t, sajnos azonban beállítása és használata (ellentétben az Optimizerrel, vagy a NEURON beépített optimalizációs moduljával) igen jelentős mennyiségű kódolást igényel, ami nagyban nehezíti a használatát.

A GENESIS szimulátor (7) optimalizációs része nem érhető el grafikus interfészen keresztül, maga a modell illesztés nagy mennyiségű kódolást igényel a szimulátor saját programnyelvében (ezzel szemben egy már adott modellből kiindulva a NEURON és az Optimizer lényegében semmilyen kódolást nem igényel). Éppen ezért egy új optimalizációs probléma megoldása GENESIS-ben igencsak időigényes lehet, és a folyamat sok hibalehetőséget tartalmaz. Másrészről viszont a program több, nagy teljesítményű optimalizációs algoritmust is kínál (tartalmaz genetikus algoritmust és szimulált hőkezelésen alapuló implementációt is), melyek kiváló eredményeket képesek produkálni, annak ellenére, hogy mind a szimulátor, mind az optimalizációért felelős modulja igen régi. Ezen felül a szoftver a PGENESIS modulon keresztül támogatja a párhuzamosítást is (ezt az Optimizer csak az inspyred algoritmusok esetében támogatja, és jelentős kódolást igényelhet a megvalósítása). Költség-függvények szempontjából a GENESIS kínálata sem tekinthető bőségesnek, csupán egyetlen beépített függvényt tartalmaz, egyéb függvényeket a felhasználónak kézzel kell hozzáadnia. Meg kell jegyeznünk azonban, hogy költség-függvény, mely az akciós potenciálok időzítését illeszti, igen szofisztikált implementációval rendelkezik, és önmagában alkalmazva is nagyon jó eredményeket képes produkálni.

A Neurofitter (14) egy, az Optimizerhez sokban hasonló, általános célú modell optimalizációs eszköz. Lényeges különbség azonban a két szoftver között, hogy a Neurofitter nem rendelkezik grafikus felülettel, így a probléma definiálását egy konfigurációs fájlon keresztül kell elvégezni, ami igen kényelmetlen és sok hibalehetőséget hordoz magában. Az Optimizerhez hasonlóan a Neurofitter is több algoritmus implementációját tartalmazza, ellenben ismét csak egy költség-függvény (a PPTD módszer) áll rendelkezésünkre. Ez azért is problémás, mert szemben a GENESIS hibafüggvényével, mely általában jól teljesített, a PPTD bizonyos problémákat jellegéből adódóan nem képes helyesen kezelni. Mivel ez a költség-függvény a fázistérben hasonlítja össze a bejövő adatot és az aktuális szimuláció kimenetét, így a probléma időfüggését nem képes kezelni, ez pedig sok esetben azt eredményezi, hogy az illesztett adatsor időben eltolódik az optimálishoz képest (lásd később az eredményeknél). A párhuzamosítás terén a

program elfogadhatóan teljesít, mivel az MPI protokollon keresztül a párhuzamosítás több formáját is támogatja.

Végezetül meg kell jegyeznünk, hogy van néhány olyan funkció, mely a fenti egyik programban (beleértve az Optimizert is) sem érhető el, erre talán a legfontosabb példa az úgynevezett multi-objective optimalizáció (a fenti szoftverek mind úgynevezett single-objective algoritmusokat tartalmaznak), melyet több esetben is alkalmaztak sikeresen modell illesztési problémák megoldására (8). Ilyen megoldást egyik általunk tesztelt szoftver sem kínál, ugyanakkor az Optimizer esetében az inspyred modulon keresztül ez egyszerűen megvalósítható lenne, mivel a csomag tartalmaz egy multi-objective optimalizációs keretrendszert is.

A fenti összehasonlításon túl szerettük volna a különböző modellillesztő szoftverek teljesítményét kvantitatív módon is összevetni. Éppen ezért az eredmények szekcióban bemutatott eseteket megkíséreltük megoldani NEURON, GENESIS és Neurofitter használatával is.

Sajnálatos módon nem sikerült az összes teszt esetet implementálnunk minden szoftverhez, mivel a GENESIS nem támogatja az AdExpIF modelleket, illetve a Neurofitter nem volt képes a modell numerikus instabilitását kezelni, így végül csak az 1-4 eseteket implementáltuk (az implementáció a GENESIS esetében nagyon sok kódolással és debugolással járt).

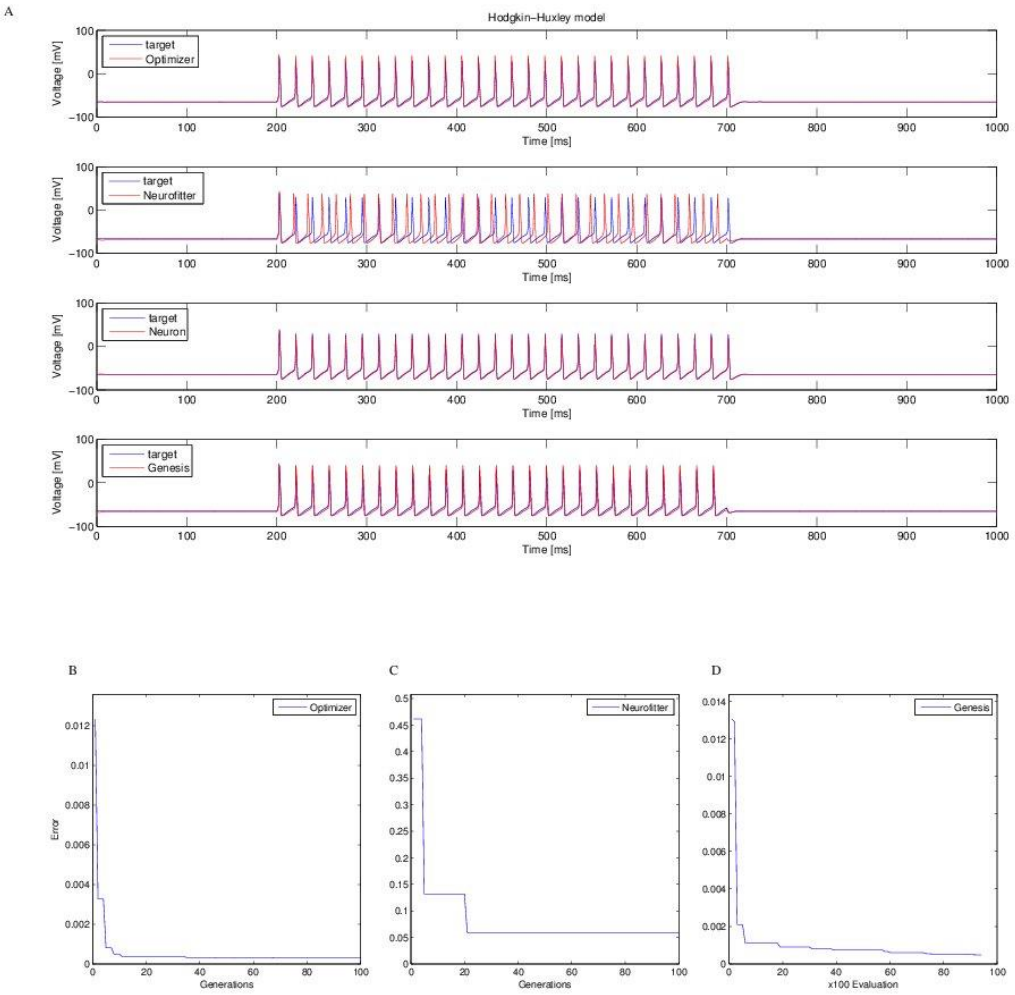
A teszt esetek összehasonlításakor az optimálisnak ítélt paraméterekkel generált adatsorok átlagos négyzetes hibáját, illetve a tüzelő modellek esetében a generált akciópotenciálok számát használtuk.

		Optimizer	Neurofitter	NEURON	GENESIS
HH	MSE(mV^2)	0.0033	0.0438	7.32E-04	0.0016
	Spike count (28)	28	32	28	28
VC	MSE (nA^2)	1.73E-07	0.0052	2.68E-05	6.29E-07
CA1 PC simple	MSE (mV^2)	0.0069	0.0125	0.0092	0.0028
	Spike count (7)	7	10	6	7
CA1 PC morphology	MSE (mV^2)	3.10E-05	3.09E-04	3.29E-05	3.58E-05

1. táblázat A legjobb megoldások hibáinak összevetése az 5 teszt esetben, az átlagos négyzetes hiba (MSE), illetve a tüzelő modellek esetében az akciópotenciálok számának felhasználásával(a zárójelben megadott értékek a céladatsorban található AP-k számát jelzik)

1.Egy kompartmentumos Hodgkin-Huxley modell konduktanciáinak (Na, K, szivárgási) meghatározása egy küszöb feletti áram injekcióval. Látható, hogy a négy szoftverből három sikeresen határozott meg olyan paramétereket melyek mind négyzetes hiba, mind az

akcióspotenciálok számát és időzítését tekintve jól illeszkedik a cél adatsorra. A Neurofitter esetében tapasztalt gyengébb teljesítmény feltételezhetően a szoftver által használt PPTD módszer jellemzőiből adódik (1. táblázat; 9. ábra).



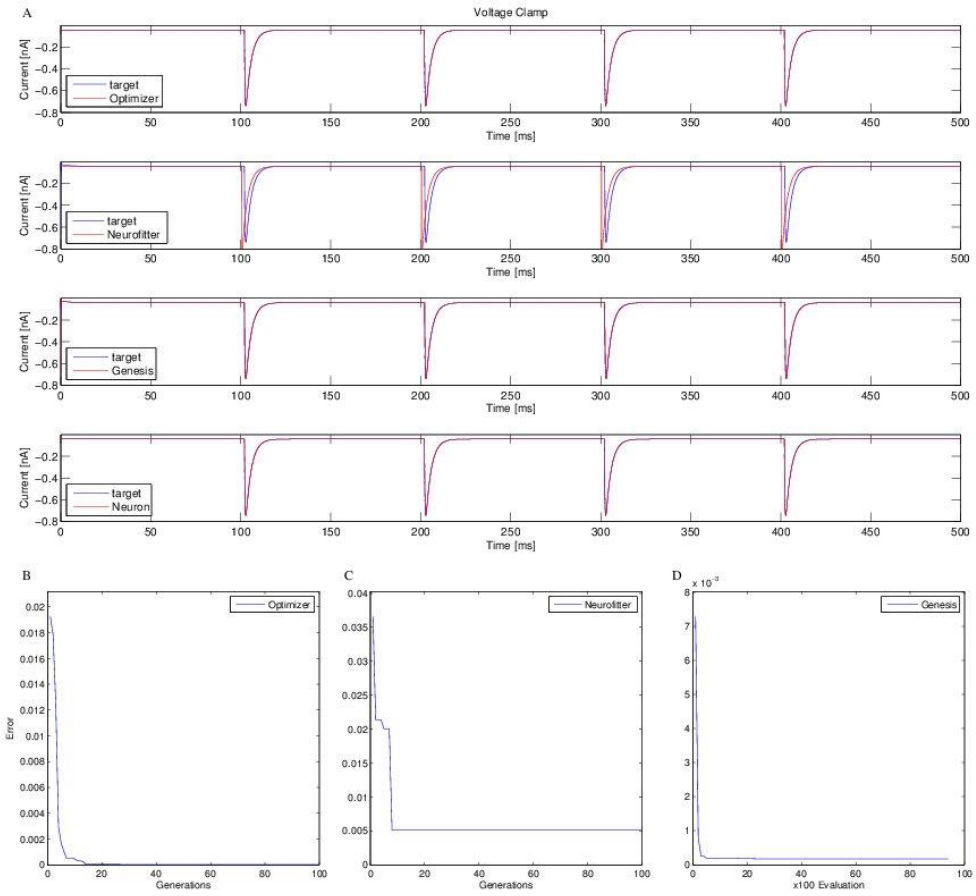
9. ábra A különböző optimalizációs szoftverek teljesítményének bemutatása az első teszt eseten keresztül. Az A ábrán a céladatsor és az optimalizációk eredményeként kapott adatsorok láthatóak. A B-D ábrákon a hiba minimumának változása látható a generációkon (illetve a GENESIS esetében minden 100. modell kiértékelés után) keresztül

Érdekes megvizsgálnunk magukat a kapott paramétereket is (2. táblázat). Azt tapasztalhatjuk, hogy a különböző programok nagyon eltérő paramétereket határoztak meg, illetve láthatjuk, hogy a paraméterek jelentősen eltérnek az eredeti paraméterektől is, ez pedig ismételten arra utal, hogy a paraméter identifikáció még az egyszerűnek látszó Hodgkin-Huxley modell esetében sem triviális feladat.

paraméterek	eredeti értékek	Optimizer	Neurofitter	NEURON	GENESIS
gnabar_hh	0.12	0.4242	0.5014	0.2687	0.0968
gkbar_hh	0.036	0.101	0.1191	0.0714	0.0294
gl_hh	0.0003	0.000769	0.000772	0.000313	0.00032

2. táblázat Az első teszt esetben a legjobb illesztéshez tartozó paraméterértékek összehasonlítása. A fenti konduktancia sűrűségek S/cm^2-ben értendőek.

2. Egykompartmentumos modellben, alapvető szinaptikus paraméterek meghatározása voltage clamp mérés alapján: az Optimizer és a GENESIS lényegében tökéletes megoldást adott mind a hiba (1. táblázat), mind a paraméter értékek (3. táblázat) tekintetében; a NEURON megoldása egy kicsivel pontatlanabb volt az előző két szoftverhez viszonyítva, míg a Neurofitter megoldása lényegesen nagyobb hibát eredményezett.(1. táblázat; 10. ábra).

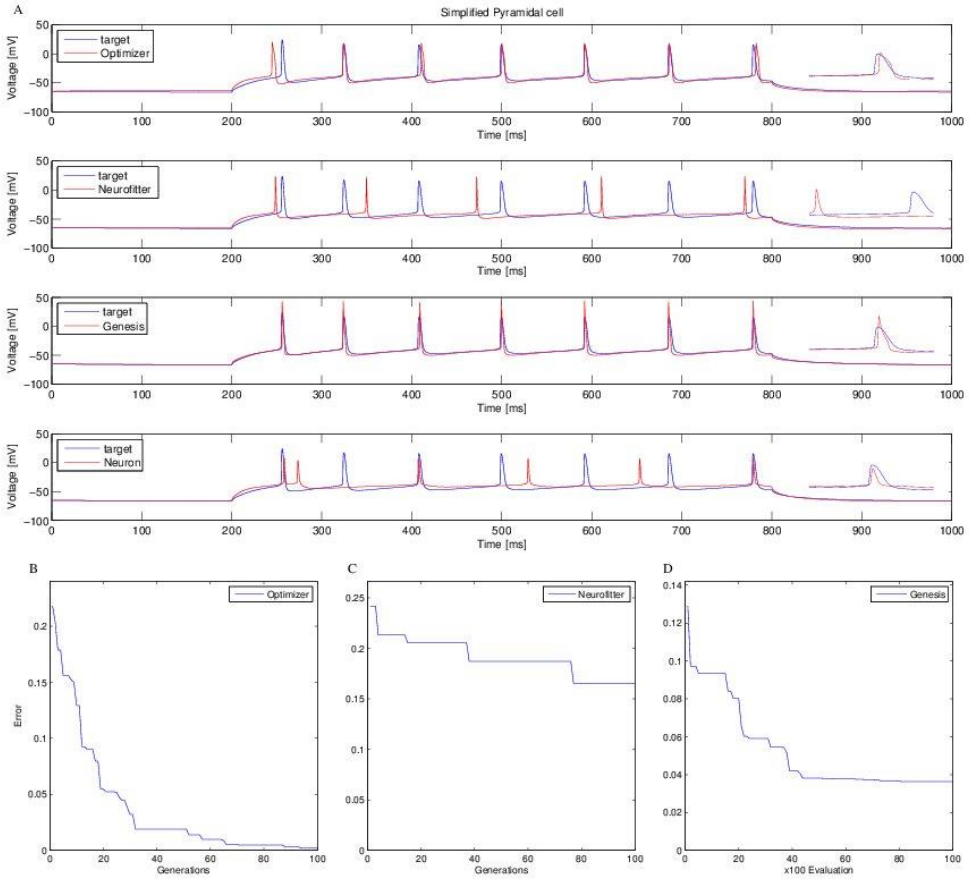


10. ábra A különböző optimalizációs szoftverek teljesítményének bemutatása a második teszt eseten keresztül. Az A ábrán a céladatsor és az optimalizációk eredményeként kapott adatsorok láthatóak. A B-D ábrákon a hiba minimumának változása látható a generációkon (illetve a GENESIS esetében minden 100. modell kiértékelés után) keresztül

paraméterek	eredeti értékek	Optimizer	Neurpfitter	NEURON	GENESIS
tau1 (ms)	0.3	0.3006	0.0406	0.3002	0.2561
tau2 (ms)	3	2.996	2.994	2.9998	3.0438
weight (uS)	0.01	0.010002	0.01209	0.009997	0.01014
delay (ms)	2	1.9783	0.3672	1.9863	2.0288

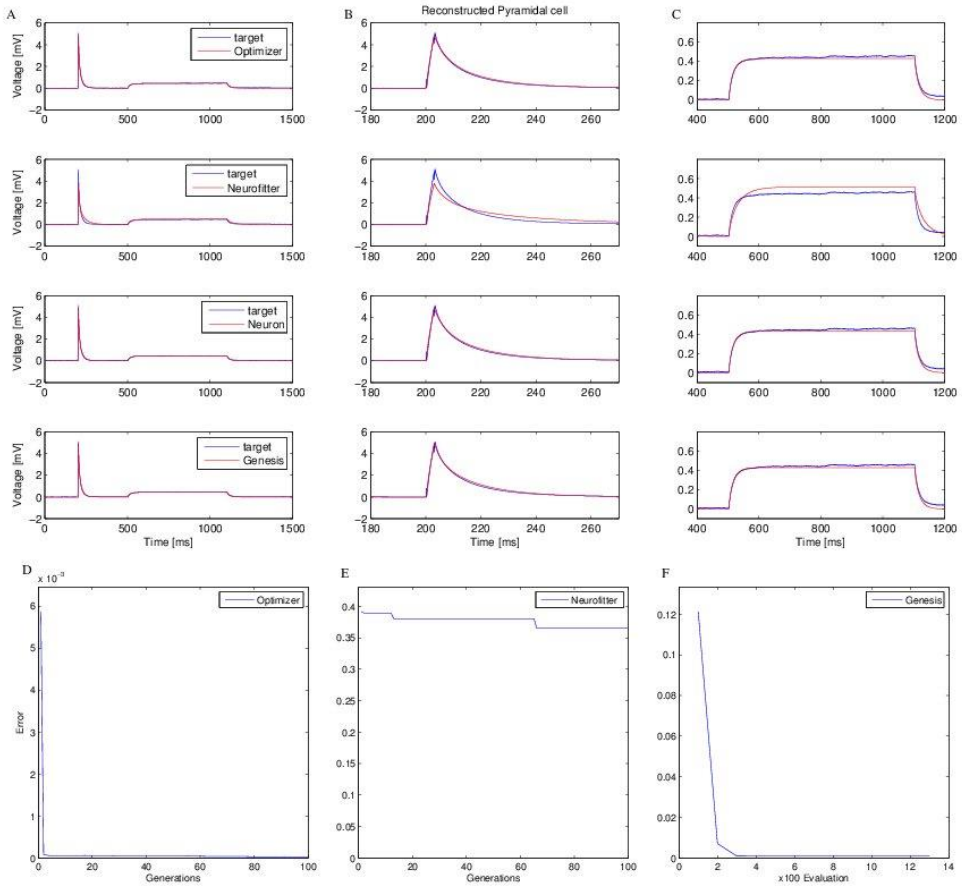
3. táblázat A második teszt esetben a legjobb illesztéshez tartozó paraméterek összevetése.

3. Egyszerűsített, hat kompartmentumos modell szomatikus, feszültségfüggő csatornáinak paraméterillesztése egy morfológiailag és biofizikailag rekonstruált CA1 piramissejt modellből származó szomatikus feszültségválasz alapján. Ezt a rendkívül komplex problémát a legnagyobb sikerrel a GENESIS-nek sikerült megoldania, mind a hiba mértékét, mind az akciópotenciálok számát és időzítését tekintve (lásd az első táblázatban), de az Optimizer is elfogadható eredményeket produkált, lényegében csak az akciópotenciálok időzítését nem sikerült pontosan beállítania a programnak (11. ábra). A NEURON és a Neurofitter által talált megoldások lényegesen gyengébb eredményeket produkáltak, itt már nem csak az akciópotenciálok időzítése csúszott el, hanem azok számában és alakjában is nagy eltérés mutatkozik a céladathoz képest. A gyengébb eredmények a NEURON esetében a használt lokális algoritmussal magyarázható, míg a Neurofitter esetében a hibafüggvény okolható a látható eltérésekért. A kapott eredmények alapján elmondható, hogy a GENESIS hibafüggvénye nagyon hatékony, mivel az Optimizer esetében hat hibafüggvény kombinálására volt szükség hasonló eredmények eléréséhez. Elmondható tehát, hogy a hibafüggvények száma nem feltétlenül garantálja a jó eredményeket.



11. ábra A különböző optimalizációs szoftverek teljesítményének bemutatása a harmadik teszt eseten keresztül. Az A ábrán a céladatsor és az optimalizációk eredményeként kapott adatsorok láthatóak. A B-D ábrákon a hiba minimumának változása látható az eddigiekhez hasonlóan

4. Morfológiailag rekonstruált CA1 piramissejt modell passzív paramétereinek illesztése komplex áram stimulussal kapott kísérleti eredmények alapján. Az Optimizer, a NEURON, és a GENESIS mind elfogadható eredményeket adott (áltagos négyzetes hibát használva hibafüggvényként), míg a Neurofitter ismét rosszul teljesített (12. ábra).



12. ábra A különböző optimalizációs szoftverek teljesítményének bemutatása a negyedik teszt eseten keresztül. Az A-C ábrákon az optimális adatsor és a céladatsor illeszkedését vizsgálhatjuk meg, a B és C ábrákon részletesen is láthatjuk a két stimulusra adott válaszokat. A D-F ábrák az eddigiekhez hasonlóan a hiba minimumának változása látható.

Összefoglalva elmondhatjuk tehát, hogy az Optimizer minden esetben remekül teljesített, az alkalmazott hiba metrikák alapján a legjobb, vagy a második legjobb eredményt adta az összehasonlított szoftverek közül. Látható továbbá, hogy a jó eredmények mellett a problémák igen széles skáláját képes lefedni, különösen igaz ez akkor, ha figyelembe vesszük, hogy az egyik esetet ki kellett hagynunk az összehasonlításból, mivel egyetlen másik eszköz sem volt képes a probléma helyes kezelésére.

Áttekintve a többi szoftver eredményeit, azt láthatjuk, hogy a GENESIS, bár nem támogatja az integrate-and-fire típusú modelleket, nagyon jó eredményeket produkált a többi esetben, feltételezhetően a hatékony hibafüggvényének és algoritmusának köszönhetően.
A NEURON esetében az eredmények már nem ilyen pozitívak. Ahogy arra számítottunk, a program által használt lokális algoritmus csupán alacsony dimenziós esetekben működött hatékonyan, akkor is csak olyan esetekben, amikor a hibafüggvény feltételezhetően egy, jól meghatározható minimummal rendelkezett.

A Neurofitter gyenge teljesítménye nagy meglepetést okozott, mivel hasonló problémákon korábban sikerrel alkalmazták (14). Feltehetően az szoftver finomhangolásával, illetve egy másik hibafüggvény használatával jobb eredményeket érhettünk volna el, különösen a komplexebb problémák esetében, sajnos azonban a program beállítása nagyon körülményes, az újabb hibafüggvények implementációját pedig a program összetettsége nehezíti.

További tesztek

Az első két teszt esetben azt is megvizsgáltuk, hogy az Optimizerrel kapott eredmények mennyiben függnek a kiindulási populáció megválasztásától. Az alább látható (4-5. Táblázat.) eredmények igazolják azt a feltételezést, hogy az első probléma esetében több, nagyjából azonosan jó megoldás létezik, mivel láthattuk, hogy a jó hibaértékek ellenére egyik szoftver sem tudta megtalálni az eredeti paramétereket, illetve a kiindulási populáció variálásával nagyjából azonos hibával rendelkező, de különböző paraméter-szetteket kaptunk eredményül (1-2. Táblázat).

A második teszt esetben azt láthatjuk, hogy a különböző kiindulási populációk lényegében azonos eredményekhez vezetnek, mely alátámasztja azt a hipotézist is, mely szerint ez a probléma egyetlen, jól definiált minimummal rendelkezik az adott paramétertartományokon. Ez egyben megmagyarázná azt is, hogy miért tudott ezzel a problémával megbirkózni a NEURON lokális algoritmusa is.

Hodgkin-Huxley			
random seed	1111	4231	1234
gkbar_hh	0.13181976	0.12158285	0.10098785
gl_hh	0.001	0.00097886	0.0007685
gnabar_hh	0.57729157	0.52743807	0.42418448
fitness értékek	0.00037693	0.00032553	0.00029689

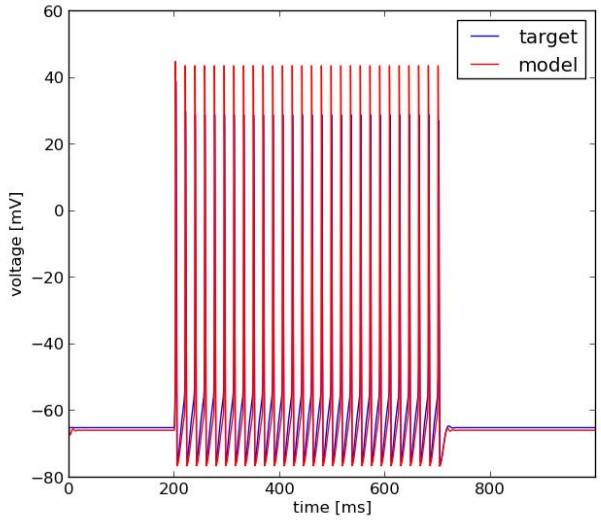
4. táblázat A kiindulási populáció variálásának hatása a fitness értékekre és a paraméterekre az első teszt esetben

Voltage Clamp			
random seed	1111	1234	4231
weight	0.00998522	0.01000202	0.00989332
delay	1.96436407	1.97830968	1.9652071
tau_rise	0.29634562	0.30064745	0.32647313
tau_decay	3.00647985	2.99599555	2.94367798
fitness értékek	3.38E-007	1.73E-007	4.69E-006

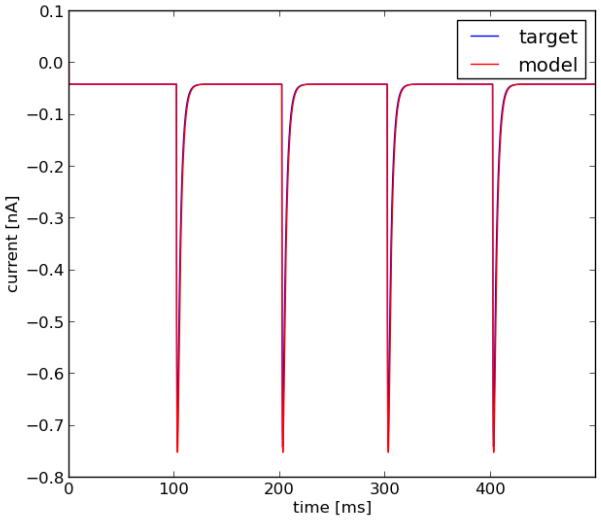
5. táblázat A kiindulási populáció variálásának hatása a fitness értékekre és a paraméterekre a második teszt esetben

Korábban azt állítottuk, hogy a gyakori idegsejt modell optimalizációs problémák nem oldhatók meg primitív algoritmusokkal, azoknál szofisztikáltabb megoldásra van szükség. Hogy ezt az állításunkat bizonyítsuk, az alábbiakban bemutatjuk a véletlen bolyongás (random search) algoritmussal kapott eredményeinket az első két teszt eseten (13-14. ábra). Mindkét esetben 10000 iterációt engedélyeztünk, látható, hogy az első esetben így elfogadható eredményeket sikerült kapnunk (MSE=0.00819 mV^2), de tekintetbe véve, hogy épp az előbb bizonyítottuk, hogy az eset több, nagyjából egyforma lokális minimummal rendelkezik, nem meglepő, hogy véletlenszerűen is sikerült egy elfogadható megoldást találnunk. A második eset már látványosan bizonyítja az állításunkat, hiszen, ahogy azt várni lehetett, a véletlenszerű pont választással nem lehetséges megtalálni a globális minimumot, ezt a három nagyságrenddel nagyobb hiba jól demonstrálja (MSE=5.2517×10^{-4} nA^2).

Figyelembe véve, hogy mindkét eset meglehetősen primitív modellekkel dolgozik, ráadásul csak három, illetve négy paramétert próbáltunk illeszteni, összességében elmondhatjuk, hogy komplex problémák megoldásához szükség van összetettebb algoritmusok használatára.



13. ábra A Random search algoritmus eredménye az első teszt eseten. Látható, hogy illeszkedés szempontjából ez a primitív algoritmus is jó eredményt ad



14. ábra A Random search algoritmus eredménye a második teszt eseten. Bár az illesztés látszólag pontos, valójában több nagyságrenddel rosszabb az eredmény, mint a komplexebb algoritmusok esetében

Távlati tervek

Ez eddigiek során bemutattuk az Optimizer jelenlegi állapotát. Ahogy a fenti eredményekből kiderül, a program már most nagyon sok hasznos funkcióval rendelkezik, ráadásul hatékonyan képes megoldani igen különböző komplexitású problémákat, akár olyanokat is, melyeket a már létező eszközökkel nem, vagy csak nagyon nehezen lehetett kezelni. A programot saját kutatásokban az intézeten belül már sikerrel alkalmaztuk, de természetesen célunk, hogy egy

nagyobb közösséghez is eljuttassuk a szoftvert, így segítve az idegsejt modellezők munkáját, illetve az olyan elektrofiziológiai kísérleteket végző kollegák munkáját, akik a modellezést, mint kiegészítő módszer alkalmazzák kutatásaikban.

Éppen ezért a tőlük beérkező visszajelzések alapján szeretnénk továbbfejleszteni az Optimizert, mind funkció, mind használhatósági szempontok alapján. Ezen felül reméljük, hogy a program rugalmas és moduláris felépítése a felhasználókat arra sarkalja, hogy ők maguk is továbbfejlesszék a programot az általuk használt szimulátorokkal és protokollokkal, illetve a nekik tetsző algoritmusokkal és szükséges hibafüggvényekkel.

A felhasználóktól érkező visszajelzésektől függetlenül természetesen mi is tervezünk még fejlesztéseket eszközölni a programon, ezek közül említenénk most néhányat. Ahogy azt a bemutatásban is hangsúlyoztuk, a program egyik legnagyobb előnye a kényelmes, áttekinthető grafikus felület. Éppen ezért szeretnénk kibővíteni a grafikus felületet, hogy még több problémát oldhassunk meg ezen keresztül. Konkrétabban szeretnénk a programot további szimulátorokkal kibővíteni, hiszen a grafikus felület lehetőségeit a külső szimulátor nem tudja teljes mértékben kihasználni. Talán a leglogikusabb lépés a PyNN (22) keretrendszer támogatása lenne, mivel ez több népszerű szimulátor (NEST, Brian és PCSIM) kezelését is lehetővé teszi egy egységes felületen keresztül.

Szeretnénk továbbá kibővíteni a lehetséges céladatok listáját komplexebb adattípusokkal, így a mostani folytonos áram és feszültség adatok mellett felhasználhatnánk diszkrét eseményeket (AP időzítések) vagy absztrakt jellemzőket, akár egymással kombinálva is. Ebben segítségünkre lehet a Python alapú neo programcsomag (23), mely egy adatreprezentációs keretrendszert tartalmaz.

Célszerű lenne a program parancssoros interfészét is továbbfejleszteni olyan módon, hogy például azonos típusú, de eltérő paraméterekkel rendelkező modellt automatikusan tudjunk illeszteni több sejtből származó adatokhoz (akár a grafikus felületen keresztül).

Végezetül érdemes lenne a modell illesztés folyamatának párhuzamosíthatóságát megvizsgálni, mivel ezek az optimalizációk nagyon időigényesek lehetnek. Elsőként szeretnénk megvizsgálni a felhasznált modulok (scipy, inspyred) által kínált lehetőségeket, illetve a későbbiekben a szimulátorok hatékonyságát is meg szeretnénk vizsgálni. A hatékonyság növelésének érdekében a számításigényes lépéseket (költség-függvény kiértékelés) szeretnénk vektorizálni.

Összegzés

Az eddigiek során bemutattunk egy új eszközt idegsejt modellek paraméterillesztésére. A modellek paramétereinek meghatározása kritikus fontosságú, ugyanakkor igen komplex és

időigényes lépése biológiailag releváns idegsejt modellek és hálózatok létrehozásában. Ezen felül megfelelően illesztett modellek új lehetőségeket kínálnak fiziológiai adatok kvantitatív elemzésében is. Sajnos azonban a modell illesztés folyamatát nagyban befolyásolják az alkalmazott módszerek implementációinak apró részletei, ezért a megfelelő szoftveres keretrendszer megválasztása kulcsfontosságú lehet. Éppen ezért éreztük szükségét az Optimizer megalkotásának, melyben a legmodernebb algoritmusok kínálta lehetőségeket igyekeztünk elérhetővé tenni egy grafikus felületen keresztül, így létrehozva egy intuitívan, ám eredményesen használható keretrendszert.

Paraméterbecslés valószínűségi keretben

Miután megbizonyosodtunk arról, hogy az Optimizer megbízhatóan, jó pontossággal képes megoldani a legkülönfélébb paraméterillesztési problémákat, áttekintettük a program alternatív felhasználási lehetőségeit. Nem ritka, hogy a kísérleti elektrofiziológia területén tevékenykedő kutatók számítógépes modelleket használnak fel munkájuk felgyorsítása, vagy éppen egyszerűsítése érdekében. Bár az idegsejt modellezés során nyert adatok nem mindig adnak hiteles képet a vizsgált sejtről vagy sejtekről, mégis sok esetben nyújtanak jelentős segítséget a kísérletes munka során, legyen szó akár egy szokatlan mérési eredmény alátámasztásáról, vagy egy nehezen mérhető paraméter értékének megbecsüléséről. A számítógépes modellek hatalmas előnye, hogy a létrehozott neuron modelleken tetszőleges jellegű mérés elvégezhető (szimulálható), míg valós kísérleti körülmények közt a mérhető paraméterek jellege, száma gyakran limitált, nem is szólva arról, hogy egy modellen végzet mérés esetében a mérési protokoll változtatása gyerekjáték a valós sejten végzett mérés módjának megváltoztatásához képest. Éppen ezért lenne roppant hasznos, ha a lehető legnagyobb mértékben tudnánk a létrehozott modelljeinkre és a köréjük felépített keretrendszerekre támaszkodni.

Egy elektrofiziológiai kísérlet megtervezése, az alkalmazott protokoll megválasztása komplex feladat, gondos tervezést igényel. Nagyban tudná segíteni ezt a tervezési folyamatot, ha valamilyen eszközzel előre jelezhetnénk a kiválasztott protokoll hatékonyságát, azaz becslést adhatnánk arról, hogy a mérés mekkora mennyiségű információt hordoz az adott biofizikai jellemzőről. Erre egy konkrét példa lehet, amikor olyan kísérleti eljárást akarunk tervezni, mely a legalkalmasabb egy sejt membránjának passzív paraméterének vagy annak eloszlásának megbecsülésére. Munkánk során egy ilyen eszköz megtervezésének lehetőségeit, problémáit és

korlátait próbáltuk megvizsgálni.

A következőkben bemutatott eredmények mind a NEURON modellező szoftvert használják, így minden paraméter a program alapértelmezett mértékegységében (azaz a membránkapacitás 1 $\mu F/cm^2$ –ben, az axiális ellenállás ohm*cm-ben, a membrán konduktancia pedig S/cm^2 –ben) értendő.

Eloszlásbecslés

Első lépésként azt vizsgáltuk meg, hogy az idegsejt modellezés eszközeivel, illetve az általunk fejlesztett szoftver-elemek felhasználásával lehetséges-e adott mérési eredmények alapján különböző paraméterek poszterior eloszlását meghatározni a kiválasztott sejttípuson belül, mivel ez az eloszlás sok lényeges információt hordozhat (mi a paraméter várhatóértéke, stb.). Ennek érdekében létrehoztunk egy egyszerű valószínűségi keretet, melyben a problémát kezelni tudtuk. Első lépésként a modell paramétereit két csoportra bontottuk; azt a paramétert, melynek eloszlását becsülni próbáljuk (D), és az összes többi paramétert (θ) elkülönítettük egymástól. Értelemszerűen a különböző paraméter eloszlások determinálják a sejt egy adott stimulusra (I) adott válaszát (R). Ezen megfontolások alapján egy adott sejtből, adott stimulusra kapott választ a következőképpen írhatunk fel (1. képlet):

$$Pi(R|D,\theta)$$

ahol:

- R: mérési eredmény
- D: eloszlás paraméterek
- θ: egyéb paraméterek

valószínűségi változók, illetve:

- i: stimulus

Ugyanakkor a modellezés során kapott mérési eredmények tekinthetők egy R' determinisztikus függvénynek, ez alapján a mérési eredményeket felírhatjuk (2. képlet)

$$R = R'(I,D,\theta) + h$$

alakban, ahol a h valamilyen paraméterekkel rendelkező zaj. A bevezetett jelölésrendszert megtartva a következő formulából indultunk ki (3. képlet):

$$Pi(D|R) = \frac{A}{B}$$

ahol (4. képlet)

$$A = Pi(R|D) * P(D)$$

és

P(D) a vizsgált paraméter prior eloszlása (ha nincs előzetes feltételezésünk, akkor használhatunk egyenletes eloszlást is), illetve (5. képlet)

$$Pi(R|D) = \int P(R|D,\theta) * P(\theta) d\theta$$

valamint (6. képlet)

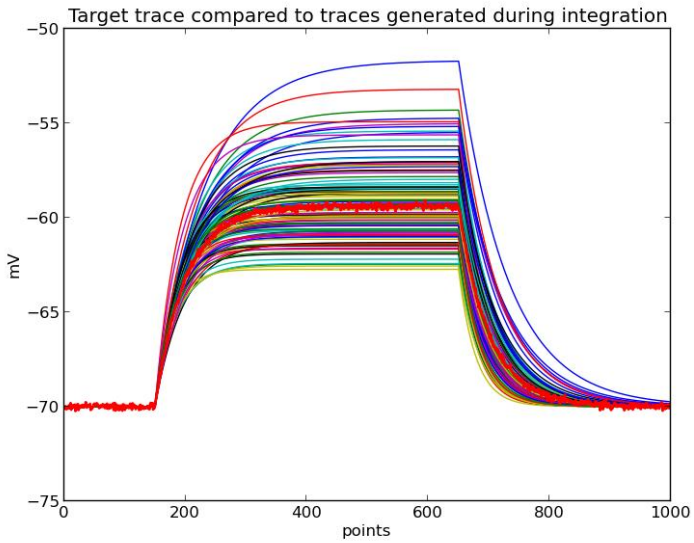
$$B = \int Pi(R|D) * P(D) dD$$

Az alkalmazott módszerek felvázolását követően rátérnék a megoldás bemutatására, elsőként kiemelve, hogy minden, modellezést érintő funkciót az Optimizer különböző elemeinek felhasználásával valósítottunk meg. Elsőként nem valós sejtek elemzésére tettünk kísérletet, egy egyszerű, egy kompartmentumos passzív modell felhasználásával próbáltuk megvizsgálni a létrehozott valószínűségi keret létjogosultságát.

A kezdeti lépések során nem kívántunk valódi mérési adatokat használni, ezért tetszőleges paraméter értékeket felhasználva (g_pas=0.0001, cm=1) egy megválasztott stimuláció protokol segítségével generáltunk egy adatsort, melyhez ezt követően adott amplitúdójú fehérzajt adtunk, így állítva el egy kvázi mérési adatot. Az így kapott adatsor 200 ms hosszú volt, és egy 0.1 pA amplitúdójú stimulációra adott választ tartalmazott, mely stimulus 100 ms-on keresztül tartott a 30. ms-tól.

A fenti integrál numerikus kiértékelésének, és ezáltal a poszterior eloszlás becslésének több lehetséges módja létezik, mi ezek közül az egyszerű, “brute force” megoldást választottuk. A módszer mellett (egyszerűségén túl) a tetszőlegesen meghatározható pontosság szólt, hiszen a vizsgált pontok számának növelésével a kapott eredmény pontossága is növekszik (lásd később).

Elsőként meghatároztuk a keresett paramétert, választásunk a membránkapacitást (cm) esett. Ezt a paramétert egy kiválasztott intervallumon ([0.001,2]) egyenletesen mintavételeztük (200 minta), majd minden így kapott értékhez generáltunk 10 g_pas értéket egy adott normális eloszlást felhasználva (átlag=0.0001, szórás=1e-10). Ezután az így kapott paraméterpárokat felhasználva szimulációkat futtatunk a fenti protokoll segítségével (15. ábra).

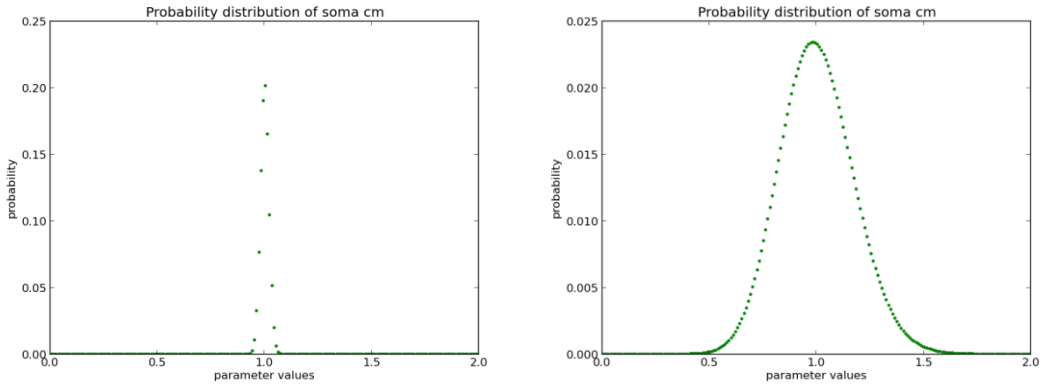


15. ábra Egy futtatás generálta adatsorok, összevetve a cél adattal (vastagon, pirossal jelölve)

Mivel a kísérleti adatsor létrehozásakor fehérzajt használtunk, így az egymást követő adatpont egymástól függetlenek, és pontonként összehasonlíthatóak. Ha ezt az összehasonlítást a négyzetes hibák összegének felhasználásával végezzük el, egy olyan értéket kapunk mely egyszerűen likelihood értékké alakítható.

Ezeket a likelihood értékeket minden cm paraméterre kiátlagoljuk, majd a fenti képlet (4. képlet) alapján beszorozzuk a priorral. Miután ezt minden cm értékre elvégeztük, a kapott értékeket összegeztük, és az egyes paraméterekhez számolt likelihood értékeket ezzel az összeggel leosztva megkapjuk a $Pi(D|R)$ értéket.

A következő ábrákon látható, hogy alacsony zaj esetén a módszerünk igen jó becslést ad a paraméter eloszlására, viszont az is látható, hogy a becslés pontosság az zaj amplitúdójának növekedésével jelentősen romlik (16. ábra).



16. ábra Alacsony zaj amplitúdó esetén a program sokkal pontosabban meg tudja határozni a keresett paramétert, míg erős zaj esetén az eredményül kapott eloszlás kiszélesedik.

Ezt követően megpróbálkoztunk egy szofisztikáltabb zajmodell használatával. A Ronald Fox, Ian Gatland (22) által kidolgozott algoritmus felhasználásával generáltunk színes zajt (lásd a következő kódrészletet).

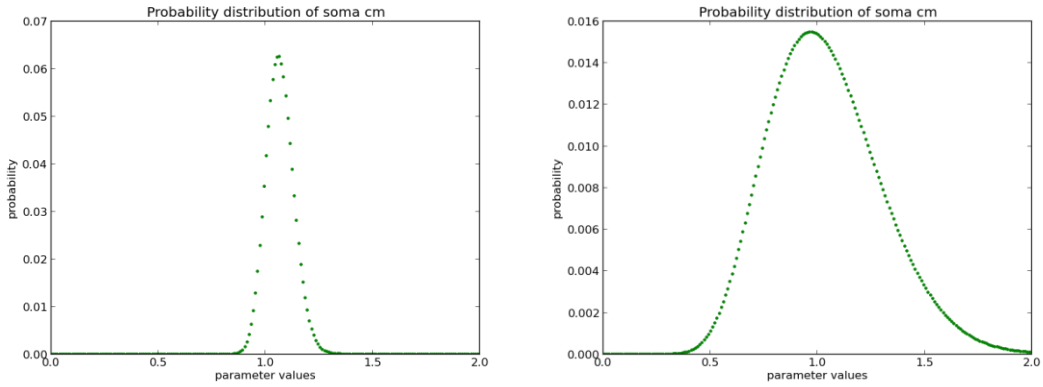
```
def coloredNoise(params,length):
    noise=[]
    D=params[0]
    lam=params[1]
    delta_t=1
    n,m = np.random.uniform(0.0,1.0,2)
    E=lambda x:exp(-x*delta_t)
    e_prev=sqrt(-2*D*lam*log(m))*cos(2*pi*n)
    noise.append(e_prev)
    for i in range(length-1):
        a,b = np.random.uniform(0.0,1.0,2)
        h=sqrt(-2*D*lam*(1-E(lam)**2)*log(a))*cos(2*pi*b)
        e_next=e_prev*E(lam)+h
        noise.append(e_next)
        e_prev=e_next
    return np.array(noise)
```

Az autókorrelációs függvény paramétereit, melyek meghatározták a színes zaj jellegét, mi magunk választottuk meg.

A zajmodellünk megváltoztatása egyben azt is jelentette, hogy a likelihood kalkulációjában a többváltozós normális eloszlás képletére kellett támaszkodnunk, szemben a korábbi egydimenziós esettel (7. képlet).

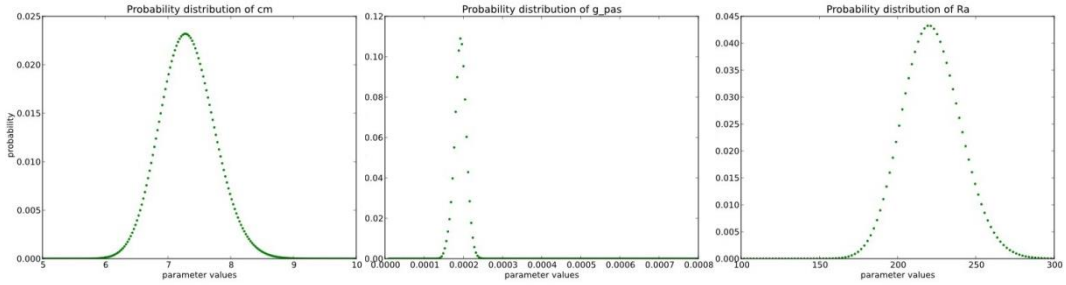
$$(2\pi)^{-\frac{k}{2}}|\boldsymbol{\Sigma}|^{-\frac{1}{2}}\, e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})},$$

Ahhoz, hogy ezt megtehessük, szükségünk volt a kovariancia mátrixra, melyet az autókorrelációs függvényből határoztunk meg, illetve szükség volt arra is, hogy az Optimizer MSE hibafüggvényét kibővítsük olyan módon, hogy a hagyományos működés helyett a kovariancia mátrix inverzével súlyozott értéket adjon vissza a képletnek (7. képlet) megfelelően.



17. ábra Látható, hogy a realisztikus zajmodell használata lényegesen nem módosítja a korábbi eredményeket (16. ábra)

Mivel úgy ítéltük, hogy a módszerünk mérsékelten zajos esetekben nagyon jó eredményt adhat (16-17. ábra), megkíséreltünk paraméter eloszlás-becsléseket végezni egy összetett, sokkompartmentumos CA1 piramissejt modelljének felhasználásával, mely modell megegyezett az Optimizer negyedik teszt eseteként bemutatott modellel. Magát a módszert változatlanul hagyva, a következő eloszlásokat és intervallumokat használva (6. táblázat) próbáltuk a modell passzív paramétereinek eloszlását megbecsülni alacsony amplitúdójú színes zaj használata mellett (18. ábra). Mivel ez a probléma jóval komplexebb az eddig bemutatottnál, várható volt, hogy az eredmények sem lesznek olyan pontosak, feltételezhetően a g_pas és Ra paraméterek eloszlásának helyes megbecsléséhez pontosabb integrálás szükséges.



18. ábra Az első ábrán látható, hogy a membránkapacitás eloszlását még elfogadható pontossággal képes megbecsülni a program, ugyanakkor a másik két parameter esetében sokkal rosszabbul teljesít.

	cm	Ra	g_pas
átlag	8	150	0.0005
szórás	2	30	0.0001
valódi érték	7.5	151	0.0003
minimum	5	100	1e-05
maximum	10	300	0.0008

6. táblázat A táblázatban megtalálhatók a felhasznált eloszlások paraméterei, az eloszlás becslés során használt intervallumok, valamint a céladat generálás során használt értékek

Osztályozás

Második lépésben azt vizsgáltuk meg, hogy az eddig használt valószínűségi keret alkalmas-e arra, hogy adott mérési eredmények alapján különböző paraméter eloszlásokat felismerjen a kiválasztott sejttípuson belül. Abból a feltevésből indultunk ki, hogy a sejttípuson belül léteznek különböző osztályok (C), ezek az osztályok a sejt, és így a modell szintjén megfeleltethetőek különböző paraméter eloszlásoknak (D). A valószínűségi modellünk egyéb elemeit a korábbiaknak megfelelően definiáltuk, a modell szintjén a különböző osztályok létezése azt jelenti, hogy a (8. képlet)

$$Pi(C|R)$$

valószínűségnek létezik egy jól definiált maximuma, azaz adott bemenetek (stimulusok) és kimenetek (mérések) esetén van olyan osztály, és ezáltal valamilyen paraméter eloszlás, melynek valószínűsége kiugró értéket mutat.

A fenti formula (8. képlet) Bayes tétel alkalmazásával átalakítható a következő alakba (9. képlet)

$$Pi(C|R) = \frac{Pi(R|C) * P(C)}{Pi(R)}$$

Ezt a formulát tovább tudjuk egyszerűsíteni, ha azt feltételezzük, hogy az osztályok egyforma valószínűséggel rendelkeznek, azaz P(C) minden osztályra nézve konstans, mivel így ez a tag elhagyható, ha az osztályok valószínűségét egymáshoz viszonyítjuk. A nevezőben található $Pi(R)$ a normalizációt szolgálja, így az egyenlet lényegi részét a $Pi(R|C)$ valószínűség adja. Ezt a valószínűséget egy kettős integrállal határozhatjuk meg a következő módon (10. képlet):

$$Pi(R|C) = \int P(\theta) \int P(D|C) * Pi(R|D,\theta) dD d\theta$$

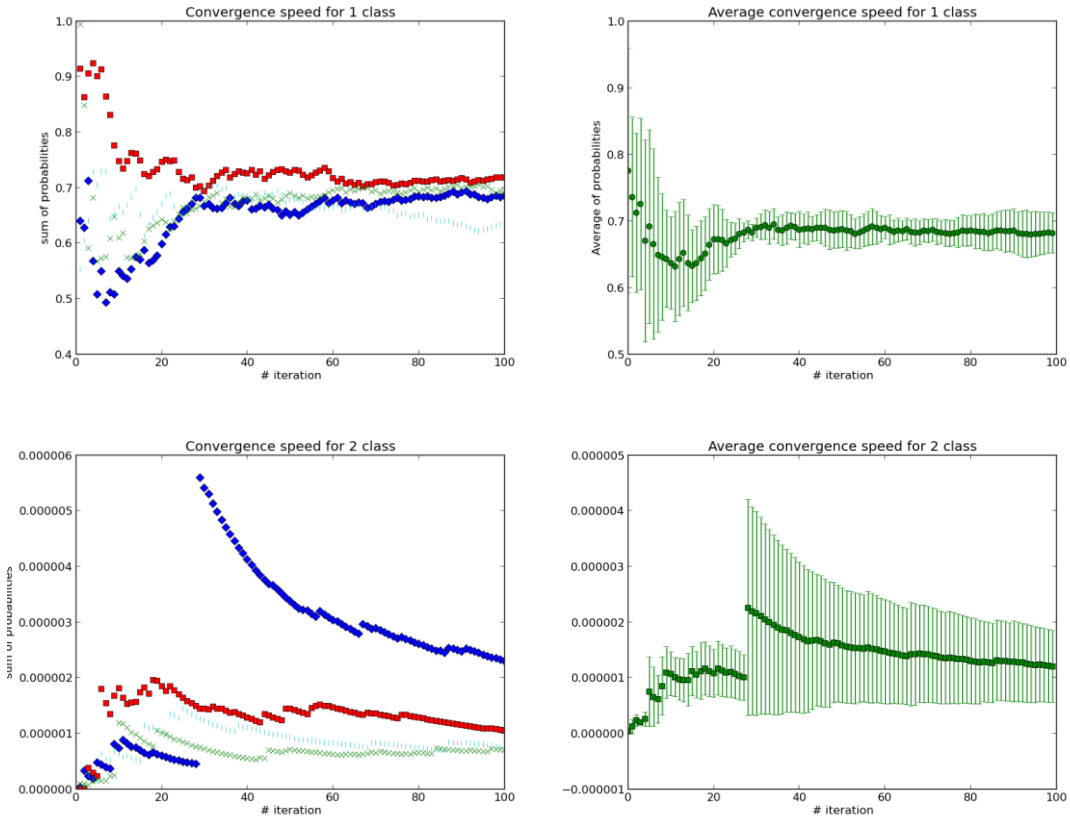
A fenti integrálok megoldására az eddigiekhez hasonlóan a brute force módszert alkalmaztuk. A probléma egyszerűsítése végett két osztályt határoztunk meg, paraméterként pedig a membránkapacitást (cm) választottuk meg. A membránkapacitás a két osztályban eltérő átlaggal (0.9 illetve 2) de azonos szórással (0.1) rendelkező normális eloszlást követ, a modell másik passzív paraméterét (g_pas) szintén egy normális eloszlással írtuk le (átlag=0.0001, szórás=1e-10), mely mindkét osztály estében megegyezett. A probléma tovább egyszerűsítése végett valós adatok helyett ismét mesterségesen készítettünk mérési adatokat, ahogy azt korábban vázoltuk. Ez a gyakorlatban azt jelentette, hogy választottunk egy cm-g_pas értékpárt (1; 0.0001), olyan módon, hogy a cm értéke közel essen az egyik osztály átlagához, majd a kiválasztott értékpárral generáltunk egy adatsort, melyhez ezt követően adott amplitúdójú fehérzajt adtunk, így állítva elő

a cél adatsort.

Ezt követően a $Pi(R|C)$ meghatározását végeztük el brute force módszerrel. A módszer lényegi lépései a következők voltak:

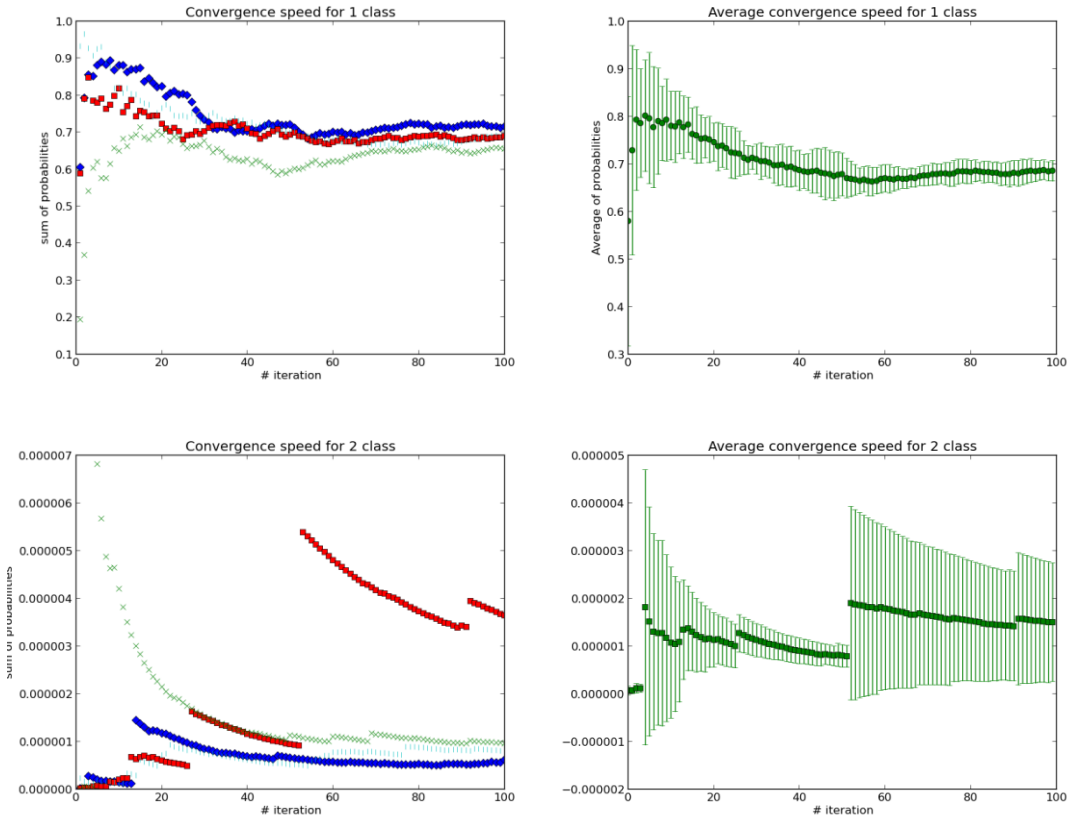
- D és théta (cm és g_pas) paramétereket generáltunk először az egyik, majd a másik osztály számára a megfelelő eloszlások alapján
- A paramétereket felhasználva szimulációkat futtatunk és a korábban leírt módon meghatároztuk a likelihood értékeket, majd pedig a valószínűségeket.

Természetesen a módszer pontossága nagyban függ a megvizsgált értékek számától, minél több paramétert értékelünk ki, annál közelebb kerülünk az integrál analitikus megoldásához. Értelemszerűen a több vizsgált paraméter több szimuláció futtatását is jelenti, ezért ez nagy komplexitású modellek esetében nagyon lelassíthatja a folyamatot.



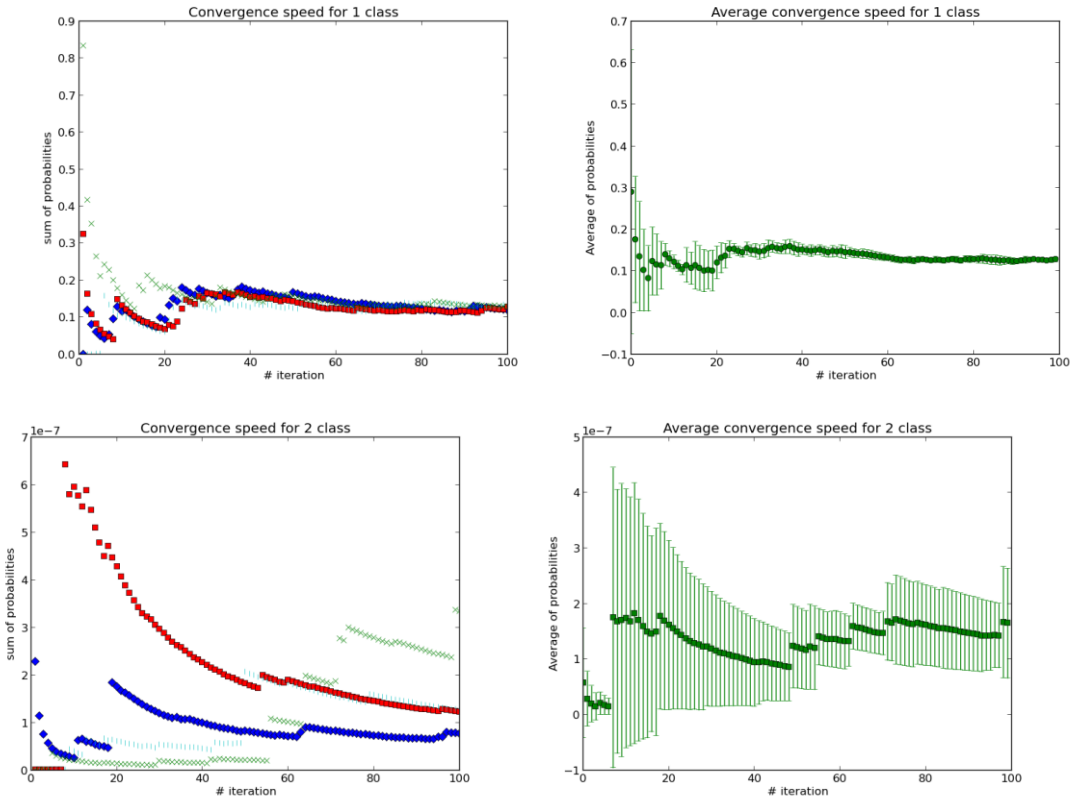
19. ábra Alacsony amplitúdójú fehérzaj felhasználásával készült adatok esetén a program könnyedén megoldja a problémát. A bal oldalt 5 futtatás konvergenciája látható, míg a jobb oldalt ezek átlaga és szórása található. Mindkét ábráról leolvasható, hogy az ilyen alacsony amplitúdójú zajjal terhelt esetben a brute force módszer hamar eléri a kívánt pontosságot

A módszer egyszerűségéből adódóan azonban könnyen megvizsgálhatjuk a konvergencia mértékét, és ezáltal a kapott eredmény pontosságát (19. ábra). A konvergencia mértéke természetesen jelentősen függeni fog a zaj amplitúdójától, de a felvázolt módszer zajjal erősen terhelt adatok esetében is működőképes maradt az egyszerű modell esetében (20. ábra).



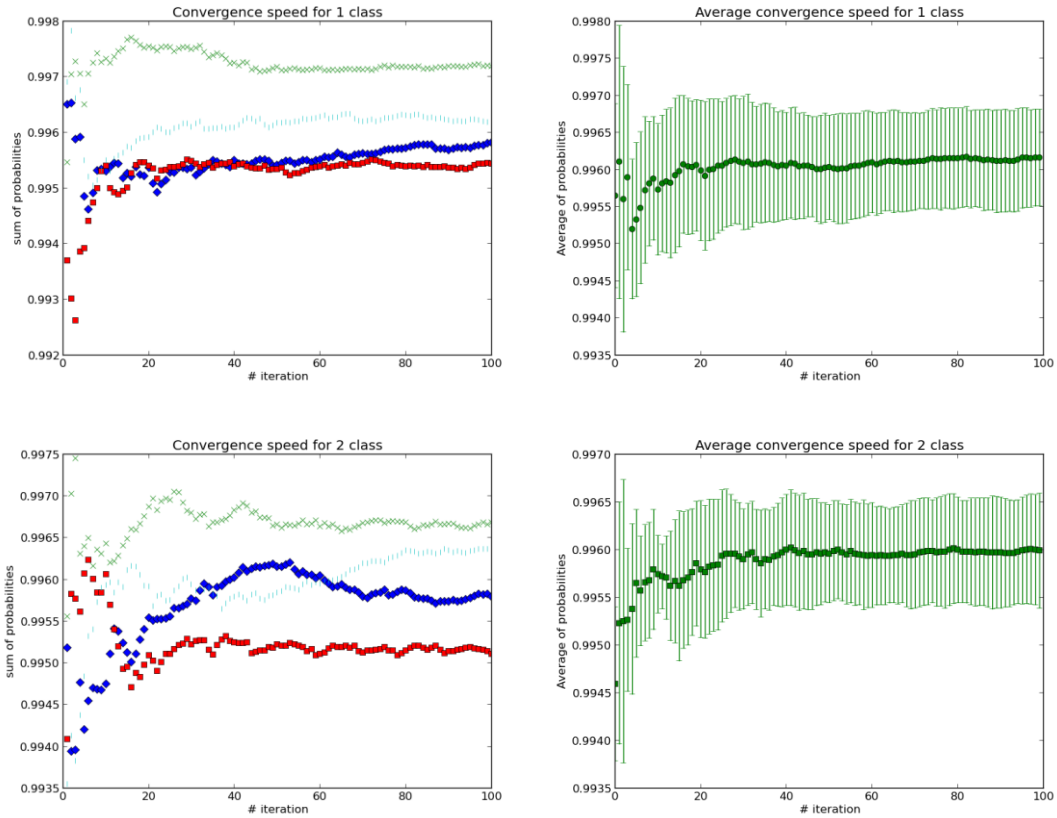
20. ábra Látható, hogy a konvergencia sebessége nem romlott jelentősen magas amplitúdójú zaj használatával sem (a 2. osztály esetében a gyengébb konvergencia azzal magyarázható, hogy a céladatunk az 1. osztályból származik)

Ha visszatérünk egy pillanatra a paraméterek eloszlásának meghatározásához, láthatjuk, hogy a g_pas paraméter esetében nagyon kis szórást engedélyeztünk csak (1e-10), ezzel lényegében egydimenzióssá téve a problémát. A lenti ábrákon látható, hogy jelentősen nagyobb (2e-5) szórás esetén sem romlik lényegesen a program teljesítménye (21. ábra).



21. ábra Látható, hogy dimenzió növelésétől függetlenül a program még mindig képes a két osztályt megkülönböztetni egymástól

A program a realisztikusabb zajmodell használata mellett már több iterációt igényelt, mint a korábbi esetekben, de még mindig képes volt megkülönböztetni a két osztályt egymástól (22. ábra), még magas zaj amplitúdó mellett is. Jól látható ugyanakkor, hogy a zaj amplitúdójának növekedésével a két osztály közti különbség fokozatosan csökken, így tehát könnyen elképzelhető olyan eset, amikor a zaj összemossa a két osztályt.



22. ábra Látható, hogy magas amplitúdójú színes zaj használata mellett a két osztály kezd egyre inkább átfedésbe kerülni, a program már nem tudja nagy biztonsággal elkülöníteni őket

Összefoglalás

Összegezve az eddigieket, elmondhatjuk, hogy az általunk kidolgozott valószínűségi modell alkalmas lehet elektrofiziológiai mérések tervezésének támogatására, illetve a mérések során kapott adatok elemzésére és értelmezésére. A sikeres tesztek nem csak elméletünk megvalósíthatóságát igazolják, hanem azt is, hogy az Optimizer moduláris felépítésének köszönhetően remekül alkalmas paraméterillesztéshez csak lazán kötődő problémák megoldására is.

Az eddigi tesztekre alapozva szeretnénk a kidolgozott módszert valódi mérési adatokon is kipróbálni, azaz szeretnénk megbecsülni egy kiválasztott paraméter eloszlását konkrét mérési adatok alapján. Ehhez azonban elsőként szükségünk van a zaj modell pontosítására. Valódi mérési eredmények esetében ugyanis nem használhatnánk tetszőleges autókorrelációs függvényt, azt a mérési adatokból kellene meghatároznunk. Már tettünk kísérletet ennek megvalósítására, azonban az autókorrelációban váratlanul megjelenő periodikus komponens miatt a probléma további vizsgálatokat igényel.

A valós kísérleti eredmények felhasználását szeretnénk komplex modellek használatával ötvözni, ez viszont azt is jelenti, hogy az eddigi alacsony dimenziós problémák helyett sokdimenziós eseteket kell kezelnünk. Mivel erre az eddig használt brute force megoldás csak korlátozottan alkalmas, mindenképpen szeretnénk megvizsgálni a Laplace közelítés felhasználhatóságát. Mivel az integrálandó görbe a marginális likelihood függvények alakjából következtetve közel gauss görbe alakú lehet, a Laplace módszer használható lenne, a lokális optimum megtalálása pedig az Optimizer felhasználásával nem jelent problémát. A közelítő módszer alkalmazása azért lehet előnyös, mert a probléma dimenziójának növekedésével a brute force módszer konvergenciájának sebessége drasztikusan lecsökken, míg a közelítés elvégzésének gyorsaságát csak az optimum megtalálásának sebessége határozza meg.

A biztató eredményekre való tekintettel a továbbiakban szeretnénk a már használt komplex modellben részletesebben is megvizsgálni a g_pas paramétert viselkedését, különös tekintettel a paraméter térbeli eloszlására. Terveink között szerepel további mérési protokollok kipróbálása is, mivel ezek összevetéséből olyan hasznos információkat kaphatunk, melyek közvetlenül segíthetnék elektrofiziológiai mérések tervezését.

Távlati terveink között szerepel egy olyan rendszer kifejlesztése, mely képes egy adott mérési protokoll információ tartalmára vonatkozó becslést adni valós mérés elvégzése nélkül.

A célkitűzések teljesítése

Munkánk során sikeresen létrehoztunk egy olyan keretrendszert mely hatékonyan képes idegsejt modellek optimalizációjára. A szoftver tesztelése során részletesen áttekintettük a NEURON modellező szoftver működését, illetve megismerkedtünk más, hasonló funkciójú szoftverrel is.

Munkánk második felében sikeresen létrehoztunk egy valószínűségi modellt, melyet részletes teszteknek vettetünk alá. A kapott eredmények azt mutatják, hogy a létrehozott modell alkalmas arra, hogy alapvető információkat adjon a vizsgált sejtről (paraméter eloszlások, jellemző esetek szétválasztása). Ugyanakkor a modell további tesztelést igényel, elsősorban valódi biológiai adatok felhasználásával szeretnénk az általunk kidolgozott módszer lehetőségeit és korlátait vizsgálni.

Köszönetnyilvánítás

Szeretném megköszönni konzulensemnek, Káli Szabolcsnak a rengeteg türelmet és szakértelmet, valamint a GENESIS-ben nyújtott segítséget. Külön köszönet illeti Szoboszlay Miklóst és Nusser Zoltánt a rendelkezésemre bocsájtott modellekért és mérési adatokért.

Irodalomjegyzék

[1] Friedrich P, Vella M, Gulyás AI, Freund TF and Káli S (2014) A flexible, interactive software tool for fitting the parameters of neuronal models. Front. Neuroinform. 8:63. doi: 10.3389/fninf.2014.00063

[2] Hay, E., Hill, S., Schürmann, F., Markram, H., and Segev, I. (2011). Models of neocortical layer 5b pyramidal cells capturing a wide range of dendritic and perisomatic active properties. PLoS Comput. Biol. 7, e1002107. doi:10.1371/journal.pcbi.1002107

[3] Gómez González JF, Mel BW and Poirazi P (2011) Distinguishing linear vs. non-linear integration in CA1 radial oblique dendrites: it's about time. Front. Comput. Neurosci. 5:44. doi: 10.3389/fncom.2011.00044

[4] Gerstner, W., and Naud, R. (2009). Neuroscience. How good are neuron models? Science 326, 379–380. doi:10.1126/science.1181936

[5] Naud, R., Marcille, N., Clopath, C., and Gerstner, W. (2008). Firing patterns in the adaptive exponential integrate-and-fire model. Biol. Cybern. 99, 335–347. doi:10.1007/s00422-008-0264-7

[6] Rossant, C., Goodman, D. F. M., Platkiewicz, J., and Brette, R. (2010). Automatic fitting of spiking neuron models to electrophysiological recordings. Front. Neuroinformatics 4, 2. doi:10.3389/neuro.11.002.2010

[7] Vanier, M. C., and Bower, J. M. (1999). A comparative survey of automated parameter-search methods for compartmental neural models. J. Comput. Neurosci. 7, 149–171

[8] Druckmann, S., Banitt, Y., Gidon, A., Schürmann, F., Markram, H., and Segev, I. (2007). A novel multiple objective optimization framework for constraining conductance-based neuron models by experimental data. Front. Neurosci. 1, 7–18. doi:10.3389/neuro.01.1.1.001.2007

[9] Bahl, A., Stemmler, M. B., Herz, A. V. M., and Roth, A. (2012). Automated optimization of a reduced layer 5 pyramidal cell model based on experimental data. J. Neurosci. Methods 210, 22–34. doi:10.1016/j.jneumeth.2012.04.006

[10] Druckmann, S., Berger, T. K., Hill, S., Schürmann, F., Markram, H., and Segev, I. (2008). Evaluating automated parameter constraining procedures of neuron models by experimental and surrogate data. Biol. Cybern. 99, 371–379. doi:10.1007/s00422-008-0269-2

[11] Káli, S., and Freund, T. F. (2005). Distinct properties of two major excitatory inputs to hippocampal pyramidal cells: a computational study. Eur. J. Neurosci. 22, 2027–2048. doi:10.1111/j.1460- 9568.2005.04406

[12] Eichner, H., and Borst, A. (2011). Hands-on parameter search for neural simulations by a MIDIcontroller. PloS One 6, e27013. doi:10.1371/journal.pone.0027013

[13] Rossant, C., Goodman, D. F. M., Fontaine, B., Platkiewicz, J., Magnusson, A. K., and Brette, R. (2011). Fitting neuron models to spike trains. Front. Neurosci. 5, 9. doi:10.3389/fnins.2011.00009

[14] Van Geit, W., Achard, P., and De Schutter, E. (2007). Neurofitter: a parameter tuning package for a wide range of electrophysiological neuron models. Front. Neuroinformatics 1, 1. doi:10.3389/neuro.11.001.2007

[15] Van Geit, W., De Schutter, E., and Achard, P. (2008). Automated neuron model optimization

techniques: a review. Biol. Cybern. 99, 241–251. doi:10.1007/s00422-008-0257-6

[16] Carnevale, N. T., and Hines, M. L. (2006). The NEURON book. Cambridge: Cambridge University Press

[17] Bower, J. M., and Beeman, D. (1998). The book of GENESIS (2nd ed.): exploring realistic neural models with the GEneral NEural SImulation System. Springer-Verlag New York, Inc.

[18] Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. (1995). A Limited Memory Algorithm for Bound

Constrained Optimization. SIAM J. Sci. Comput. 16, 1190–1208. doi:10.1137/0916069

[19] Wales, D J, and Doye J P K (1997). Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms. J. Phys. Chem. A, 1997, 101 (28), pp 5111–5116 DOI: 10.1021/jp970984n

[20] Nelder, J. A., and Mead, R. (1965). A Simplex Method for Function Minimization. Comput. J. 7, 308– 313. doi:10.1093/comjnl/7.4.308

[21] Brent, R. P. (2002). Algorithms for minimization without derivatives. Mineola, N.Y.: Dover

Publications

[22] Davison, A. P., Brüderle, D., Eppler, J., Kremkow, J., Muller, E., Pecevski, D., Perrinet, L., and Yger, P. (2008). PyNN: A Common Interface for Neuronal Network Simulators. Front.

Neuroinformatics 2, 11. doi:10.3389/neuro.11.011.2008.

[23] Garcia, S., Guarino, D., Jaillet, F., Jennings, T., Pröpper, R., Rautenberg, P. L., Rodgers, C. C., Sobolev, A., Wachtler, T., Yger, P., et al. (2014). Neo: an object model for handling electrophysiology data in multiple formats. Front. Neuroinformatics 8, 10. doi:10.3389/fninf.2014.00010.

[24] Ronald F. Fox, Ian R. Gatland, (1988). Fast, accurate algorithm for numerical simulation of exponentially correlated colored noise. Physical Review A - PHYS REV A , vol. 38, no. 11, pp. 5938-5940, 1988 DOI: 10.1103/PhysRevA.38.5938