Міністерство охорони здоров'я України
Державний вищий навчальний заклад
«Тернопільський державний медичний університет
імені І.Я. Горбачевського»

Тернопільський національний технічний університет імені Івана Пулюя

МОН України

Кваліфікаційна наукова
праця на правах рукопису

Майхрук Зоряна Василівна

УДК 519.876.2: 612.014.3

ДИСЕРТАЦІЯ
МОДЕЛІ ТА МЕТОДИ НЕЛІНІЙНОГО АНАЛІЗУ ЕЛЕКТРИЧНОЇ АКТИВНОСТІ СИСТЕМИ ХОДЖКІНА-ХАКСЛІ

01.05.02 – математичне моделювання та обчислювальні методи

технічні науки

Подається на здобуття наукового ступеня кандидата технічних наук

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело _____З. В. Майхрук

Науковий керівник Марценюк Василь Петрович

доктор технічних наук, професор

Тернопіль – 2017

ЗМІСТ

АНОТАЦІЯ.....5
Перелік умовних позначень.....10
ВСТУП.....11
РОЗДІЛ 1 Математичне моделювання електричної збудливості біологічних клітин.....19
1.1 Математична модель електричної активності клітин Ходжкіна-Хакслі і її застосування в електрофізіології.....19
1.1.1 Історичні передумови появи рівнянь Ходжкіна-Хакслі дляопису мембранного потенціалу.....23
1.1.2 Напруга і миттєва залежність провідності.....24
1.2 Дослідження електричної активності клітин на основі двовимірних моделей.....31
1.2.1 Швидка фазова площина.....32
1.2.2 Якісний аналіз на основі двовимірної моделі.....34
1.3 Моделювання електричної збудливості для різних видів клітин.....38
1.3.1 Моделі збудливості серцевих клітин.....38
1.3.2 Волокна Пуркіньє. Феноменологічний підхід.....39
1.3.3 Синоатріальний (SA) вузол.....41
1.3.4 Шлуночкові клітини.....43
Висновки до першого розділу.....45
РОЗДІЛ 2 Використання методів обчислювальної математики стосовно вирішення проблем дослідження системи електричної активності нервових клітин…….....47
2.1 Модифікація й спеціалізація існуючого методу обчислення експонент Ляпунова з метою дослідження стійкості з урахованням особливостей математичної моделі Ходжкіна-Хакслі. Створення програмного засобу комп'ютерної реалізації методу.....47

2.2 Розробка методу оптимізації математичної моделі Ходжкіна-Хакслі для задачі оптимального керування біфуркацією при зміні прикладеного струму. Отримання необхідних умов оптимальності 53
2.3 Модифікація прямого чисельного методу для побудови оптимального керування біфуркацією в моделі Ходжкіна-Хакслі з метою створення ефективного програмного засобу комп'ютерної реалізації 60
2.3.1 Чисельне дослідження виникнення біфуркацій в моделі Ходжкіна-Хакслі при біфуркаційному параметрі I_{app} 63
2.3.2 Прямий метод чисельного розв'язування задачі оптимального керування 69
Висновки до другого розділу 75
РОЗДІЛ 3 Розроблення мультиваріативних методів дослідження моделі Ходжкіна-Хакслі у різних режимах їх функціонування, інтерпретація результатів моделювання. 77
3.1 Розроблення мультиваріативного методу якісного аналізу дослідження нелінійних систем. 77
3.2 Спеціалізація алгоритмів технології data mining для класифікації траєкторій динамічних систем Ходжкіна-Хаслі з урахованням особливостей реальних технічних задач. 83
3.3 Модифікація методу класифікації типів збудливості нервових клітин з побудовою класифікаційних правил та дерев рішень. 87
Висновки до третього розділу 94
РОЗДІЛ 4 Побудова програмної системи моделювання електрофізіологічних досліджень 95
4.1 Організація структури Web – інтегрованої програмної системи електрофізіологічних досліджень на основі моделі Ходжкіна-Хакслі. 95
4.1.1 Принципи побудови програмного середовища електрофізіологічних досліджень. 95
4.1.2 Концептуальна модель ПСЕФД. Структуризація об'єкта та системи керування.. 97

4.1.3 Структура інформаційної моделі інтегрованого середовища програмного середовища електрофізіологічних досліджень. 99
4.2 Особливості комп'ютерного моделювання у фізіологічних дослідженнях 102
4.2.1 Аналіз програмного забезпечення, що використовується у електрофізіологічних дослідженнях. 106
4.2.2 Програмне середовище реалізоване у вигляді пакету Java-класів 107
4.2.2.1 Програмна реалізація мультиваріативного методу для системи Ходжкіна-Хакслі. 113
4.2.3 Особливості програмної реалізації моделі Ходжкіна-Хакслі. 116
4.2.3.1 SQL-реалізація розрахунку інформаційних показників приросту інформації 121
4.2.3.2 SQL-реалізація розрахунку відношення інформаційних показників приростів інформації. 122
4.3 Оцінювання обчислювальної складності з метою забезпечення якісного аналізу моделі Ходжкіна-Хакслі 123
4.3.1 Проблема обчислювальної складності алгоритму індукції дерева рішень. Оцінка складності виконання алгоритму. 123
4.3.2 Оцінка складності виконання алгоритму на основі методу послідовного покриття. 126
4.3.2.1 Оптимізація побудови копій класифікаційних правил. 128
Висновки до четвертого розділу 131
ВИСНОВКИ 132
СПИСКИ ВИКОРИСТАНИХ ДЖЕРЕЛ 134
ДОДАТКИ 149
Додатки А Акти впровадження 149
Додатки Б Свідоцтва авторських прав на витвір 153
Додатки В Таблиці 155
Додатки Г Лістинги 158

АНОТАЦІЯ

Майхрук З.В. **Моделі та методи нелінійного аналізу електричної активності системи Ходжкіна-Хакслі. – Кваліфікаційна наукова праця на правах рукопису.**

Дисертація на здобуття наукового ступеня кандидата технічних наук за спеціальністю 01.05.02 «Математичне моделювання та обчислювальні методи» – Державний вищий навчальний заклад «Тернопільський державний медичний університет імені І. Я. Горбачевського» – Тернопільський національний технічний університет імені Івана Пулюя МОН України, Тернопіль, 2017.

Дисертацію присвячено математичному моделюванню електричної активності клітини на основі системи збудливості нейрона Ходжкіна-Хакслі.

У роботі отримала подальший розвиток теорія математичного моделювання електрофізіологічної активності клітин за рахунок отримання для системи Ходжкіна-Хакслі методів дослідження стійкості, оптимізації, керування та нелінійного аналізу. Розроблено математичні методи системного аналізу та прийняття рішень на основі системи Ходжкіна-Хакслі для якісного аналізу електричної активності нервових клітин.

Існування в системі Ходжкіна-Хакслі складної (навіть при певних наборах параметрів хаотичної) поведінки вказує на необхідність розгляду керування в моделі – в першу чергу за рахунок зовнішнього прикладеного струму.

Тому в роботі розроблено математичні методи керування біфуркацією у системі Ходжкіна-Хакслі, що виникає при зміні прикладеного струму, які забезпечують створення ефективних програмних засобів комп'ютерної реалізації. Метод побудови оптимального керування на основі принципу максимуму опирається на необхідні умови оптимальності, представлені у вигляді крайової задачі. Для чисельного знаходження оптимального керування

в моделі Ходжкіна-Хакслі адаптовано прямий чисельний метод, що зводить нескінченновимірну задачу оптимального керування до скінченновимірної задачі нелінійного програмування.

Вперше запропоновано підхід дослідження системи Ходжкіна-Хакслі у різних режимах їх функціонування на основі мультиваріативного методу, що включає алгоритм послідовного покриття або ж алгоритм індукції дерева рішень та забезпечує біологічну інтерпретацію результатів моделювання. Метод полягає у генеруванні випадковим чином початкових значень та значень швидкісних параметрів, які б належали практично обґрунтованій області. Для кожного з наборів таких параметрів здійснюється інтегрування системи з отриманням відповідних траєкторій. До отриманих результатів далі застосовується алгоритм технології data mining (індукції дерева рішень, метод послідовного покриття та ін.) з метою отримання певних структур знань для прийняття рішень.

У цілому метод поєднує у собі підхід Монте-Карло для формування навчальних наборів та класифікаційні алгоритми data mining: метод послідовного покриття з генерацією класифікаційних правил та метод індукції дерева рішень.

Метод індукції дерева рішень, будучи складнішим в реалізації, допускає наочну візуалізацію та апріорні значення ймовірностей для типу збудливості клітини виходячи із співвідношень між початковими значеннями та швидкісними параметрами у моделі Ходжкіна-Хакслі.

Запропонований мультиваріативний метод якісного аналізу моделей електрофізіологічних процесів є підходом, що дозволяє вирішувати задачі класифікації збудливості клітин, які не можуть бути вирішені іншими традиційними методами, наприклад теорії стійкості, або ж граничних циклів.

Отримав подальший розвиток обчислювальний метод розрахунку експонент Ляпунова, який було модифіковано та спеціалізовано з метою підвищення його ефективності з урахуванням особливостей нелінійної динаміки в системі Ходжкіна-Хакслі.

Методи системного аналізу, розроблені в даній роботі, доведено до комп'ютерної реалізації. Розроблено Web-інтегроване програмне середовище електрофізіологічних досліджень, представлене у вигляді пакету програмних класів.

Ключові слова: система Ходжкіна-Хакслі, електрична активність клітини, оптимальне керування, принцип максимуму, якісний аналіз, класифікаційні правила, дерево рішень.

SUMMARY

Mayhruk Z.V. Models and methods of nonlinear analysis of electrical activity of the Hodgkin-Huxley. – Qualifying scientific work. The manuscript.

The thesis for the degree of candidate of technical sciences, specialty 01.05.02 «Mathematical modeling and computational methods» – State Higher Educational Institution «Ternopil State Medical University» – Ternopol National Technical University Ivan Pul'uj MES Ukraine, Ternopol, 2017.

The thesis is devoted to mathematical modeling of the electrical activity of the cells on the basis of neuron excitability Hodgkin-Huxley.

Further developed the theory of mathematical modeling of electrophysiological activity of cells by receiving system Hodgkin-Huxley methods of stability, optimization, management, and nonlinear analysis. The mathematical methods of system analysis and decision-making on the basis of Hodgkin-Huxley for qualitative analysis of the electrical activity of nerve cells.

The existence of a system of Hodgkin-Huxley complex (even with a certain set of parameters chaotic) behavior points to the need to consider the management model – primarily due to external applied current.

Mathematical methods for managing bifurcation in the system of Hodgkin-Huxley that occurs when changing the applied current, which ensure the establishment of effective implementation of computer software. Method of optimal

control construction using maximum principle is based on necessary conditions of optimality presented as a boundary value problem. For the purpose of numerical search of optimal control in Hodgkin-Huxley model there is adapted direct numerical method reducing the infinite dimensional optimal control problem to finite dimensional problem of nonlinear programming.

The first time the research system approach Hodgkin-Huxley in different modes of operation based multivariate method comprising sequential algorithm coverage or decision tree induction algorithm and provides the biological interpretation of simulation results. The method consists in generating random initial values and the values of speed parameters to be included practically reasonable area. For each of the sets of parameters are integrating systems to obtain relevant trajectories. With the results then applies the algorithm technology data mining (decision tree induction, method of sequential covering and so on.) in order to obtain certain structures of knowledge for decision making.

In general, the method combines the Monte Carlo approach to create training sets and classification algorithms for data mining: sequential covering method to generate classification rules and decision tree induction method.

The advantages of classification rules which are based on the five-step algorithm are that they meet the natural reflection of knowledge and thinking of human and that they are more expressive. In addition, the algorithm of sequential covering is easier to implement and debugging compared with recursive decision trees algorithms, and its computational complexity is simpler compared to finite automata.

The method of induction of decision trees, being difficult to implement, and allows a clear visualization priori probability values for type excitability of cells based on the relationship between initial values and speed parameters in the Hodgkin-Huxley model.

The proposed method multyvariatyvnyy qualitative analysis of patterns of electrophysiological processes is an approach that can solve the problem of

classification excitability of cells that can not be solved by other conventional methods, such as stability theory or limit cycles.

There is developed computational method for calculating Lyapunov exponents, which is modified and specialized to improve its efficiency allowing investigation for the nonlinear dynamics of Hodgkin-Huxley system.

System analysis methods developed in this study are appropriate for computer implementation. Web-integrated software environment of electrophysiological research is developed which is implemented as package of software classes.

Keywords: Hodgkin-Huxley system, the electrical activity of cells, optimal control, maximum principle, qualitative analysis, classification rules, decision tree.

Перелік умовних позначень

TTX – тетродотоксин

Tea – тетраетіламмоній

SA – синоатріальний вузол

AB – атріовентрикулярний вузол

YNI – модель Янагіхари

MNT – модель начесть авторів McAllister, Noble та Tsien

БР – рівняння Бейлера-Ройтера

LR – рівняння Луо-Руді

ПСЕФД – програмне середовище електрофізіологічних досліджень

ІКС – клініко-лабораторні дослідження, консультативна комп'ютерна діагностика, постійний моніторинг пацієнта

АСУ – апаратне системне управління

ІССЕФД – інтегровне середовище системних електрофізіологічних досліджень

SDT (Scientific Directions Topology) – проекція топологія наукових напрямків медичних досліджень, яких дотримується ПСЕФД

SDS (Software and Data Structures) – проекція структури програмного забезпечення і Ресурсів даних, які використовуються в системі

UIP (User Information Profiles) – проекція інформаційні профілі користувачів, які взаємодіють з ПСЕФД як безпосередньо в її структурі, так і за її межами (віддалені користувачі)

NRP – проекція опис нових результатів і задач, які виникають при проведенні біомедичних досліджень

NRU (New Results Upgrade) – проекція шляхи впровадження нових результатів та розв'язування нових задач для вдосконалення ПСЕФД

ІССЕФД – позначимо як IMSRE (Integrated Medical System Researches Environment

ВСТУП

Актуальність теми. При дослідженні біологічних процесів і явищ ширше застосування отримують методи математичного моделювання. При цьому під поняттям модель мають на увазі двокомпонентну структуру – перша складова містить загальний опис явища на основі поточних знань, що також називається евристичною моделлю або біологічними припущеннями, в той час друга складова – математична структура, в якій намагаються відобразити попередньо розроблену евристичну модель. Класичним і найчастіше вживаним математичним формалізмом є диференціальні та різницеві рівняння, оскільки перші моделі в біології будувалися такими ж методами, як і моделі у фізиці. При використанні такого математичного формалізму слід визначитися, що є змінними, а що параметрами моделі. Параметри визначаються на основі експериментів, вимірювань чи спостережень, тоді як змінні є невідомими, що розраховуються (аналізуються) на основі моделі. Розроблена математична модель є предметом аналітичних і комп'ютерних досліджень, завдяки яким вивчаються властивості розв'язків. На цьому етапі суттєвим є вдалий вибір об'єкту моделювання, виходячи з медико-біологічної проблеми, яку планується вирішити.

Важливий внесок у розвиток теорії та методів математичного моделювання та оптимізації, загальної теорії керування, а також в розробку відповідних програмних засобів в Україні внесли Б.М. Бублик [1], А.П. Власюк [2], Ф.Г. Гаращенко [3 – 9], Ю.М. Єрмольєв [10], О.К. Закусило [11], М.З. Згуровський [12, 13], В.М. Кунцевич [14 – 20], С.І. Ляшко [21 – 23], О.Г. Наконечний [24, 25], Ю.М. Онопчук [26, 27], І.В. Сергієнко [28], А.А. Чикрій [17, 29, 30] та ін.

Результати застосування методів математичного моделювання та теорії автоматичного керування в медицині та біології відображені в роботах М.М. Амосова [31 – 33], Ю.Г. Антомонова [34 – 38], Н. Бейлі [39], Р. Ледлі [40], І.М. Ляшенка [41], В.П. Марценюка [42 – 44], Г.І. Марчука [45, 46], М. Мекі [47], О.П. Мінцера [48], Дж.М. Муррея [49], О.Г. Наконечного [50],

Ю.М. Онопчука [51], Ю.І. Петуніна [52, 53], М.Б. Славіна [54], Дж. Снейда [55], В.О. Яценка [56] та інших.

Проблеми математичного моделювання біосигналів вивчалися в роботах Я.П. Драгана [57 – 62], С.А. Лупенка [63 – 65], М.П. Приймака [66 – 70], Л.М. Щербака [71 – 75], Б.І. Яворського [76] та інших.

Вивчення медико-біологічних процесів вимагає розробки і дослідження їх математичних моделей на клітинному рівні. Жива клітина є відкритою системою. Умовою життя є неперервний обмін енергією і речовиною між клітиною і її оточенням. Клітинні мембрани, що забезпечують автономність відокремлених ними компартментів, повинні володіти механізмами селективного транспорту речовин, необхідних для життя клітин. Особливо важливим з точки зору біології клітини є транспортування іонів через клітинну мембрану. Транспортні властивості мембран забезпечують внутрішній іонний склад клітин, що відрізняється від іонного складу їх оточення.

Особливе зацікавлення викликало питання транспортування іонів через мембрани нервових клітин, пов’язане з процесами електричної активності та передачі інформації. Перші методики реєстрації електричної активності нервових клітин з’явилися ще в XIX столітті. Ю. Бернштейн виконав реєстрацію потенціалу дії нейрона в 1868 році. Одночасно розроблялася концепція стимуляції нервової клітини. Першу вдалу спробу стимуляції здійснив і описав Р. Герард в 1933 році. Наступним кроком стало опис явища виникнення потенціалу дії в нервових клітинах кальмара на основі фізичних явищ А. Ходжкіном та А. Хакслі, удостоєних за це відкриття у 1969 році Нобелевської премії в галузі медицини. Це була перша фізична модель нервової клітини, що відображає роботу іонних каналів, на якій ґрунтується більшість сучасних моделей.

Незважаючи на значні здобутки у сфері математичного моделювання медико-біологічних процесів, при використанні підходу до вивчення електричної активності клітин виникає ряд проблем. У переважній більшості дослідження стосуються побудови моделей іонної провідності клітинних

мембран на основі формалізму Ходжкіна-Хакслі. Основні практичні результати стосуються встановлення наближеного аналітичного вигляду швидкісних параметрів в нелінійних диференціальних рівняннях статистичними методами для різних груп клітин. У той же час отримання результатів щодо якісних характеристик клітин (наприклад класифікація типів збудливості) в аналітичному вигляді (у вигляді умов, що включають чисельні параметри) на основі системи типу Ходжкіна-Хакслі не видається можливим в силу складних нелінійностей в рівняннях. Одним із можливих підходів, що використовується, є зведення до двовимірної моделі Фіцьхью-Нагумо, яка ґрунтується на певних обмеженнях щодо евристичної моделі.

Тому в роботі пропонується принципово відмінний підхід якісного аналізу системи Ходжкіна-Хакслі, що використовує мультиваріативний метод імітаційного моделювання із подальшим застосуванням алгоритмів data mining. Отримані таким чином критерії можуть бути представлені у вигляді структур знань – класифікаційних правил, або ж дерев рішень. Результати дають можливість отримувати умови для класифікації типів збудливості для визначених груп клітин, а впровадження в моделі керування може бути застосоване для конкретних патологій та певних груп пацієнтів.

Таким чином, розробка методу якісного аналізу електричної активності клітини для системи Ходжкіна-Хакслі, його математичної та комп’ютерної

Зв’язок роботи з науковими програмами, планами, темами. У 1998 році Всесвітня організація охорони здоров’я прийняла документ «Політика досягнення здоров’я для всіх у XXI столітті». Відповідно до нього в Україні розроблена Міжгалузева комплексна програма «Здоров’я нації». Її завданнями є зміцнення здоров’я населення, збереження працездатності, поліпшення демографічної ситуації в державі та підвищення ефективності медико-санітарної допомоги. Саме на сприяння їх виконанню націлена дисертаційна робота.

Базовими для даної роботи стали комплексні науково-дослідні роботи Тернопільського державного медичного університету ім. І. Я. Горбачевського

на теми: «Медичні закономірності та інформаційні моделі перебігу патологічних процесів при різних функціональних умовах та їх корекція» (№ ДР 0110U001937), «Розробка і вдосконалення новітніх технологій ранньої діагностики та операційного лікування хірургічних захворювань на засадах доказової медицини» (№ ДР 0110U001937) та «Системні дослідження та інформаційні технології в задачах медичної науки та освіти» (№ ДР 0113U001800), в яких здобувач була виконавцем.

Мета і задачі дослідження. Метою роботи є удосконалення методів і засобів математичного, комп'ютерного моделювання та обчислювальних методів для їх інтеграції в інформаційні технології діагностики електрофізіологічних процесів збудливості клітин.

Для досягнення мети розв'язано такі задачі:

- розвинути теорію математичного моделювання електрофізіологічних процесів збудливості клітин шляхом розробки нових алгоритмів дослідження стійкості, керування нелінійною динамікою, а також мультиваріативних методів якісного аналізу системи Ходжкіна-Хакслі;
- розвинути та ефективно використати методи обчислювальної математики стосовно вирішення проблем створення і дослідження нових обчислювальних методів і алгоритмів, що враховують особливості електрофізіологічних процесів збудливості клітин, забезпечивши створення ефективних програмних засобів комп'ютерної реалізації системи типу Ходжкіна-Хакслі;
- розвинути теорію побудови програмних систем моделювання, а також систем, методів та засобів напівнатурного моделювання електричної активності клітинних мембран, включаючи інформаційні технології їх використання при проведенні електрофізіологічних досліджень;
- розробити новий метод організації та оптимізації процесів моделювання збудливості клітин на основі системи Ходжкіна-Хакслі, включаючи процеси підготовки первинної інформації, визначення складу та структури, настроювання та верифікації, перевірки та забезпечення якості

комп’ютерних моделей, дослідження моделей для різних типів збудливості клітин, інтерпретації результатів моделювання.

Об’єкт дослідження – процеси збудливості нервових клітин.

Предмет дослідження –математичні моделі на основі систем нелінійних диференціальних рівнянь та їх застосування в задачах якісного аналізу процесів збудливості нервових клітин.

Методи дослідження. Моделювання здійснюється в класах нелінійних диференціальних рівнянь типу Ходжкіна-Хакслі. Для дослідження нелінійної динаміки в моделях використовували метод з використанням експонент Ляпунова. При вивченні задач керування біфуркаціями за допомогою прикладеного струму використовували принцип максимуму Понтрягіна та прямий чисельний метод. Класифікацію типів збудливості клітин здійснювали спеціально розробленим методом, що включав імітаційне моделювання (метод Монте-Карло) та алгоритми data mining (індукція дерева рішень та метод послідовного покриття). Методи розробки програмного забезпечення базуються на використанні мови UML та об’єктно-орієнтованої мови програмування Java.

Наукова новизна одержаних результатів полягає в тому, що

- отримала подальший розвиток теорія математичного моделювання електрофізіологічної активності клітин за рахунок розробки для системи Ходжкіна-Хакслі методів дослідження стійкості, оптимізації, керування та нелінійного аналізу;

- вперше розроблено математичні методи системного аналізу та прийняття рішень на основі системи Ходжкіна-Хакслі для якісного аналізу електричної активності нервових клітин;

- вперше розроблено математичні методи керування біфуркацією у системі Ходжкіна-Хакслі, що виникає при зміні прикладеного струму, які забезпечують створення ефективних програмних засобів комп'ютерної реалізації;

- вперше запропоновано підхід дослідження системи Ходжкіна-Хакслі у різних режимах їх функціонування на основі мультиваріативного методу, що включає алгоритм послідовного покриття або ж алгоритм індукції дерева

рішень та забезпечує біологічну інтерпретацію результатів моделювання;

- отримав подальший розвиток обчислювальний метод розрахунку експонент Ляпунова, який було модифіковано та спеціаліалізовано з метою підвищення його ефективності з урахуванням особливостей нелінійної динаміки в системі Ходжкіна-Хакслі.

Практичне значення одержаних результатів. Методи системного аналізу, розроблені в даній роботі, доведено до комп'ютерної реалізації. Розроблено Web-інтегроване програмне середовище електрофізіологічних досліджень (Свідоцтво про реєстрацію авторського права на твір №57045, 17.10.2014, Державна служба інтелектуальної власності України: «Система підтримки прийняття рішень в системних медичних дослідженнях») (Додаток Б1), (Свідоцтво про реєстрацію авторського права на твір №57280, 20.11.2014, Державна служба інтелектуальної власності України: «Мультиваріативний метод дослідження збудливості нейрона з індукцією дерева рішень») (Додаток Б2). Розроблені середовища містять програмний інтерфейс, орієнтований на користувача, та відкриту бібліотеку відповідних Java-класів (Додаток Г).

Результати роботи впроваджено у наукову роботу та навчальний процес Тернопільського державного медичного університету імені І.Я. Горбачевського, зокрема розроблену математичну модель та програмне середовище електрофізіологічних досліджень на основі системи Ходжкіна-Хакслі використано:

- в науковому процесі кафедри патологічної фізіології Тернопільського державного медичного університету ім. І. Я. Горбачевського у матеріалі курсу лекцій та практичних занять під час вивчення предмету «Патологічна фізіологія» (Додаток А1);

- в навчальному процесі кафедри нормальної фізіології Тернопільського державного медичного університету ім. І. Я. Горбачевського у матеріалі курсу лекцій та практичних занять в розділі вивчення медичних інформаційних систем (Додаток А2);

- в науково-навчальному процесі кафедри фізики, інформатики та медичної апаратури Одеського національного медичного університету у матеріалі курсу лекцій та практичних занять під час вивчення предмету «Медична інформатика» (Додаток А3);

- в науковому процесі кафедри комп'ютерних технологій Луцького національного технічного університету для використання при вирішенні задач класифікації в складних динамічних системах (Додаток А4).

Особистий внесок здобувача. Всі результати, які становлять основний зміст дисертації, автор отримав самостійно. У наукових працях, опублікованих із співавторами, автору дисертації належить: у [77] – розробка концептуальної моделі системи підтримки прийняття рішень для медичних системних досліджень; [78] – розробка методу побудови оптимального керування біфуркацією в моделі Ходжкіна-Хакслі на основі принципу максимуму; у [79] – розробка програмного забезпечення для реалізації чисельного методу оптимального керування біфуркацією в моделі Ходжкіна-Хакслі; у [80] – розробка чисельного методу та його програмна реалізація з метою дослідження нелінійної динаміки в моделі Ходжкіна-Хакслі на основі експонент Ляпунова; у [81] – розробка алгоритму та його програмна реалізація з метою якісного аналізу системи Ходжкіна-Хакслі електричної активності аксона на основі класифікаційних правил; у [82] − розробка та програмна реалізація мультиваріативного методу якісного аналізу моделі Ходжкіна-Хакслі з індукцією дерева рішень; у [83] – розробка алгоритмів якісного аналізу моделі Ходжкіна-Хакслі; [84] – розробка мультиваріативного методу якісного аналізу моделі Ходжкіна-Хакслі для різних типів збудливості.

Апробація результатів дисертації. Основні результати проведених досліджень дисертації доповідались й обговорювались на міжнародних та вітчизняних науково-практичних конференціях: VI науково-практичній конференції «Здобутки клінічної та експериментальної медицини» (Тернопіль, 2011); 18-й міжнародній конференції «Problems of Decision Making under Uncertainties» (Ялта, 2011); 10-й ювілейній міжнародній науково-практичній

конференції «Биофизические стандарты и информационные технологии в медицине» (Одеса, 2011); XIV міжнародній науковій конференції ім. академіка М. Кравчука (Київ, 2012); міжнародній конференції «Проблеми прийняття рішень в умовах невизначеності» (Мукачево, 2012); XXI міжнародній конференції «Проблеми прийняття рішень в умовах невизначеності» (Східниця, 2013); XVI міжнародній конференції «Моделювання та дослідження стійкості динамічних систем» (Київ, 2013); XXII міжнародній конференції «Проблеми прийняття рішень в умовах невизначеності» (Ялта-Форос, 2013); конференції до 90-річчя з дня народження ак. В. М. Глушкова (Київ, 2013); 5-й науково-практичній конференції з міжнародною участю «Науково-технічний прогрес і оптимізація технологічних процесів створення лікарських препаратів» (Тернопіль, 2013); 20-й міжнародній конференції «Проблеми прийняття рішень в умовах невизначеності» (Брно, 2014).

У повному обсязі робота доповідалась й отримала позитивний відгук на семінарах: в ДВНЗ «Тернопільський державний медичний університет імені І. Я. Горбачевського МОЗ України» та Тернопільському національному технічному університеті імені Івана Пулюя.

Публікації. За темою дисертаційної роботи опубліковано 22 наукові праці: [74 - 84] наукові статті у фахових виданнях з технічних наук, з них 4 у виданнях, які індексуються у міжнародних наукометричних базах: [74, 84] – Web of Science, [77, 83] – Index Copernicus, 12 тез міжнародних та всеукраїнських конференцій [85 - 96], 1 працю опубліковано без співавторів [3]. Отримано 2 свідоцтва на авторські твори [97, 98].

РОЗДІЛ 1
МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ ЕЛЕКТРИЧНОЇ ЗБУДЛИВОСТІ БІОЛОГІЧНИХ КЛІТИН

Математичне моделювання знайшло застосування в медико-біологічних системах на усіх рівнях – від органів і систем до клітинного.

Для біологічних об’єктів характерні насамперед біохімічні процеси, кількісний аспект яких складно, а частіше і неможливо подати у вигляді математичних рівнянь. Проте і в цій галузі здійснюють спроби застосування математичного моделювання як перспективного методу дослідження. Проведення збудження у нервових і м’язових волокнах уже тривалий час є об’єктом дослідження багатьох учених.

Сьогодні загальновизнаною для електричної активності клітин є математична модель Ходжкіна-Хакслі [99], яка ґрунтується на даних численних електрофізіологічних експериментів, проведених для нервових клітин – нейронів. Враховуючи одинакову біофізичну природу процесів збудження, у попередніх дослідженнях було розроблено математичні моделі електичної активнності клітинних мембран для основних груп клітин. Потребують розробки методи якісного аналізу для запропонованих моделей типу Ходжкіна-Хакслі.

1.1 Математична модель електричної активності клітин Ходжкіна-Хакслі і її застосування в електрофізіології

Розробка рівнянь математичної моделі Ходжкіна-Хакслі до моделювання електричної активності гігантського аксона кальмара ґрунтується на такій фізичній моделі, представленій у вигляді електричної схеми (рис. 1.1).

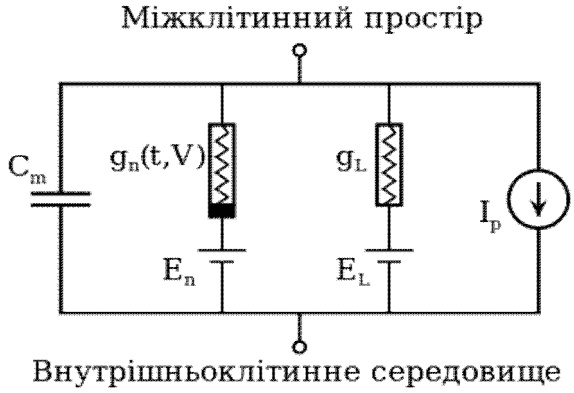


Рис. 1.1 [104]. Фізичне моделювання електричної активності клітинної мембрани на основі підходу Ходжкіна-Хакслі

У цій моделі кожен компонент збудливої клітини описується як електричний елемент. Подвійний ліпідний шар клітинної мембрани представляється як електроємність C_m. Іонні канали представлені електричною провідністю g_i, де i – специфічний іонний канал, яка залежить як від напруги, так і часу. Залежні від потенціалу іонні канали електричної активності нелінійної провідності – g_n, а лінійної – g_L. Іонні насоси представлені джерелом струму I_{app}. Відповідно електрохімічні градієнти і потік іонів представлені батареями (Е). V – різниця між мембранним потенціалом та потенціалом спокою, яка розглядається як функція від часу t: $V=V(t)$.

Струм через біліпідний шар становитиме:

$$I_c = C_m \frac{dV}{dt}, \tag{1.1}$$

Струм через заданий іонний канал буде:

$$I_i = g_i (V - V_i), \tag{1.2}$$

де V_i – рівноважний потенціал i-го іонного каналу.

Для клітини з калієвими, натрієвими та хлорними каналами загальний струм через мембрану I становитиме:

$$I = I_c + I_K + I_{Na} + I_L . \tag{1.3}$$

Струм через заданий іонний канал буде:

$$I_i = g_i (V - V_i) . \tag{1.4}$$

де V_i – рівноважний потенціал *i*-го іонного каналу.

Для клітини з калієвими, натрієвими та хлорними каналами загальний струм через мембрану I становитиме:

$$I = I_c + I_K + I_{Na} + I_L$$

На основі такого уявлення про клітинну мембрану як про паралельно під'єднаний конденсатор C_m з іонним струмом I_{ion}, модель Ходжкіна-Хакслі виходить з такого рівняння:

$$C_m \frac{dV}{dt} + I_{ion}(V,t) = 0 , \tag{1.5}$$

де V позначає різницю внутрішнього V_i і зовнішнього V_e потенціалів $(V = V_i - V_e)$. У гігантського аксона кальмара, який було використано в експерименті Ходжкіна і Хакслі, як і в багатьох інших нервових клітин, є основними іонні струми – струм іонів натрію і струм іонів калію. Хоча існують і інші іонні струми, в першу чергу струм хлору, в теорії Ходжкіна-Хакслі вони вважаються малими і об'єднані в один – під назвою струму витоку. Оскільки для миттєвих значень кривих $I - V$ для відкритих каналів Na^+ і K^+ в

гігантському аксоні кальмарі спостерігається лінійна залежність, то (1.5) набуває вигляду

$$C_m \frac{dV}{dt} = -g_{Na}(V - V_{Na}) - g_K(V - V_K) - g_L(V - V_L) + I_{app} \tag{1.6}$$

Тут I_{app} – прикладений струм. Під час потенціалу дії встановлено приплив натрію 3.7 пмоль/см^2 і подальше витікання 4.3 пмоль/см^2 калію. Ці величини настільки малі, що можна насправді припустити, що концентрації іонів, а отже й потенціали рівноваги, є сталими і не залежать від потенціалу дії. Важливо зауважити, що вибір лінійних кривих $I - V$ для трьох різних типів каналів диктується в основному експериментальними даними. Аксони в інших видів (наприклад, хребетних) мають іонні канали, які краще описуються кривими $I - V$ іншого вигляду. Тим не менш, якісний характер результатів, отриманих Ходжкіном і Хакслі, залишається значною мірою правильним і для моделей, які використовують більш складні форми кривих $I - V$ для опису іонних струмів.

Рівняння (1.5) – це звичайне диференціальне рівняння першого порядку, яке може бути переписане у вигляді

$$C_m \frac{dV}{dt} = -g_{eff}(V - V_{eq}) + I_{app}\,, \tag{1.7}$$

де $g_{eff} = g_{Na} + g_K + g_L$, $V_{eq} = (g_{Na}V_{Na} + g_K V_K + g_L V_L)/g_{eff}$. V_{eq} – це мембранний потенціал спокою, він є балансом між змінами потенціалів для трьох іонних струмів. Насправді, в стані спокою, провідності натрію і інших іонів (струму витоку) є малі порівняно з провідністю калію. Тому значення потенціалу спокою є близьким до рівноважного потенціалу калію.

Величина $R_m = 1/g_{eff}$ є пасивним опором мембрани.

Часовою константою для цього рівняння є

$$\tau_m = C_m R_m \,, \tag{1.8}$$

порядку 1мс. Звідси випливає, що при постійному прикладеному струмі мембранний потенціал повинен швидко зрівноважитися на рівні

$$V = V_{eq} + R_m I_{ipp} \,. \tag{1.9}$$

Для досить малих прикладених струмів саме це і має місце. Однак для великих прикладених струмів спостерігається повністю інша ситуація. Якщо припустити, що модель (1.5) є коректною, то єдино можливим поясненням цих відмінностей є те, що провідності не є сталими, а залежать якимось чином від напруги. Історично склалося так, що ключовим кроком до визначення провідності стала поява можливості виміряти окремі іонні струми і від цього вже досліджувати зміни в провідності. Це стало блискучим досягненням Ходжкіна і Хакслі в 1952 році.

Отже, в основі підходу Ходжкіна-Хакслі для опису електричної активності клітинної мембрани була покладена ідея використання фізичної моделі мембрани як паралельно під'єднаного до провідностей і джерела струму конденсатора і розробки на її основі відповідної математичної моделі, що включає систему чотирьох нелінійних диференціальних рівнянь.

1.1.1 Історичні передумови появи рівнянь Ходжкіна-Хакслі для опису мембранного потенціалу. До 1939 року вважали, що мембранний потенціал відіграє важливу роль в стані мембрани, але не було жодного способу, щоб його виміряти [100]. Було відомо, що клітинна мембрана розділює концентрації різних іонів всередині і поза клітиною.

Застосовуючи рівняння Нернста Джуліус Бернштейн [107 – 109] прийшов до ідеї, що мембрана в стані спокою є напівпроникна для калію, маючи на увазі, що в стані спокою V має бути близько – 70мВ. Він вважав, що в процесі

активності виникає пробій в опорі мембрани для всіх іонних потоків і різниці потенціалів зникають, тобто V буде наближатися до нуля.

У серії з п'яти статей, які з'явилися в журналі «Journal of Physiology» в 1952 році, Алан Ходжкін і Андру Хакслі разом з Бернардом Кацом, який був співавтором провідної роботи, розглядали динамічні іонні провідності, які генерують потенціал дії нервової клітини [101 – 106]. Вони були нагороджені в 1963 році Нобелівською премією з фізіології та медицини (спільно з Джоном С. Еклсі).

Отже, передумовами до розробки моделі Ходжкіна-Хакслі послужили спостереження про різницю концентрацій іонів всередині та зовні клітини, до опису якої можуть застосовуватися рівняння типу Нернста.

1.1.2 Напруга і миттєва залежність провідності. Ключовим кроком у класифікації динаміки провідності стала розробка методу двоелектродної фіксації потенціалу (на сьогодні з цією метою використовується метод локальної фіксації потенціалу, за розробку якого в 1991 році німецькі дослідники Ервін Неєр та Берт Сакман отримали Нобелевську премію). Суть методу полягає в тому, що фіксується мембранний потенціал, а потім вимірюється струм, який повинен подаватися з метою отримання сталої напруги. Таким чином подається струм, який повинен дорівнювати трансмембранному струму. Метод фіксації потенціалу надає спосіб вимірювання трансмембранного струму. Ключовим моментом є те, що напруга може бути посилена з одного сталого рівня на інший, і таким чином іонні струми можуть бути виміряні при постійній відомій напрузі. Отже, навіть коли провідності є функціями напруги (як це насправді є), метод фіксації потенціалу усуває будь-які зміни напруги і дозволяє вимірювати провідність як функцію лише від часу.

Ходжкін та Хакслі виявили, що напруги були активовані і фіксовані на більш високому рівні, загальний іонний струм протікав спочатку всередину клітини, але в наступні моменти часу виникав зовнішній струм (рис.1.2). В силу

ряду причин вони стверджували, що початковий вхідний струм здійснюється майже повністю іонами Na^+, в той час як зовнішній струм, який розвивається пізніше здійснюється в основному за рахунок іонів K^+. При цих припущеннях, Ходжкін і Хакслі, щоб розділити загальний іонний струм замінили 90% позаклітинного натрію на холін (в'язка рідина комплексу вітамінів В, що міститься в багатьох тваринних і рослинних тканинах). При цьому аксон не збудився, лише незначно змінився потенціал спокою.

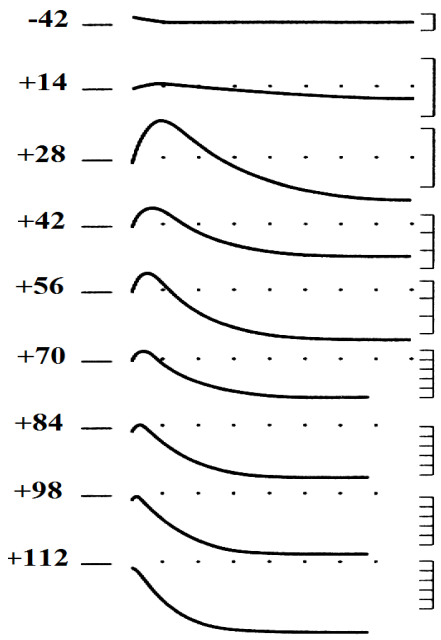


Рис.1.2. Графік експериментального результату опису загального струму мембрани у відповідь на крок деполяризації. Цифри ліворуч показують кінцеве значення мембранного потенціалу в мВ. Інтервал між точками на горизонтальній шкалі є 1мс, у той час як одна поділка на вертикальній шкалі відповідає 0,5мА/см^2 [101].

Оскільки припускається, що відразу ж після того, як напруга була посилена, іонний струм переносить всі іони Na^+, то можна виміряти початкові струми іонів Na^+ у відповідь на стрибок напруги. Відзначимо, що хоча струми Na^+ можуть бути виміряні безпосередньо відразу після стадії прикладання напруги, вони не можуть бути виміряні безпосередньо протягом більш

тривалого періоду часу, оскільки загальний іонний струм починає включати складову від струму іонів K^+.

Позначимо струм іонів Na^+ для двох випадків нормального позаклітинного і нульового позаклітинного середовищ через I^1_{Na} і I^2_{Na} відповідно. Тоді відношення двох струмів,

$$I^1_{Na} / I^2_{Na} = K \tag{1.10}$$

може бути виміряне безпосередньо з експериментальних даних.

Далі Ходжкін і Хакслі зробили ще два припущення. По-перше, припустили, що співвідношення натрієвого струму K не залежить від часу і, таким чином, є сталим протягом кожного експерименту затиску напруги. Іншими словами, амплітуда і напрям струму Na^+ може бути порушена низькою концентрацією іонів Na^+ в позаклітинному розчині. По-друге, що калієві канали не схильні до зміни позаклітинної концентрації натрію. Існує доказ того, що консідеровані канали для іонів натрія і калія незалежні. Тетродотоксини (ТТХ), як відомо, блокують струми натрію, залишаючи калієві струми майже неушкодженими, в той час як тетраетіламмоній (Tea) має протилежний ефект блокування струму калію, але не струму натрію. Насамкінець можна показати, що ввівши позначення $I_{ion} = I_{Na} + I_K$ і $K = I^1_K / I^2_K$, випливає, що якщо $I^1_{ion} - I^1_{Na} = I^2_{ion} - I^2_{Na}$, то

$$I^1_{Na} = \frac{K}{K-1}(I^1_{ion} - I^2_{ion}), \tag{1.11}$$

$$I_K = \frac{I^1_{ion} - KI^2_{ion}}{1-K}. \tag{1.12}$$

Отже, вимірявши загальні іонні струми в двох випадках, і враховуючи співвідношення струмів *K*, можна визначити окремі струми іонів Na^+ і *K*.

Із окремих струмів отримуємо провідність

$$g_{Na} = \frac{I_{Na}}{V - V_{Na}}, \qquad g_K = \frac{I_K}{V - V_K}. \tag{1.13}$$

Результат залежить від конкретної (лінійної) моделі, що використовується для опису кривої *I-V* з каналами Na^+ і *K*. Припускають, що для конкретного моменту часу для кривої *I-V* канали Na^+ і *K* є лінійними.

Рівняння Ходжкіна-Хакслі для гігантського аксона кальмара мають вигляд:

$$C_m \frac{dv}{dt} = -\overline{g}_K n^4 (v - v_K) - \overline{g}_{Na} m^3 h (v - v_{Na}) - \overline{g}_L (v - v_L) + I_{app}, \tag{1.14}$$

$$\frac{dm}{dt} = \alpha_m (1 - m) - \beta_m m, \tag{1.15}$$

$$\frac{dn}{dt} = \alpha_n (1 - n) - \beta_n n, \tag{1.16}$$

$$\frac{dh}{dt} = \alpha_h (1 - h) - \beta_h h. \tag{1.17}$$

Функції α і β запропоновані Ходжкіном і Хакслі; їх значення визначаються в одиницях $(ms)^{-1}$,

$$\alpha_m = 0.1\frac{25 - v}{\exp^{(\frac{25-v}{10})} - 1}, \tag{1.18}$$

$$\beta_m = 4\exp(\frac{-v}{18}), \tag{1.19}$$

$$\alpha_h = 0.07\exp(\frac{-v}{20}), \tag{1.20}$$

$$\beta_h = \frac{1}{\exp^{(\frac{30-v}{10})} + 1}, \tag{1.21}$$

$$\alpha_n = 0.01\frac{10 - v}{\exp^{(\frac{10-v}{10})} - 1}, \tag{1.22}$$

$$\beta_n = 0.125\exp^{(\frac{-v}{80})}. \tag{1.23}$$

Змінна v − описує мембранний потенціал, зміни якого визначаються сумою струмів що протікають через мембрану: струму іонних каналів (натрієвих і калієвих), струму витоку (іони хлору), відхилення від інших $(V = V_{eq} + v)$, вимірюється в одиницях − мВ, щільність струму – mS/cm^2 і ємність в – MF/cm^2.

Параметри $\overline{g}_{Na}$, $\overline{g}_K$, $\overline{g}_L$ визначають максимальну провідність відповідних типів каналів, V_{Na}, V_K і V_L − рівноважні потенціали за відповідним типом іонів, значення яких визначаються рівнянням Нернста [108, 109] і залежать від співвідношення концентрацій іонів по обидві сторони від мембрани. Залежно від амплітуди зовнішнього збурення можлива поява слабкого підпорогового відгуку, або генерації імпульсу.

Решта константи

$$\bar{g}_{Na} = 120,\ \bar{g}_K = 36,\ \bar{g}_L = 0.3, \tag{1.24}$$

при цьому рівноважні потенціали $v_{Na} = 115$, $v_K = -12$ і $v_L = 10.6$. На рис.1.3 показані функції станів рівноваги, а сталі часу наведені на рис.1.4.

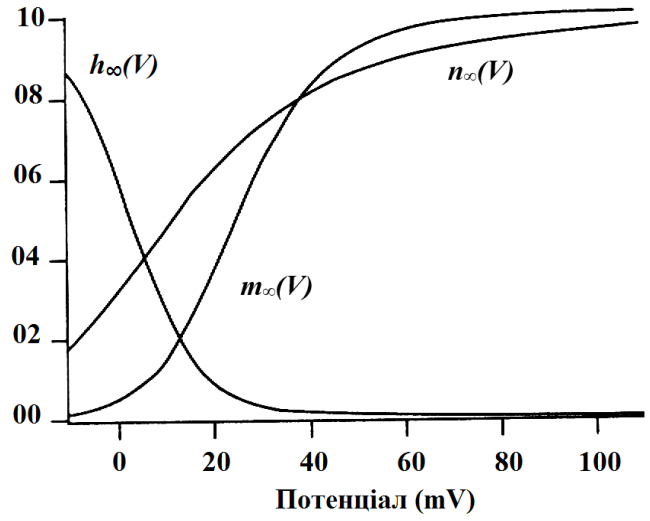


Рис. 1.3. Графік функцій станів рівноваги $m_\infty(v)$, $n_\infty(v)$ і $h_\infty(v)$.

Розглядалися прості моделі каналів для проходження іонів Na^+ і K^+ і було показано, як швидкісні константи в простих кінетичних моделях можуть бути визначені на основі даних про цілу клітину або про один канал. Подібно як моделі з (1.11) − (1.14) можуть бути отримані моделі іонних каналів, що складаються з декількох субодиниць, кожна з яких підпорядковується простій двовимірній моделі. У рівняннях Ходжкіна-Хакслі, припускається, що канал Na^+ складається з трьох "m" воріт і одних "h" воріт, кожні з яких можуть бути закриті або відкриті. Якщо ворота працюють самостійно, то частка відкритих з них становить m^3h, де m і h описуються рівняннями системи для двох каналів.

Аналогічним чином, якщо є четверо воріт *"n"* на канал калію, всі з яких повинні бути відкриті для калію, то частка відкритих каналів K^+ є n^4.

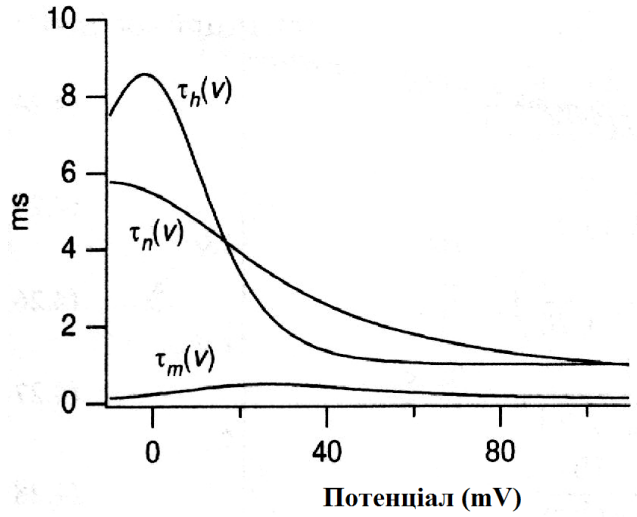


Рис. 1.4. Графік сталих часу $\tau_m = (V)$, $\tau_n = (V)$ і $\tau_h = (V)$.

Описано, як рівняння Ходжкіна-Хакслі повинні працювати в якісному вигляді. Якщо невеликі струми прикладені до клітини протягом короткого періоду часу, то потенціали повертаються швидко до рівноважного стану $v = 0$, після чого струм зникає. Рівноважний потенціал близький до калієвого потенціалу Нернста $v_K = -12$. Існує завжди конкуренція між трьома іонними струмами для управління потенціалом при відповідному потенціалі спокою. Наприклад, якщо струми витоку калію можуть бути заблоковані, або провідність натрію різко зросте, то вираз $g_{Na}(V - V_{Na})$ повинен домінувати в (рис.1.2) так довго, поки v буде нижче v_{Na}. Аналогічним чином, у той же час v вище v_K. Оскільки $v_K < v < v_{Na}$, то v обов'язково обмежується і лежить в діапазоні $v_K < v < v_{Na}$.

Якщо g_{Na} і g_K сталі, то рівновага при $v = 0$ буде стійка і після будь-якого стимулу потенціал повернеться миттєво до стану спокою, але оскільки g_{Na} і g_K можуть змінюватися, то різні струми можуть впливати по-своєму. Реальна

послідовність явищ визначається динамікою m, n і h. Найбільш важливим є те, що $\tau_m(v)$ набагато менше, ніж $\tau_n(v)$ або $\tau_h(v)$. Тому $m(t)$ реагує набагато швидше на зміни в v, n, або h. Тому система Ходжкіна-Хакслі є збудливою системою.

Якщо потенціал злегка піднятий невеликим стимулюючим струмом, то система повертається до стабільної рівноваги. Коли протягом певного періоду часу потенціал v злегка підняти невеликим стимулюючим струмом, то система повернеться до стійкої рівноваги. Коли ж протягом певного періоду часу потенціал v є підвищений, активацією натрію m відстежуючи $m_\infty(v_0)$, і якщо збуджуюча сила струму досить велика, щоб підвищити потенціал, тобто $m_\infty(v_0)$ є на досить високому рівні (вище його порогу), то, перш ніж система може повернутися до спокою, m збільшиться настільки, щоб змінити струм, внаслідок чого виникає автокаталітичний внутрішній натрієвий струм.

Тепер, коли потенціал m продовжує зростати і вхідний струм натрію збільшується, то це веде до виникнення потенціалу v_{Na}.

Отже, метод фіксації потенціалу дозволив встановити параметри, пов'язані з електричною провідністю через клітинні мембрани і які в подальшому були використані для розробки математичних моделей на основі нелінійних диференціальних рівнянь.

1.2 Дослідження електричної активності клітин на основі двовимірних моделей

Є два способи, коли система Ходжкіна-Хакслі може бути змодельована автономним генератором. Перший – надати постійний струм достатньої сили. При цьому деякі із змінних моделей мають швидку кінетику, а інші – набагато повільнішу: зокрема, m і v – швидкі змінні (тобто канал іонів Na^+ активізується швидко і мембранний потенціал змінюється швидко), в той час як канали K^+ активуються повільно. Таким чином, на початкових етапах потенціалу дії n і h залишаються практично незмінними, а m і v

розрізняються. Це дозволяє повний 4-вимірний фазовий простір розбити на дрібні шматки, фіксуючи повільні змінні і враховуючи поведінку моделі залежно тільки від двох швидких змінних. Хоча цей опис є точним тільки для початкових стадій потенціалу дії, він надає зручний спосіб для вивчення процесу збудження клітини.

1.2.1 Швидка фазова площина. Отже, зафіксуємо повільні змінні n і h у своїх станах спокою, які називаємо n_0 і h_0. Розглянемо, як m і v поводитимуться у відповідь на стимуляцію. Диференціальні рівняння для фазової площини матимуть вигляд

$$C_m \frac{dv}{dt} = -\bar{g}_K n_0^4 (v - v_K) - \bar{g}_{Na} m^3 h_0 (v - v_{Na}) - \bar{g}_L (v - v_L), \tag{1.25}$$

$$\frac{dm}{dt} = \alpha_m (1 - m) - \beta_m m, \tag{1.26}$$

або,

$$\tau \frac{dm}{dt} = m_\infty - m. \tag{1.27}$$

Двовимірна система може бути найбільш легкою при дослідженні фазових ділянок (m, v), які наведено на рис.1.5. Криві визначаються із рівностей $dv/dt = 0$ і $dm/dt = 0$. Крива $m = m_\infty(v)$, яка представлена на рис.1.3. визначається з

$$v = \frac{-\bar{g}_K m^3 h_0 v_{Na} + \bar{g}_K n_0^4 v_K + \bar{g}_L v_L}{\bar{g}_{Na} m^3 h_0 + \bar{g}_K n_0^4 + \bar{g}_L}. \tag{1.28}$$

Для параметрів системи Ходжкіна-Хакслі m і v перетинаються в трьох місцях, що відповідають трьом стаціонарним станам швидких рівнянь. Ці три точки перетину не є стаціонарними станами повної системи, а лише швидкої

підсистеми. В контексті швидкої фазової площини їх називають стаціонарними станами. Позначимо ці три стійких стани як v_r, v_s і v_e.

Показано, що v_r і v_e є стійкі стаціонарні стани швидкої підсистеми, в той час як v_s − сідлова точка, тобто v_s − одновимірний стійкий многовид. Стійкий многовид ділить площину (m, v) на дві області: будь-які траєкторії, починаючи зліва від стійкого многовиду уникають потрапляння до стану v_e і в кінцевому підсумку повинні повернутися до стану спокою v_r. А будь-яка траєкторія, починаючи праворуч від стійкого многовиду, уникає повернення в стан спокою v_r, а прямує в збуджений стан v_e.

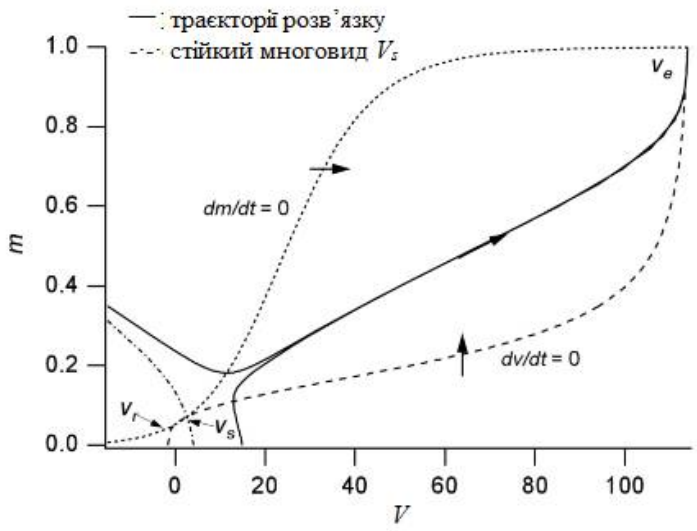


Рис. 1.5. Фазова площина системи Ходжкіна-Хакслі. Показано пунктирними лініями нулькліни $dv/dt = 0$ і $dm/dt = 0$ ($h_0 = 0.596$, $n_0 = 0.3176$), суцільними лініями -- дві звичайні траєкторії і сідлову точку v_s.

Збільшення концентрації позаклітинних іонів калію впливає на збільшення калієвого потенціалу Нернста, ефективне підвищення потенціалу спокою (оскільки потенціал спокою близький до калієвого потенціалу Нернста). Цей механізм створення автономного генератора з нормально

збудливих, але не коливальних клітин має важливе значення для деяких серцевих аритмій.

Отже, 4-вимірний фазовий простір моделі Ходжкіна-Хакслі може бути на початкових стадіях потенціалу дії зведений до двовимірного за рахунок розгляду лише двох швидких змінних моделі. Такий підхід надає зручний спосіб для вивчення процесу збудження клітини.

1.2.2 Якісний аналіз на основі двовимірної моделі. Фітцхью [110 – 112] виконав якісне дослідження системи Ходжкіна-Хакслі, що дозволяє краще зрозуміти якісну поведінку моделі. Більш докладні аналізи також були наведені у Рінзел [113], Тру [114], Коул та ін. [115] та Сабах і Спенглер [116]. Підхід Фітцхью грунтується на двох стійких стаціонарних станах, що спричиняють пороговий ефект. Будь-яке відхилення від стану спокою, яке є не досить великим, щоб перетнути стійкий многовид, в кінцевому підсумку повертається до стану спокою. У той час збурення, яке перетинає стійкий многовид, призводить до стану збудження. Приклади траєкторій представлено на рис.1.5., де пунктирними лініями показано нульклiни – розв'язки рівнянь $dv/dt = 0$ і $dm/dt = 0$.

Якщо m і v були єдиними змінними в моделі, то v залишатиметься на v_e на невизначений термін. Таким чином, щоб бачити те, що відбувається на більш тривалому часовому масштабі розглянемо, як повільно варіації в n і h впливають на якісний характер швидкої фази площині. Зазначимо, що, оскільки $v_e > v_r$ випливає, що $h_\infty(v_e) < h_\infty(v_r)$ і $m_\infty(v_e) > m_\infty(v_r)$ Отже, в той час як v знаходиться в збудженому стані, а h починає зменшуватися, таким чином, інактивації Na^+ провідності і n починають зростати, таким чином, активації K^+ провідність відзначають, що m в швидкій фазі площини не залежить від n і h, v. На рис.1.5. були зроблені з використанням стаціонарних значень n і h, різні значення n і h змінювали форму v. У n міру збільшення і h зменшення, зменшуване v рухається вліво і вгору, як показано на рис.1.6. У v

переміщується вгору і вліво, v_e і v_s рухатимуться в напрямку один до одного, в той час v_r переміщатиметься вліво.

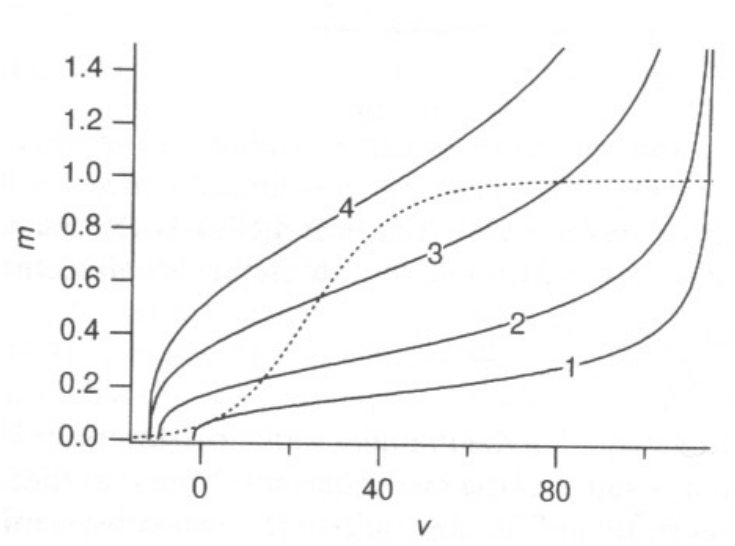


Рис. 1.6. Швидка фазова площина як функція повільних змінних системи Ходжкіна-Хакслі. Показано пунктирною лінією нульклину *m* та зникнення сталих станів. Значення параметрів для цих кривих − (1) $h_0 = 0.596$, $n_0 = 0.3176$; (2) $h_0 = 0.4$, $n_0 = 0.5$; (3) $h_0 = 0.2$, $n_0 = 0.7$; (4) $h_0 = 0.1$, $n_0 = 0.8$.

Під час цієї фази напруга на v_e повільно зменшується. Зрештою, v_e і v_s зливаються і зникають в сідлово-вузловій біфуркації. Коли це відбувається, то лише v_r залишається у стійкому стані, і тому розв'язок має повернутися в стан спокою. Оскільки пунктирна лінія перейшла вгору і вліво, v_r не є стаціонарним станом повної системи. Однак, коли v зменшується до v_r, n і h повертаються до їх стійкого стану і, в той час, як вони це роблять, v_r повільно збільшується до тих пір, поки не досягнуто стаціонарний стан повної системи і потенціал дії є повним.

Схема повного потенціалу дії показана на рис.1.7. і тут позначені важливі моменти для порівняння з фазовою площиною системи. Графік швидко-

повільної площини системи Ходжкіна-Хакслі показано на рис.1.8. Графік узагальненої фазової площини Фітцхью-Нагумо показано на рис.1.9.

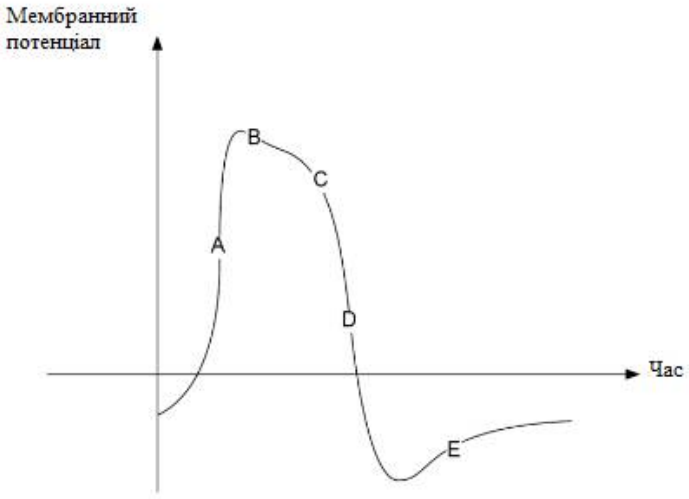


Рис. 1.7. Графік повного потенціалу дії. A: Понадпороговий стимул викликає швидке зростання v аж до збудливого стану. B: v знаходиться в збудженому стані v_e, що повільно зменшується при зростанні n та спаданні h, тобто, коли v_e прямує до v_s. C: v_e і v_s зникають при біфуркації сідлового вузла. D: розв'язок повинен повернутися до стану спокою v_r. E: n і h повільно повертаються до їх станів спокою, і при цьому v_r повільно зростає аж до досягнення стану спокою чотиривимірної системи.

Подання системи Ходжкіна-Хакслі в термінах двох змінних, однієї швидкої і однієї повільної − основа моделі Фітцхью-Нагумо для збудливості.

Особливістю моделі Фітцхью-Нагумо є те [117 – 119], що вона складається з двох динамічних систем і може бути вивчена з використанням фазових методів. Є два характерні фазові портрети. За припущенням, існує тільки один стаціонарний стан при $v = v^*$, $w = w^*$, з $f(v^*, w^*) = g(v^*, w^*) = 0$. Не обмежуючи загальності припускається, що цей стійкий стан знаходиться в початку координат. Приблизно посередині гілки розв'язку $v = V_0(w)$ поруч з екстремальними значеннями кривої $f(v,w) = 0$ є точка біфуркації Хопфа. Тобто,

якщо параметри змінюються таким чином, що стаціонарний стан проходить через цю точку, то виникає періодична орбіта у вигляді коливань стійкого граничного циклу.

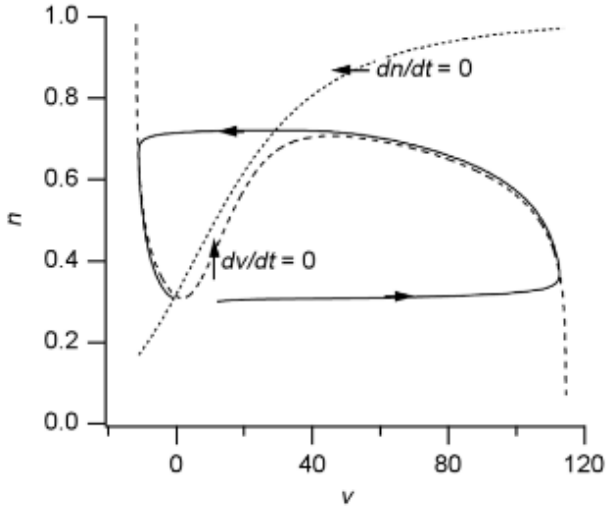


Рис. 1.8. Швидко-повільна площина системи Ходжкіна-Хакслі

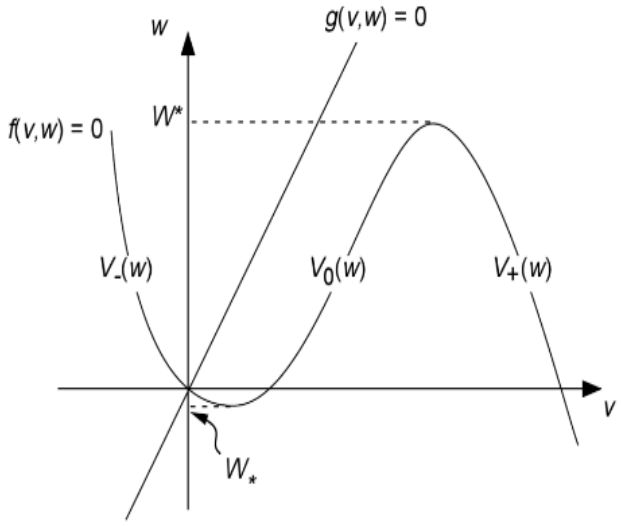


Рис. 1.9. Схема узагальненої фазової площини Фітцхью-Нагумо.

Отже, на основі двовимірної моделі Фітцхью-Нагумо можна провести якісний аналіз моделі електричної збудливості клітин, побудувавши на фазовій площині відповідні нульокліни та отримавши на їх перетині стаціонарні стани.

Така модель показує шлях виникнення біфуркації Хопфа та стійкого граничного циклу.

1.3 Моделювання електричної збудливості для різних видів клітин

1.3.1 Моделі збудливості серцевих клітин. Основні типи серцевих клітин – це вузлові клітки (синоатрикулярних (SA) і атріовентрикулярних (AB) вузлів), клітини волокон Пуркіньє і клітини міокарда, кожні з яких мають дещо інші функції.

Основна функція синоатрикулярних (SA) вузлових клітин є забезпечення сигналу водія ритму серця. Функція атріовентрикулярних вузлових клітин – передавати електричний сигнал від передсердя до шлуночка з затримкою. Клітини волокон Пуркіньє призначені, в першу чергу, для швидкого проведення сигналу з метою активування клітин міокарда, передсердних, шлуночкових і м'язових клітин, які є скорочувальними а також збудливими.

Через різні функції типи клітин мають дещо відмінні форми потенціалів дії. Потенціал дії для вузлових клітин SA короткий, схожий на потенціал дії Ходжкіна-Хакслі, в той час як потенціали дії як для волокон Пуркіньє, так і клітин міокарда мають суттєво пролонговану дію на потенціал, що сприяє м'язовому скороченню. Типові іонні концентрації наведені в (Додаток В табл.1.1)

Після винайдення моделі Ходжкіном і Хакслі, була проведена значна робота, щоб застосувати таку модель для багатьох різних типів клітин, в тому числі серцевих клітин. Кількісні моделі клітин серця є типу Ходжкіна-Хакслі. Вони служать меті відтворення особливостей форми потенціалу дії, намагаючись дати обгрунтовані пояснення механізмам (через іонні канали і струми) їх поведінки. Основні труднощі при дослідженні збудливості серцевих клітин є те, що існує безліч різних типів клітин і багато різних типів іонних каналів. Навіть у межах одного типу клітин, є суттєві відмінності. Наприклад, в шлуночках, епікардіальних, мідміокардіальних і ендокардіальних клітини

існують помітні відмінності в тривалості потенціалу дії. Атріовентрикулярні (АВ) вузлові клітин істотно розрізняються, ділячись на кілька підтипів. У багатьох відношеннях гігантського аксона кальмара, опрацьований Ходжкіном і Хакслі був ідеальний для дослідження, оскільки кількість і типи каналів невеликі.

1.3.2 Волокна Пуркіньє. Феноменологічний підхід. Перша модель, що описує потенціал дії серцевої клітини була запропонована Ноблем [120] для волокон клітин Пуркіньє. Основна мета цієї моделі – показати, що потенціал дії клітин волокон Пуркіньє, які помітно відрізняються від потенціалу дії аксону кальмара, може бути описаний системою типу Ходжкіна-Хакслі. Потенціал дії у клітин волокон Пуркіньє автоколивальний з різким рухом вгору, що проскакує і повертається протягом тривалого часу (300-400мс в порівнянні з 3мс для кальмара аксона) до стану спокою.

Такі моделі мають тип системи Ходжкіна-Хакслі. Вони представлені в термінах іонних струмів і провідностей. В моделі є струми, що визначаються як внутрішній струм натрію, струму калію назовні і струму витоку хлориду, всі з яких, прискається, що задовольняють лінійну залежність $I - V$,

$$I = g(V - V_{eg}). \tag{1.29}$$

Для моделі Нобля [121], всі зміни в провідності, які були виміряні в розчинах з дефіцитом натрію розглядалися для струмів, які переносять іони калію і тому називаються калієвими струмами (таким чином, струм хлориду прийнятий рівним нулю).

На основі традиційного підходу Ходжкіна-Хакслі баланс трансмембранних струмів виражається законом збереження

$$C_m \frac{dV}{dt} + g_{Na}(V - V_{Na}) + (g_{K_1} + g_{K_2})(V - V_K) + g_{an}(V - V_{an}) = I_{app}, \tag{1.30}$$

тут $g_{an} = 0$ і $V_{an} = -60$ провідний і рівноважний потенціал, відповідно, для аніонів струму та іонів хлору (який, в силу попереднього зауваження, не істотний). До того ж, $V_{Na} = 40$ і $V_K = -100$.

Система передбачає розгляд двох різних типів калієвих каналів. При цьому миттєва напруга залежить від каналу і від часу. Часовозалежний калієвий канал, який має аналогічну форму до калієвого каналу Ходжкіна-Хакслі, за винятком того, що він приблизно в 100 раз повільніший у своїй відповіді. Цей струм іноді називають затримкою випрямляча струму, тому що він затримується й тому, що це в першу чергу струм назовні. Провідність для цього каналу g_{K_2} залежить від змінної активації калію залежно від часу n згідно

$$g_{K_2} = 1.2\mathrm{n}^4 . \tag{1.31}$$

Провідність для миттєвого каналу описано емпірично

$$g_{K_1} = 1.2\exp\left(-\frac{V+90}{50}\right) + 0.015\exp\left(\frac{V+90}{60}\right). \tag{1.32}$$

Провідність натрію для моделі Нобля є схожою до системи Ходжкіна-Хакслі

$$g_{Na} = 400m^3h + g_i , \tag{1.33}$$

тут $g_i = 0.14$.

Часова залежність змінних m, n і h має вигляд

$$\frac{dw}{dt} = \alpha_w(1-w) - \beta_w w , \tag{1.34}$$

з $w = m, n$ або h де α_w і β_w мають вигляд

$$\frac{C_1 \exp\left(\frac{V - V_0}{C_2}\right) + C_3 (V - V_0)}{1 + C_4 \exp\left(\frac{V - V_0}{C_3}\right)} \tag{1.35}$$

Константи $C_1, \ldots, C_5$ і V_0 представлені в (Додаток В табл.1.2)

У моделі Нобля $C_m = 12$, що є нереально великим. Це значення було використано тому, що дає правильний масштаб часу для тривалості потенціалу дії.

Чисельні розрахунки показують, що модель Нобля виробляє потенціал дії, який має правильні характеристики. Різкий рух вгору відбувається за рахунок швидкого струму натрію всередину.

Через різкий стрибок на початку потенціалу дії неможливо відтворити потенціал дії волокон Пуркіньє за допомогою моделі Фітцхью-Нагумо з двома змінними. Однак, поклавши $m = m_\infty(V)$, модель Нобля може бути зведена до моделі трьох змінних, яка зберігає головні якісні властивості оригінальної моделі.

1.3.3 Синоатріальний (SA) вузол. Основна функція клітин SA вузлів – розпочати регулярне серцебиття. Вони мають малоскоротливу функцію і, отже, містять мало кальцію. Найбільш широко використовувана модель потенціалу дії для SA вузлових клітин походить з роботи Янагіхари [122] і отримала скорочену назву YNI-модель на честь авторів. Модель включає в себе чотири залежних від часу струми. Це струм натрію I_{Na}, який швидко проникає всередину клітини і струм калію (обидва є аналогічні до струмів Ходжкіна-Хакслі), повільний струму всередину I_s, струм гіперполяризації I_h, струму витоку I_i.

$$C_m \frac{dV}{dt} + I_{Na} + I_K + I_i + I_s + I_h = I_{app}\,, \tag{1.36}$$

де

$$\alpha_p = 9\times10^{-3}\frac{1}{1+\exp\left(-\dfrac{V+3.8}{9.71}\right)} + 6\times10^{-4}\,, \tag{1.38}$$

$$\alpha_q = 3.4\times10^{-4}\frac{(V+100)}{\exp\left(\dfrac{V+100}{4.4}\right)-1} + 4.95\times10^{-5}\,, \tag{1.39}$$

$$\beta_q = 5\times10^{-4}\frac{(V+40)}{1-\exp\left(-\dfrac{V+40}{6}\right)} + 8.45\times10^{-5}\,, \tag{1.40}$$

$$\alpha_d = 1.045\times10^{-2}\frac{(V+35)}{1-\exp\left(-\dfrac{V+35}{2.5}\right)} + 3.125\times10^{-2}\frac{V}{1-\exp\left(-\dfrac{V}{4.8}\right)}, \tag{1.41}$$

Поведінка розв’язків рівнянь потенціалу дії формується аналогічно потенціалу дії Ходжкіна-Хакслі, але є періодичною за часом і повільнішою. Струм натрію є швидким, і є невелика втрата в точності в заміні $m(t)$ з $m_\infty(V)$. Найбільш значний струм в моделі YNI є повільний вхідний струм I_s. Це не тільки струм, що забезпечує більшу частину руху «насоса» вгору, він також несе відповідальність за коливання після реполяризації.

Оскільки потенціал дії вузла SA не має первинного сплеску, то його відносно легко можна відтворити за допомогою двох змінних моделі Фітцхью-Нагумо.

1.3.4 Шлуночкові клітини. Рівняння Білера-Рейтера [123] з'явилися незабаром після моделі MNT [124] і моделювали електричний струм для шлуночків клітини міокарда. Як і моделі описані раніше, ця модель заснована на даних, отриманих від напруги затиску експериментів. Рівняння Білера-Рейтера менш складні, що рівняння MNT, а є тільки 4 трансмембранні струми, які описані, два всередині струму, один швидкий і один повільний і два зовнішні струми, один раз незалежні і залежні один раз.

Як звичайно, існує струм іонів натрію всередину

$$I_{Na} = (4\mathrm{m}^3 hj + 0.003)(V - 50), \tag{1.42}$$

із змінними *m, h,* і *j*. Тут Білер і Рейтер визнали за необхідне включити змінну реактивації *j*, оскільки процес активації відбувається набагато повільніше, ніж інактивації і не може бути точно змодельований з однією змінною *h*. Таким чином, струм натрію активується *m*, інактивується *h*, і реактивується *j* за допомогою найповільнішої із трьох змінних. Функції h_∞ і j_∞ ідентичні.

Струм калію має дві складові – незалежний від часу струм

$$I_K = 1.4\frac{\exp(0.04(V+85))-1}{\exp(0.08(V+53))+\exp(0.04(V+53))} + 0.07\frac{V+23}{1.0-\exp(-0.04(V+23))} \tag{1.43}$$

і струм активування назовні

$$I_x = 0.8\mathrm{x}\frac{\exp(0.04(V+77))-1}{\exp(0.08(V+35))}. \tag{1.44}$$

Струм кардіостимулятора калію використовується в моделях MNT.

Основна відмінність між шлуночковою клітиною Пуркіньє і вузлом SA або вузлового осередку є наявність кальцію, який необхідний, щоб активувати механізм скоротливості. Для рівнянь Білер-Ройтер, приплив кальцію моделюється за допомогою повільного вхідного струму

$$I_S = 0.09\, fd(V + 82.3 + 13.0287 \ln[Ca]_i), \tag{1.45}$$

Активується d і інактивується j. Від виникнення потенціалу для I_s кальцію залежить внутрішня концентрація кальцію, що моделюється за допомогою

$$\frac{dc}{dt} = 0.07(1 - c) - I_S, \tag{1.46}$$

де $c = 10^7 [Ca]_i)$.

α_w і β_w мають вигляд (1.37) з константами, представленими в (Додаток В табл.1.5) У цих рівняннях V в мВ, провідності в одиницях mS/cm^2 і час вимірюється в мілісекундах (мс). Форма потенціалу дії Белера-Ройтера – довге плато потенціалу дії Білера-Ройтера підтримується шляхом повільного струму (кальцію) всередину і повернення до потенціалу спокою за посередництва повільного струму калію назовні, показано на рис.1.10.

В Додатку В табл.1.6 представлений огляд струмів і сталих, використовуваних у кожній з чотирьох моделей, представлених вище.

Представлені попередньо моделі електричної активності клітин грунтуються на певних наближеннях і спрощеннях. Тому подальші дослідження стосувалися поліпшення моделей з метою кращого узгодження з фізіологією процесів.

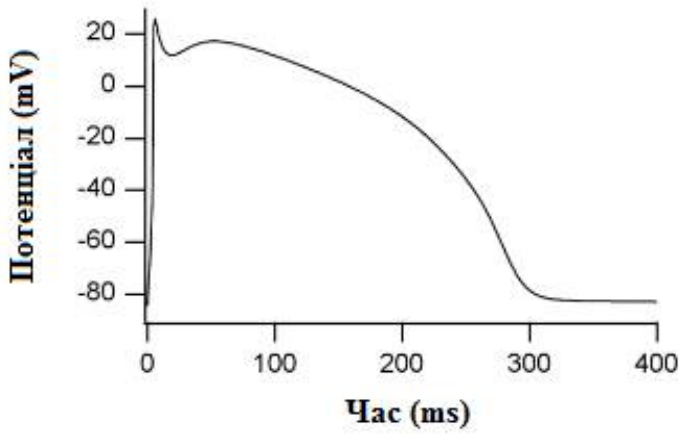


Рис.1.10. Потенціал дії Білера-Ройтера.

Отже, дослідження серцевих клітин, що мають складнішу структуру, і надалі залишається актуальним. Особливо це стосується якісної поведінки моделей типу Ходжкіна-Хакслі для вказаних груп клітин. Кількісні деталі виявляються не настільки принциповими. Створювані математичні моделі відображають майстерність моделювання, а також те, щоб у моделі деякі ефекти зберігалися, а інші відкидалися, а треті – усереднювалися.

Тому в підрозділі дано короткий огляд найбільш важливих моделей поведінки клітин серця і представлено основні відмінності між типами клітин.

Висновки до першого розділу

У розділі представлено моделі електрофізіології для певних груп клітин з використанням підходу Ходжкіна-Хакслі [101] та Фіцхью-Нагумо [117, 118].

Математична теорія Ходжкіна-Хакслі насьогодні є загальновизнаною і вона ґрунтується на даних чисельних електрофізіологічних експериментів, проведених для нервових клітин – нейронів. З математичної точки зору вона грунтується на системі чотирьох нелінійних диференціальних рівнянь, що робить неможливим застосування аналітичних підходів і методів.

У моделі Фіцхью-Нагумо спрощено чотиривимірний фазовий простір шляхом застосування низки двовимірних перетинів при різних фіксованих

значеннях n і h. Однак, розглядаючи різні перетини можна встановити нові аспекти потенціалу дії. Зокрема, розглядаючи перетин з участю однієї швидкої змінної і однієї повільної змінної отримуємо опис моделі Ходжкіна-Хакслі.

Для нервових і м'язових тканин у спокої характерна різниця потенціалів між внутрішньою та зовнішньою сторонами клітини за рахунок градієнта концентрацій різних йонів ($Na+$, $K+$, $Cl-$, $Ca++$…) через мембрану клітини. Проникність йонів через мембрану залежить від потенціалу. Якщо клітина деполяризується деяким пороговим способом, то виникає значна зміна потенціалу – потенціал дії. Для моделювання поширення потенціалу дії у нервових волокнах Ходжкін і Хакслі використовували зміну проникності мембрани з точки зору активації та інактивації йонних каналів.

На основі цих ідей і методів електрофізіологами було проведено значний обсяг досліджень і, зокрема, було отримано точні дані про серцевий потенціал дії [123]. Для розуміння ефектів поширення хвилі збудження у серці було розроблено математичні моделі. Як правило, вони ґрунтуються на даних вимірювань йонних струмів. Серед них можна виділити модель Мак-Алістера–Нобля–Цяня [124] для волокон Пуркіньє і модель Білера–Рейтера [123] для міокарда. Ці моделі мають деякі недоліки через їх використання лише для окремих ділянок провідної системи серця, а також наявності великої кількості змінних у рівняннях моделей.

Проведений огляд літературних джерел свідчить, що незважаючи на грунтовно розроблені математичні моделі і проведений на їх основі кількісний аналіз, потребують розробки методи якісного аналізу систем електрофізіолоії – зокрема задачі оптимізації, оптимального керування та класифікації.

РОЗДІЛ 2
ВИКОРИСТАННЯ МЕТОДІВ ОБЧИСЛЮВАЛЬНОЇ МАТЕМАТИКИ СТОСОВНО ВИРІШЕННЯ ПРОБЛЕМ ДОСЛІДЖЕННЯ СИСТЕМИ ЕЛЕКТРИЧНОЇ АКТИВНОСТІ НЕРВОВИХ КЛІТИН

Чимало патологічних процесів, які відбуваються в живому організмі, можуть бути описані на клітинному і субклітинному рівні на основі моделей типу Ходжкіна-Хакслі. При аналізі таких патологічних процесів досить часто є важливим не стільки чисельні значення концентрацій певних іонів в певний момент часу, скільки якісний характер зміни їх концентрацій – концентрація прагне до деякого сталого рівня, періодичні зміни і ін.

Такі задачі дослідження стійкості та оптимального керування для моделей типу Ходжкіна-Хакслі в найпростіших випадках можуть бути розв’язані за допомогою методів теорії динамічних систем і обчислювальної математики.

У розділі пропонується алгоритм розрахунку експонент Ляпунова, що можуть бути використані для якісного аналізу траєкторій моделі, включаючи стійкість або нестійкість розв’язків, дослідження детермінованого хаосу.

У випадку, коли небажаним є виникнення періодичного розв’язку за рахунок зміни параметрів нелінійної системи (біфуркація розв’язків), як у випадку хвороби Паркінсона і інших так званих «динамічних» хвороб [159], запропоновано алгоритм для оптимальної стабілізації моделі Ходжкіна-Хакслі за рахунок прикладеного струму.

2.1 Модифікація й спеціалізація існуючого методу обчислення експонент Ляпунова з метою дослідження стійкості з урахованням особливостей математичної системи Ходжкіна-Хакслі. Створення програмного засобу комп'ютерної реалізації методу

Розглядається модель Ходжкіна-Хакслі з використанням параметрів на

прикладі електричної активності гігантського аксона кальмара (2.1) – (2.4).

$$\frac{dV}{dt} = -g_K n^4 (V - V_K) - g_{Na} m^3 h (V - V_{Na}) - g_L (V - V_L) + I_{app}, \tag{2.1}$$

$$\frac{dm}{dt} = (1-m)0.1\frac{25-V}{\exp^{\frac{25-V}{10}} - 1} - m4\exp^{-\frac{V}{18}}, \tag{2.2}$$

$$\frac{dn}{dt} = (1-n)0.1\frac{\frac{10-V}{10}}{\exp^{\frac{10-V}{10}} - 1} - n0.125\exp^{-\frac{V}{80}}, \tag{2.3}$$

$$\frac{dh}{dt} = 0.07\exp^{-\frac{V}{20}}(1-h) - \frac{h}{1+\exp^{\frac{30-V}{10}}}. \tag{2.4}$$

Існуючий алгоритм визначення експонент Ляпунова для диференціальних рівнянь грунтується на наступному понятті.

Дві траєкторії у фазовому просторі $x(t) = f^t(x_0)$ і $x(t) + \delta x(t) = f^t(x_0 + \delta x_0)$, які розташовані дуже близько, віддаляються одна від одної експоненціально з часом. Середня швидкість розходження цих траєкторій називається експонентою Ляпунова λ і визначається із співвідношення $\|\delta x(t)\| \approx e^{\lambda t}\|\delta x_0\|$ як

$$\lambda = \lim_{\substack{t\to\infty \\ \|\delta x_0\|\to 0}} \frac{1}{t}\ln\frac{\|\delta x(t)\|}{\|\delta x_0\|} \tag{2.5}$$

Визначення експонент Ляпунова системи диференціальних рівнянь в даній роботі грунтується на існуючій методиці, запропонованій в роботах [125 – 127].

Експоненти Ляпунова визначаються переходом вздовж головної осі з центру нескінченномалої сфери. Центр сфери отримується на основі нелінійних диференціальних рівнянь при певних початкових умовах. Траєкторії точок на поверхні сфери визначаються на основі лінеаризованих диференціальних рівнянь в точках нескінченно мало віддалених від центра сфери. Головна вісь

визначається лінеаризованими рівняннями і набором ортонормованих векторів, прикріплених до центру сфери. Для побудови ортонормованого базису використовується метод Грама-Шмідта.

Лінеаризацію системи нелінійних диференціальних рівнянь (2.1) – (2.4) здійснено в околі певного стаціонарного стану (V^*, m^*, n^*, h^*).

Було здійснено програмну реалізацію алгоритму розрахунку експонент Ляпунова. Програмне середовище, запропоноване в даній роботі, реалізоване у вигляді пакету Java-класів. В нього входять такі пакети і класи.

Пакет fde призначений для отримання чисельного розв'язку диференціальних рівнянь. Пакет graph містить класи, які призначені для графічної візуалізації розв'язків рівнянь. В пакеті Hodgkin_huxley містяться класи з описом системи описані Ходжкіном-Хакслі і алгоритмом розрахунку експонент Ляпунова. Сюди входять класи:

- Hodgkin_HuxleyLiapunovExponentsSystem – описує праві частини диференціальних рівнянь системи Ходжкіна-Хакслі і алгоритм розрахунку експонент Ляпунова;
- Hodgkin_HuxleyLiapunov Exponents System Graph – використовується для побудови графіків розв'язків рівнянь;
- Hodgkin_HuxleyLiapunov Exponents SystemGraph Menu – клас, який описує головне меню програми;
- Hodgkin_Huxley Liapunov Exponents System Input Data Frame – клас-фрейм для вводу початкових параметрів моделі.

Експоненти Ляпунова розраховуються в ході виконання циклу

for (double i = x0; i<= x1; i + = hmax) {....}

Тут i – лічильник цикла, x0 – початок інтервалу інтегрування, x1 – кінець інтервалу інтегрування.

На кожній ітерації відбувається побудова отронормованого базису методом Грама-Шмідта. Для цього здійснюється побудова і нормалізація першого вектору базиса:

```
znorm [1] = 0.;
for (int j = 1; j <= n_nonlinear_system; j + +)
        {znorm[1] = znorm [1] + Math.pow (y [n_nonlinear_system * j +1], 2);}
znorm [1] = Math.sqrt (znorm [1]);
for (int j = 1; j <= n_nonlinear_system; j + +)
      {y[n_nonlinear_system * j +1] = y [n_nonlinear_system * j +1] / znorm[1];}
```

І на його основі побудова решти векторів базису, яка здійснюється під час виконання цикла:

```
for (int j = 2; j <= n_nonlinear_system; j + +) {....}
```

На кожній ітерації відбуваються такі дії:

- Визначення коефіцієнтів ренормалізаціїГрама-Шмідта

```
for (int k = 1; k <= j-1; k + +)
                    {gsc[k] = 0.;
for (int l = 1; l <= n_nonlinear_system; l + +)
                     {gsc[k] = gsc [k] + y [n_nonlinear_system * l + j]
                      * y [n_nonlinear_system * l +k]; }}
```

- побудова вектора

```
for (int k = 1; k <= n_nonlinear_system; k + +)
                {for(int l = 1; l <= j-1; l + +) {
y[n_nonlinear_system*k+j]=y[n_nonlinear_system*k+j]-
gsc[l]*y[n_nonlinear_system*k+l]; }}
```

- визначення норми вектора

```
znorm [j] = 0.;
for (int k = 1; k <= n_nonlinear_system; k + +) {
znorm[j]=znorm[j]+Math.pow(y[n_nonlinear_system*k+j],2); }
znorm[j]=Math.sqrt(znorm[j]);
```

- нормалізація вектора

```
for (int k = 1; k <= n_nonlinear_system; k + +)
                    {if(znorm [j]! = 0.)
```

```
{y[n_nonlinear_system * k + j] =
y [n_nonlinear_system * k + j] / znorm [j];}
elseSystem.out.println («Null exception»);}}
```

На наступному кроці виконання алгоритма відбувається побудова експонент Ляпунова:

```
for (int k = 1; k <= n_nonlinear_system; k + +)
{cum[k] = cum [ k] + Math.log (znorm [k]) / Math.log (2.);}
```

Їх нормалізація і виведення на кожній ітерації:

```
if ((Math.round (i / hmax))% io == 0)
{for(int k = 1; k <= n_nonlinear_system; k + +)
{System.out.println(«x =» + x + ",
Liapunov exponent =" + cum [k] / x);}}}
```

Опис правих частин диференціальних рівнянь системи Ходжкіна-Хакслі і правих частин лінеаризованих диференціальних рівнянь системи Ходжкіна-Хакслі відбувається у функції fcn ().

Для обчислення експонент Ляпунова були використані параметри системи (2.1) – (2.4). Вікно введення параметрів показано на рис.2.1. В результаті виконання програми отримані розв’язки системи (2.1) – (2.4) рис.2.2.

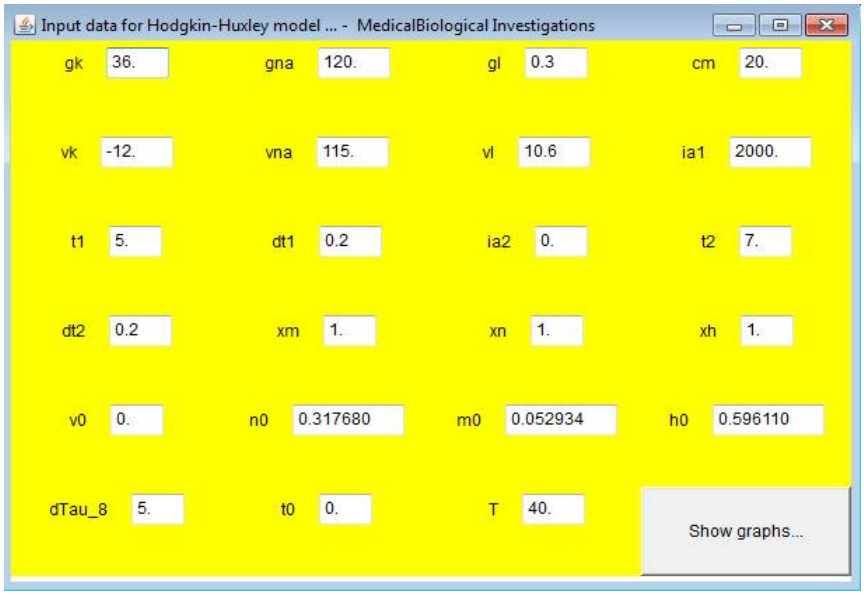


Рис. 2.1. Вікно введення параметрів системи Ходжкіна-Хакслі

Експоненти Ляпунова є фундаментальною характеристикою динамічної системи являється фундаментальною характеристикою динамичної системи, оскільки а саме на них грунтується найзагальніш означення хаосу [128]. А саме, динамічна система є хаотичною, якщо її атрактор має принаймні одну додатню експоненту Ляпунова.

На рисунку 2.2 для прикладеного струму у вигляді імпульсного сигналу отримано траєкторії моделі Ходжкіна-Хакслі. Чисельний експеримент виявив наявність додатньої експоненти Ляпунова λ_1, що свідчить про хаотичний характер траєкторії системи Ходжкіна-Хакслі у цьому випадку.

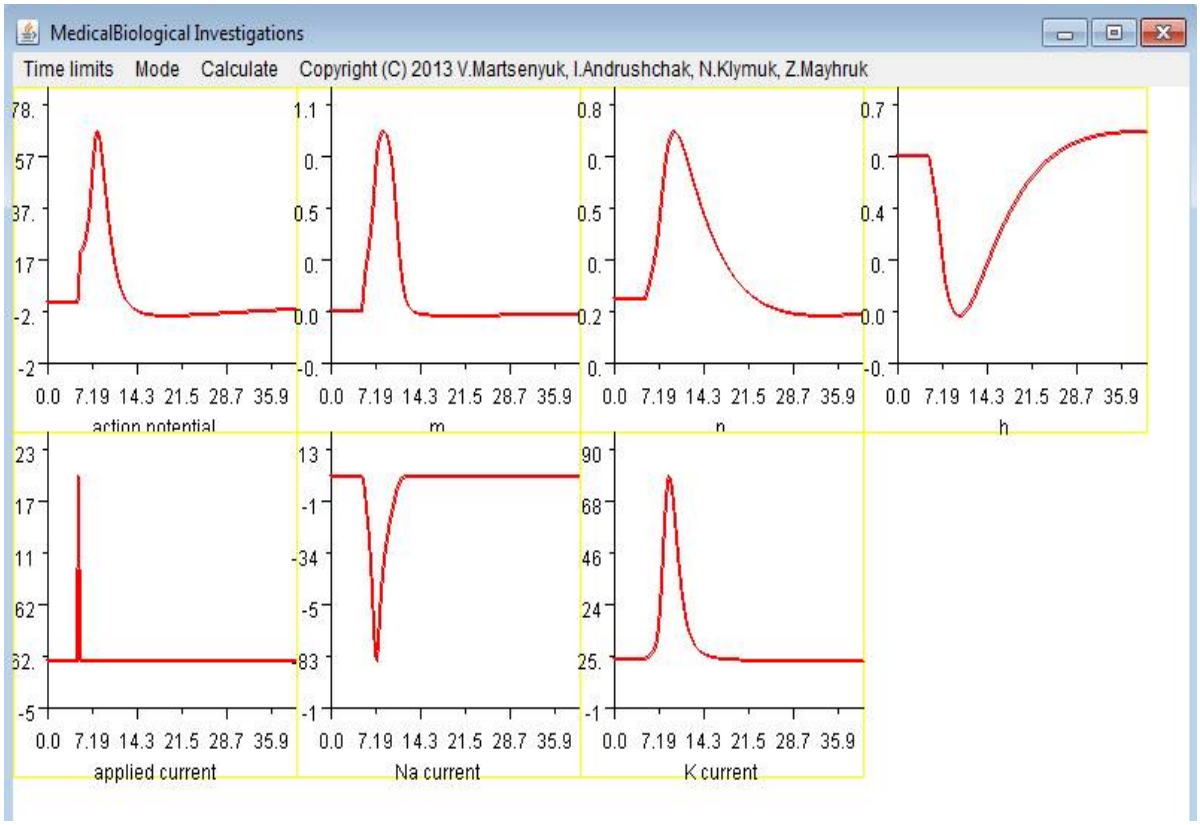


Рис. 2.2. Розв’язки системи Ходжкіна-Хакслі для параметрів рис. 2.1.

Створена у роботі програмна реалізація методу розрахунку експонент Ляпунова може бути використана у веб-інтегрованих проектах щодо фізіологічного моделювання.

Отже, запропоновано та програмно реалізовано метод обчислення експонент Ляпунова для моделі Ходжкіна-Хакслі. З цією метою було здійснено лінеаризацію моделі. Метод полягає в порівнянні на кожному кроці розходження розв'язків моделі Ходжкіна-Хакслі з її лінеаризованим наближенням.

Таким чином створений програмний засіб може бути застосований для дослідження нелінійної динаміки моделей типу Ходжкіна-Хакслі, включаючи стійкість та хаотичну поведінку траєкторій

2.2 Розробка методу оптимізації математичної системи Ходжкіна-Хакслі для задачі оптимального керування біфуркацією при зміні прикладеного струму. Отримання необхідних умов оптимальності

З метою розв'язання задачі розвитку теорії математичного моделювання електрофізіологічних процесів збудливості клітин розроблено новий алгоритм керування нелінійною динамікою системи Ходжкіна-Хакслі, представлений далі. Таке керування пов'язане з поняттям біфуркації.

Під біфуркацією мають на увазі якісні зміни у структурі розв'язків динамічної системи внаслідок варіації параметрів системи. Керування біфуркацією полягає в побудові керування, що змінює біфуркаційні властивості заданої нелінійної системи з метою уникнення небажаної нестійкості або отримання потрібної динамічної поведінки [133].

Отже, розглянемо систему керування:

$$\frac{dV}{dt} = -g_K n^4 (V - V_K) - g_{Na} m^3 h (V - V_{Na}) - g_L (V - V_L) + I_{app} u, \tag{2.6}$$

$$\frac{dm}{dt} = (1-m)0.1\frac{25-V}{\exp^{\frac{25-V}{10}} - 1} - m4\exp^{-\frac{V}{18}}, \tag{2.7}$$

$$\frac{dn}{dt} = (1-n)0.1\frac{\frac{10-V}{10}}{\exp^{\frac{10-V}{10}}-1} - n0.125\exp^{-\frac{V}{80}}, \tag{2.8}$$

$$\frac{dh}{dt} = 0.07\exp^{-\frac{V}{20}}(1-h) - \frac{h}{1+\exp^{\frac{30-V}{10}}}. \tag{2.9}$$

У задачах оптимального керування електрофізіології розглядаємо таку множину керувань:

$$U = \{u(t) : a \le u(t) \le b, \quad t_0 \le t \le t_f, \quad u(t) - \textit{вимірна}\}.$$

Тут $a, b, t_f > 0$. В якості керуючого впливу переважно виступає прикладений струм.

Запропонований тут метод розв'язку задачі оптимального керування використовує наступний загальний підхід у випадку стану системи $x(t) \in R^n$ при заданому керуванні $u \in U$, що визначається системою звичайних диференціальних рівнянь:

$$\begin{aligned} &\frac{dx(t)}{dt} = f(t, x, u), \\ &x(t_0) = x_0 \end{aligned} \tag{2.10}$$

де $f : R \times R^n \times R \to R^n$ є неперервною і має неперервні перші частинні похідні відносно x та u.

Оскільки припускається, що $u(t)$ є вимірною та обмеженою, то права частина системи (2.10) є неперервною відносно x і лише вимірною відносно t для фіксованого x. Отже, розв'язки (2.10) є абсолютно неперервними функціями, що задовольняють (2.10) майже всюди. При таких умовах існування розв'язку (2.10) $x(t,u)$ доведене в роботі [129].

Задача оптимального керування містить критерій якості $J[u]$ вигляду:

$$J[u] = \int_{t_0}^{t_f} L(t,x,u)dt + \phi(x(t_f)),$$

де L – задана дійснозначна функція і ϕ – неперервнодиференційовна дійснозначна функція. Метою є знаходження керування $u^* \in U$, такого що

$$J[u^*] = \inf_{u \in U} J[u]. \qquad (2.11)$$

Після того, як описано модель та визначено критерій якості в теорії оптимального керування ставлять низку задач [130]:

- доведення існування оптимального керування;
- опис побудови оптимального керування;
- доведення єдиності оптимального керування;
- чисельне обчислення оптимального керування;
- дослідження залежності оптимального керування від параметрів моделі.

Достатні умови існування оптимального керування для задачі (2.10) – (2.11) без термінальної складової в критерії якості наведено в роботах [131].

Твердження 2.1. [131] Розглядається задача оптимального керування (2.10) – (2.11) на фіксованому інтервалі $[t_0, t_f]$. Припустимо, що:

1) існує стала $M > 0$ така, що $\|x(t,u)\| \le M$ для всіх $u \in U$ та $t_0 \le t \le t_f$;

2) L є напівнеперервною знизу;

3) множина $D^+ = \{(y^0, y) : \exists v \in U,\ y = f(t,x,v),\ y^0 \ge L(t,x,v)\}$ є опуклою для $(t,x) \in [t_0, t_f] \times \{|x| \le M\}$.

Тоді існує оптимальне керування $u^* \in U$.

Опис побудови оптимального керування для задачі (2.10) – (2.11) дає принцип максимуму Понтрягіна з термінальною складовою [132].

Твердження 2.2. [132] Нехай $u^* \in U$ – оптимальне керування в задачі (2.10) – (2.11). Тоді існує спряжена функція $\lambda : R \to R^n$ така, що $x(t,u^*)$, u^*, λ задовольняють систему:

$$\begin{aligned} &\frac{dx(t)}{dt} = f(t,x,u^*), \\ &x(t_0) = x_0 \end{aligned} \tag{2.12}$$

та спряжену систему:

$$\begin{aligned} &\frac{d\lambda(t)}{dt} = -\frac{\partial H}{\partial x} = -L_x(t,x,u^*) - \lambda^T f_x(t,x,u^*), \\ &\lambda(t_f) = \phi'(x(t_f)), \quad \textit{умова трансвертальності} \end{aligned} \tag{2.13}$$

де функція Гамільтона-Понтрягіна H задається як:

$$H(t,x,u) = L(t,x,u) + \lambda^T f(t,x,u) \tag{2.14}$$

На основі Твердження 2.1. ми бачимо, що оптимальне керування в задачі (2.6) – (2.11) існує, оскільки підінтегральний вираз в критерії якості є опуклою функцією а траєкторія системи належить простору L^∞.

Застосуємо Твердження 2.2. для отримання необхідних умов оптимальності. Функція Гамільтона-Понтрягіна має вигляд:

$$H = V^2 + \lambda_1(-g_K n^4 (V - V_K) - g_{Na} m^3 h (V - V_{Na}) - g_L (V - V_L) + I_{app} u) +$$
$$+ \lambda_2((1-m)0.1\frac{25-V}{\exp^{\frac{25-V}{10}} - 1} - m4\exp^{-\frac{V}{18}}) +$$
$$+ \lambda_3((1-n)0.1\frac{\frac{10-V}{10}}{\exp^{\frac{10-V}{10}} - 1} - n0.125\exp^{-\frac{V}{80}}) +$$
$$+ \lambda_4(0.07\exp^{-\frac{V}{20}}(1-h) - \frac{h}{1+\exp^{\frac{30-V}{10}}})$$

Отже, з Твердження 2.2. маємо спряжену систему:

$$\frac{d\lambda_1}{dt} = -\frac{\partial H}{\partial V} = -2V + \lambda_1(g_K n^4 + g_{Na} m^3 h + g_L) +$$
$$+ \lambda_2(-(1-m)0.1\frac{1-\exp^{\frac{25-V}{10}} + \frac{25-V}{10}\exp^{\frac{25-V}{10}}}{(\exp^{\frac{25-V}{10}} - 1)^2} - m\frac{2}{9}\exp^{-\frac{V}{18}}) +$$
$$+ \lambda_3(-(1-n)0.1\frac{1-\exp^{\frac{10-V}{10}} + \frac{10-V}{10}\exp^{\frac{10-V}{10}}}{(\exp^{\frac{10-V}{10}} - 1)^2} - n\frac{0.125}{80}\exp^{-\frac{V}{80}}) +$$
$$+ \lambda_4(\frac{0.07}{20}\exp^{-\frac{V}{20}}(1-h) + \frac{h}{10*(1+\exp^{\frac{30-V}{10}})^2}\exp^{\frac{30-V}{10}}) \qquad (2.15)$$

$$\frac{d\lambda_2}{dt} = -\frac{\partial H}{\partial m} = \lambda_1 3 g_{Na} m^2 h (V - V_{Na}) + \lambda_2(0.1\frac{25-V}{\exp^{\frac{25-V}{10}} - 1} + 4\exp^{-\frac{V}{18}}),$$

$$\frac{d\lambda_3}{dt} = -\frac{\partial H}{\partial n} = \lambda_1 g_K 4n^3 (V - V_K) + \lambda_3(0.1\frac{\frac{10-V}{10}}{\exp^{\frac{10-V}{10}} - 1} + 0.125\exp^{-\frac{V}{80}}),$$

$$\frac{d\lambda_4}{dt} = -\frac{\partial H}{\partial h} = \lambda_1 g_{Na} m^3 (V - V_{Na}) + \lambda_4 (0.07 \exp^{-\frac{V}{20}} + \frac{1}{1 + \exp^{\frac{30-V}{10}}}).$$

Згідно принципупу максимуму Понтрягіна маємо

$$u^*(t) = \operatorname{sgn} \lambda_1(t).$$

Отже, виходячи з Твердження 2.1. оптимальне керування в задачі (2.6) – (2.11) може бути побудоване в результаті розв'язку такої крайової задачі:

$$\frac{dV}{dt} = -g_K n^4 (V - V_K) - g_{Na} m^3 h (V - V_{Na}) - g_L (V - V_L) + I_{app} \operatorname{sgn} \lambda_1(t),$$

$$\frac{dm}{dt} = (1-m)0.1\frac{25 - V}{\exp^{\frac{25-V}{10}} - 1} - m4\exp^{-\frac{V}{18}},$$

$$\frac{dn}{dt} = (1-n)0.1\frac{\frac{10 - V}{10}}{\exp^{\frac{10-V}{10}} - 1} - n0.125\exp^{-\frac{V}{80}}, \qquad (2.16)$$

$$\frac{dh}{dt} = 0.07\exp^{-\frac{V}{20}}(1-h) - \frac{h}{1 + \exp^{\frac{30-V}{10}}}.$$

$$V(0) = V_0,\ m(0) = m_0,\ n(0) = n_0,\ h(0) = h_0,$$

$$\frac{d\lambda_1}{dt} = -\frac{\partial H}{\partial V} = -2V + \lambda_1(g_K n^4 + g_{Na} m^3 h + g_L) +$$

$$+ \lambda_2(-(1-m)0.1\frac{1-\exp^{\frac{25-V}{10}} + \frac{25-V}{10}\exp^{\frac{25-V}{10}}}{(\exp^{\frac{25-V}{10}}-1)^2} - m\frac{2}{9}\exp^{-\frac{V}{18}}) +$$

$$+ \lambda_3(-(1-n)0.1\frac{1-\exp^{\frac{10-V}{10}} + \frac{10-V}{10}\exp^{\frac{10-V}{10}}}{(\exp^{\frac{10-V}{10}}-1)^2} - n\frac{0.125}{80}\exp^{-\frac{V}{80}}) +$$

$$+ \lambda_4(\frac{0.07}{20}\exp^{-\frac{V}{20}}(1-h) + \frac{h}{10*(1+\exp^{\frac{30-V}{10}})^2}\exp^{\frac{30-V}{10}})$$

,

$$\frac{d\lambda_2}{dt} = -\frac{\partial H}{\partial m} = \lambda_1 3 g_{Na} m^2 h(V - V_{Na}) + \lambda_2(0.1\frac{25-V}{\exp^{\frac{25-V}{10}}-1} + 4\exp^{-\frac{V}{18}}),$$

$$\frac{d\lambda_3}{dt} = -\frac{\partial H}{\partial n} = \lambda_1 g_K 4n^3(V - V_K) + \lambda_3(0.1\frac{\frac{10-V}{10}}{\exp^{\frac{10-V}{10}}-1} + 0.125\exp^{-\frac{V}{80}}),$$

$$\frac{d\lambda_4}{dt} = -\frac{\partial H}{\partial h} = \lambda_1 g_{Na} m^3(V - V_{Na}) + \lambda_4(0.07\exp^{-\frac{V}{20}} + \frac{1}{1+\exp^{\frac{30-V}{10}}}),$$

$$\lambda_k(t_f) = 0, \quad k = \overline{1,4}.$$

Показано, що для досить малого значення t_f розв'язок системи (2.16) є єдиним.

Припустимо навпаки, що існують два розв'язки (2.13), а саме:

$$X^* = (V^*, m^*, n^*, h^*, \lambda_1^*, \lambda_2^*, \lambda_3^*, \lambda_4^*) \text{ і}$$

$$X^{**} = (V^{**}, m^{**}, n^{**}, h^{**}, \lambda_1^{**}, \lambda_2^{**}, \lambda_3^{**}, \lambda_4^{**}).$$

Праві частини системи (2.16) є Ліпшицевими функціями аргументів $V, m, n, h, \lambda_1, \lambda_2, \lambda_3, \lambda_4$. Звідси існує стала $C > 0$ така, що:

$$\left\|X^*(t)-X^{**}(t)\right\| \le \int_0^{t_f} C(\left\|X^*(s)-X^{**}(s)\right\|)ds. \tag{2.17}$$

Застосовуючи до (2.17) теорему про середнє значення, маємо, що існує момент часу $\xi : 0 \le \xi \le t_f$ такий, що:

$$\left\|X^*(t)-X^{**}(t)\right\| \le t_f C(\left\|X^*(\xi)-X^{**}(\xi)\right\|).$$

при всіх $t \in [0,t_f]$. Якщо ж виберемо t_f таким, що $t_f < \frac{1}{C}$, то отримуємо суперечність.

Отже, було розроблено систему керування на основі моделі Ходжкіна-Хакслі за рахунок прикладеного струму. При цьому для визначеного критерію якості та множини керування обгрунтовано існування та єдність оптимального керування, а також запропоновано опис побудови оптимального керування виходячи з необхідних умов оптимальності принципу максимуму Понтрягіна.

2.3 Модифікація прямого чисельного методу для побудови оптимального керування біфуркацією в моделі Ходжкіна-Хакслі з метою створення ефективного програмного засобу комп'ютерної реалізації

Унаслідок варіації параметрів динамічної системи можуть виникати якісні зміни у структурі розв’язків. Керування такими біфуркаціями полягає в побудові керуючого впливу, що змінює властивості заданої нелінійної системи з метою уникнення небажаної нестійкості, коливань або отримання потрібної динамічної поведінки.

Для практичної побудови оптимального керування в моделі Ходжкіна-Хакслі (2.6) – (2.9) запропоновано використання так званого прямого чисельного методу, названого так на відміну від методів, що полягають на

чисельному розв'язуванні крайових задач типу (2.16).

У загальному випадку розглядається система керування для фазових координат $x(t) \in R^n$, вектора керувань $u(t) \in R^m$ і невідомих параметрів $p \in R^{n_p}$:

$$\begin{aligned} &\frac{dx(t)}{dt} = f(t, x, u, p), \\ &x(t_0) = x_0 \end{aligned} \tag{2.18}$$

Накладаються обмеження на стан системи, керування та параметри у вигляді рівностей:

$$c(t, x, u, p) = 0, \quad t \in [t_0, t_f], \tag{2.19}$$

де $c(t, x, u, p) \in R^{n_c}$,

у вигляді нерівностей:

$$d(t, x, u, p) \le 0, \quad t \in [t_0, t_f], \tag{2.20}$$

де $d(t, x, u, p) \in R^{n_d}$,

обмеження на стан системи в кінцевий момент часу та параметри у вигляді рівностей:

$$\psi(x(t_f) p) = 0, \tag{2.21}$$

де $\psi(x(t_f), p) \in R^{n_\psi}$,

у вигляді нерівностей:

$$\gamma(x(t_f) p) \le 0, \tag{2.22}$$

де $\gamma(x(t_f), p) \in R^{n_\gamma}$.

Задача полягає в знаходженні керування $u(t) \in R^m$ та параметрів $p \in R^{n_p}$, що мінімізують критерій якості:

$$J[u, p] = \int_{t_0}^{t_f} L(t, x, u, p)dt + \phi(x(t_f), p),$$

тобто:

$$J[u^*, p^*] = \inf_{u, p \in (16)-(19)} J[u, p]. \qquad (2.23)$$

Зауважимо, що хоча в постановці задачі (2.18) – (2.23) вважається, що t_f – фіксований, вона може бути пристосована до задачі оптимальної швидкодії. Це можна зробити нормалізуючи часову змінну t і поклавши невідомий кінцевий час як параметр.

Чисельний метод полягає в тому, що нескінченновимірна задача типу (2.18) – (2.23) зводиться до скінченновимірної задачі оптимізації.

Це досягається шляхом дискретизації часового інтервалу $t \in [t_0, t_f]$ з використанням N вузлів t_i таких, що $t_0 = t_0 < t_1 < ... < t_{N-1} = t_f$.

В кожен момент часу t_i керування є невідомим скалярним вектором $\overline{u}_i \in R^m$. На кожному відкритому інтервалі $t \in (t_i, t_{i+1})$, $i = \overline{0, N-2}$ керування наближається шляхом лінійної апроксимації:

$$u(t) = \overline{u}_i + \frac{t - t_i}{t_{i+1} - t_i}(\overline{u}_{i+1} - \overline{u}_i). \qquad (2.24)$$

Набір векторів керування у вузлах t_i формують загальний вектор:

$$\widetilde{u} = [\overline{u}_0^T, ..., \overline{u}_{N-1}^T]^T . \quad (2.25)$$

Для заданого початкового наближення $\widetilde{u}$ ми можемо проінтегрувати (2.18) на $t \in [t_0, t_f]$ і отримати траєкторію $x(t, \widetilde{u}, p)$. Таким чином нескінченновимірна задача (2.18) – (2.23) апроксимується скінченновимірною задачею нелінійного програмування відносно $\widetilde{u}, p$:

$$J(\widetilde{u}, p) = \int_{t_0}^{t_f} L(t, x(t, \widetilde{u}, p), \widetilde{u}, p)dt + \phi(x(t_f, \widetilde{u}, p), p) \to \inf_{\widetilde{u}, p}, \quad (2.26)$$

при обмеженнях:

$$\widetilde{c} = [c(t_0)^T, ..., c(t_{N-1})^T, \psi^T]^T = 0 \quad (2.27)$$

$$\widetilde{d} = [d(t_0)^T, ..., d(t_{N-1})^T, \gamma^T]^T \le 0 \quad (2.28)$$

та $u(t)$ апроксимується (2.24).

Отже, для задачі керування на основі моделі Ходжкіна-Хакслі було запропоновано прямий чисельний метод, що полягає у апроксимації нескінченновимірної задачі (2.18) – (2.23) скінченновимірною задачею нелінійного програмування.

2.3.1 Чисельне дослідження виникнення біфуркацій в системі Ходжкіна-Хакслі при біфуркаційному параметрі I_{app}. Дослідимо виникнення біфуркацій в системі Ходжкіна-Хакслі. Підхід полягає у тому, що змінюючи прикладений струм I_{app} ми керуємо формою потенціалу дії на клітинній мембрані.

Так при I_{app}=5 мА/см^2 розв'язок системи – стійкий вузол рис.2.3.

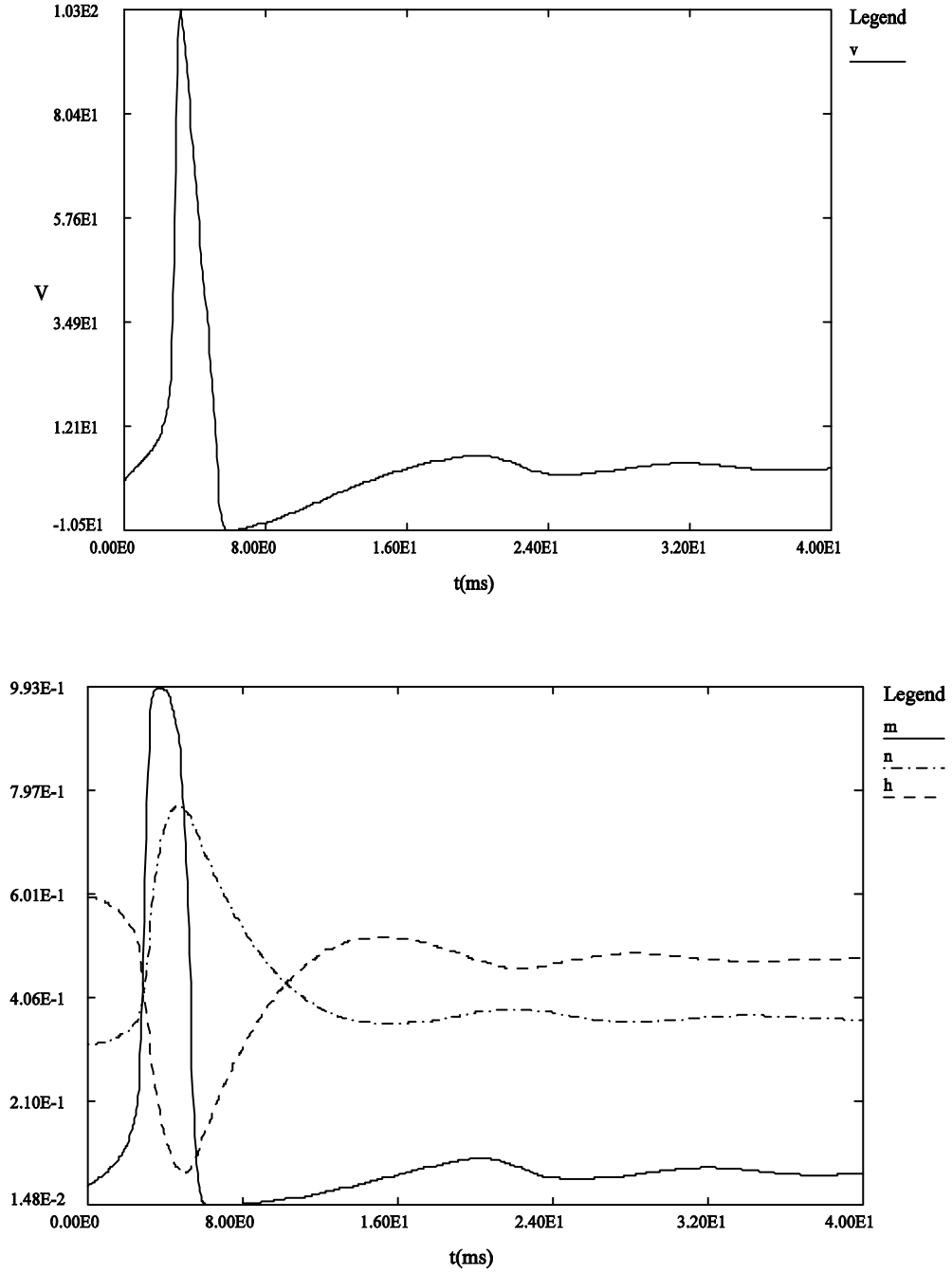


Рис. 2.3. $I_{app} = 5$ мА/см^2

При $I_{app} = 9.779638$ мА/см^2 в системі виникає біфуркація і розв'язок – граничний цикл рис.2.4.

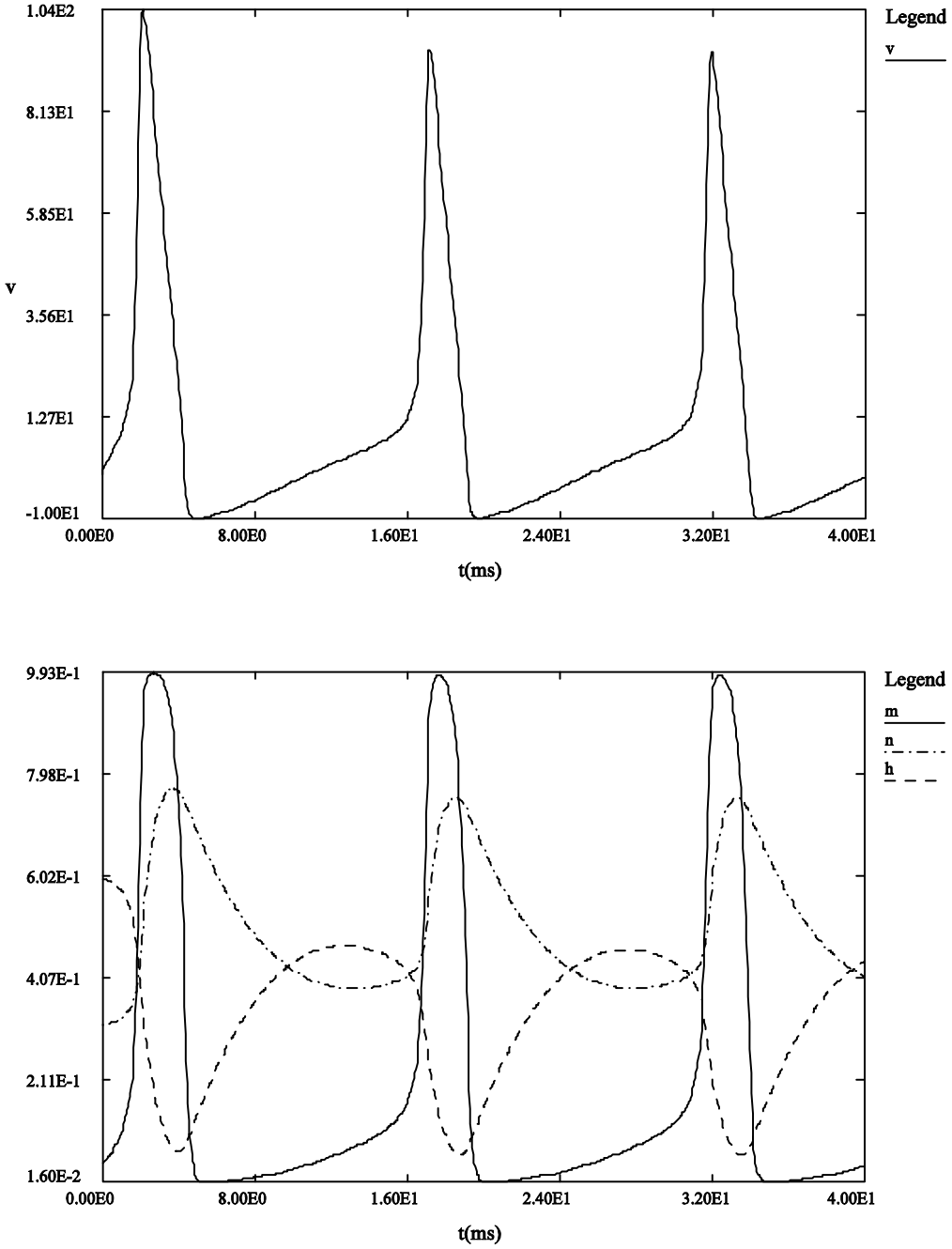


Рис. 2.4. I_{app} = 9.779638 мА/см^2

Із збільшенням I_{app} амплітуда періодичних коливань в граничних циклах спадає рис.2.5.

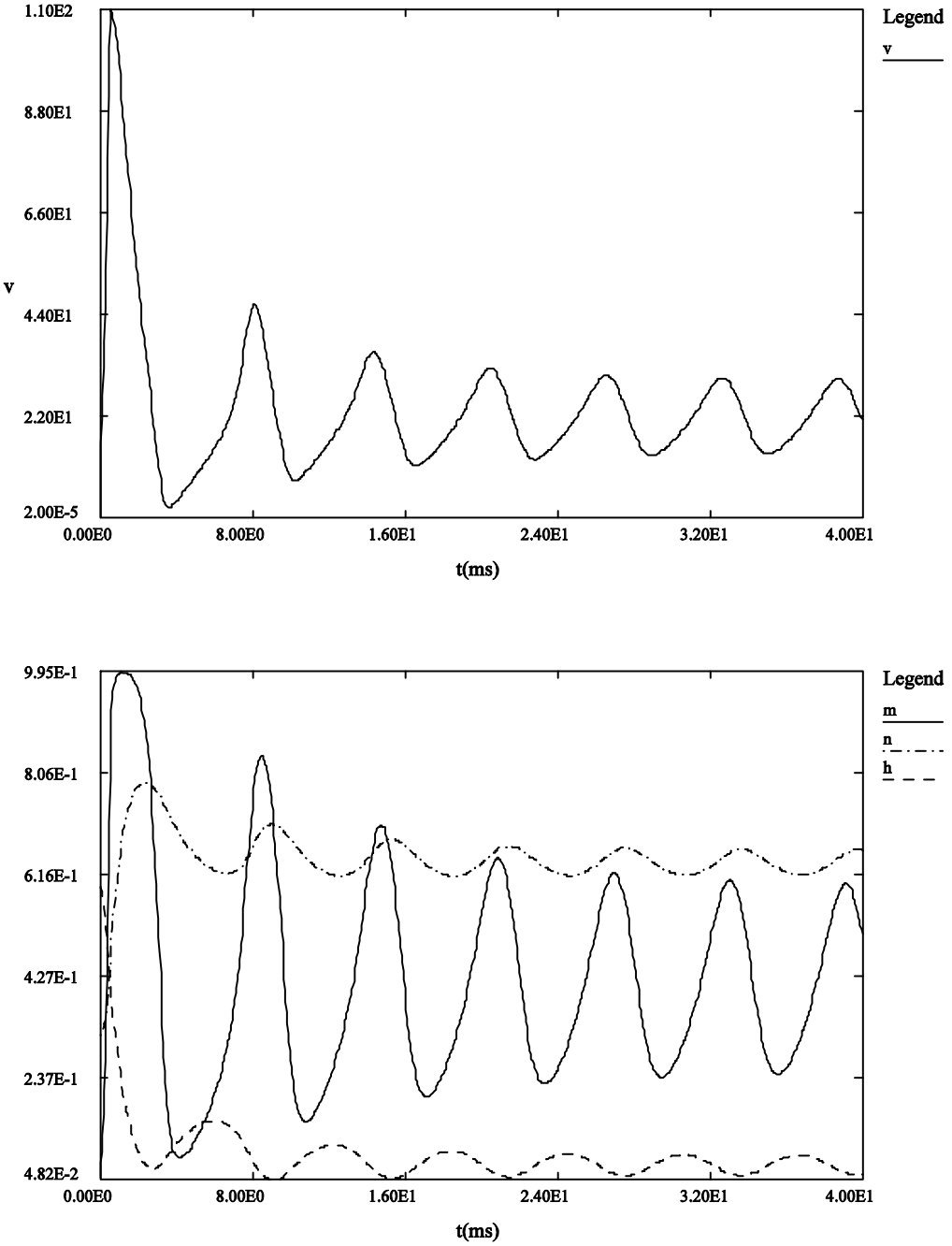


Рис.2.5. $I_{app} =$ 140 мА/см^2

Таким чином при $I_{app} = 154.526634$ мА/см^2 граничний цикл знову переходить у стійкий вузол рис.2.6, 2.7.

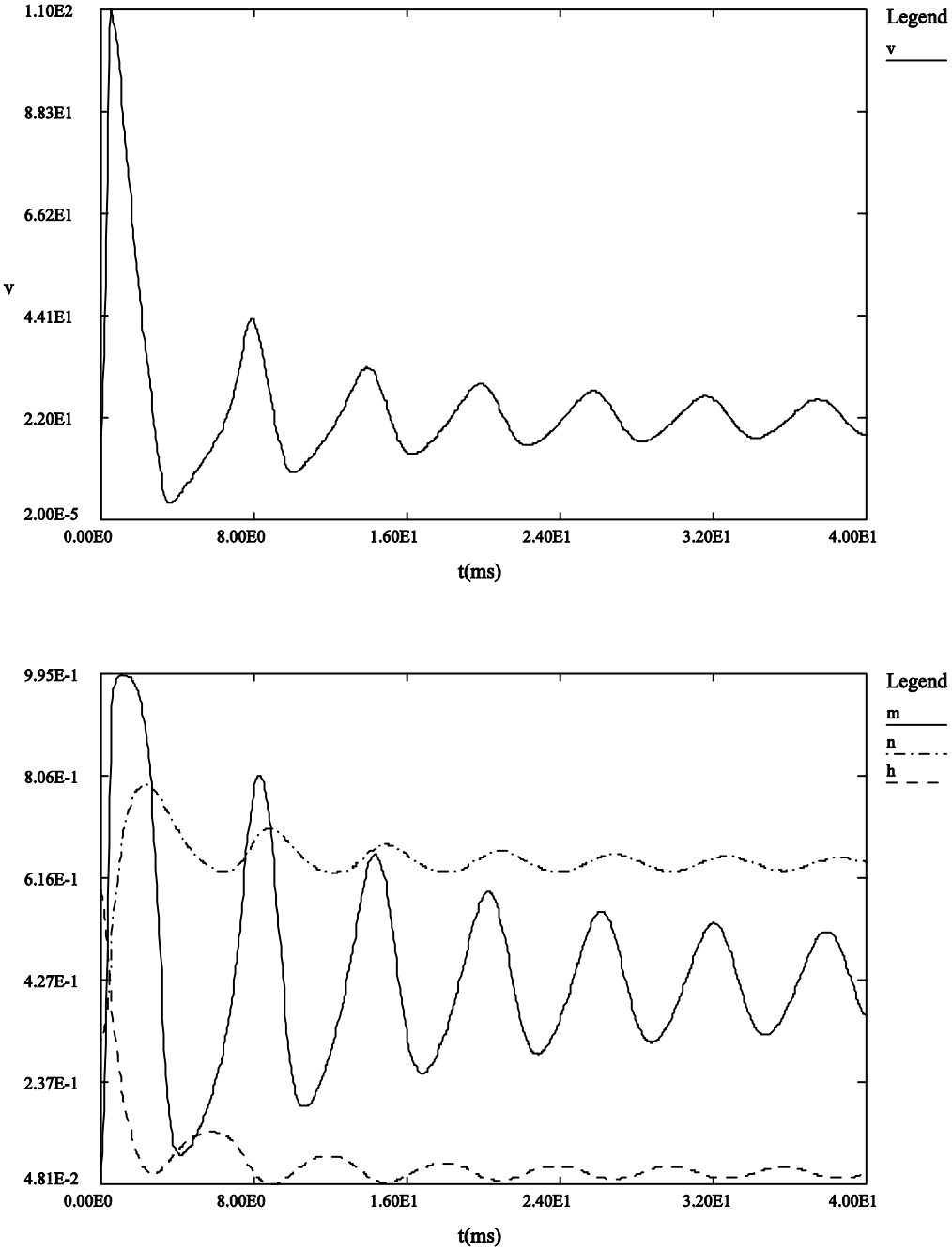


Рис. 2.6. $I_{app} = 154.526634$ мА/см^2

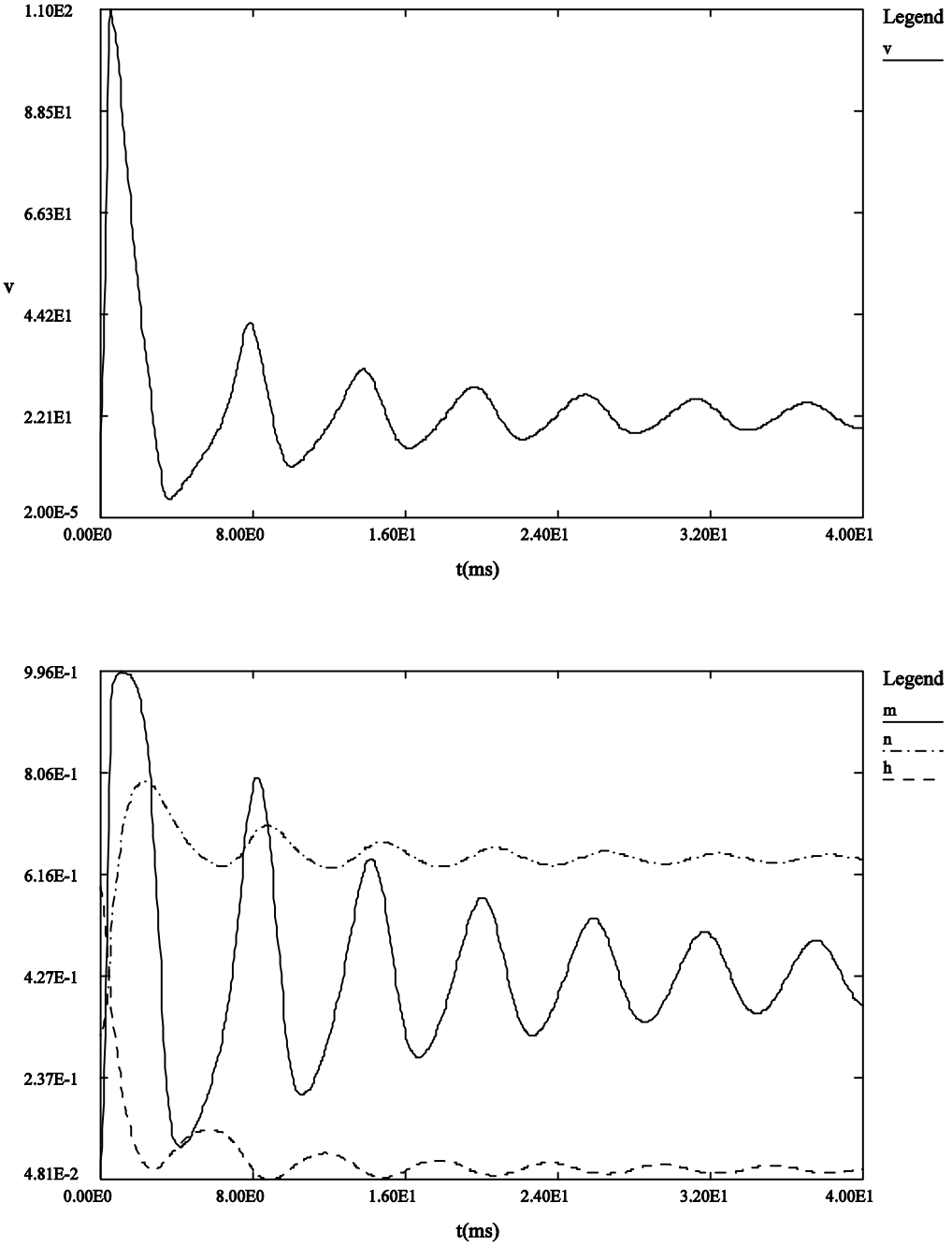


Рис. 2.7. I_{app} = 160 мА/см^2

Отже, при значеннях I_{app} = 9.779638 мА/см^2 та I_{app} = 154.526634 мА/см^2 в моделі (2.1) – (2.4) виникають біфуркації. Значний практичний інтерес має задача оптимальної стабілізації моделі (2.1) – (2.4) за рахунок вибору прикладеного струму при I_{app} = 9.779638 мА/см^2. Як показано в роботі [134], в цьому випадку виникає біфуркація Хопфа.

Отже, при зміні прикладеного струму в моделі Ходжкіна-Хакслі наступає якісна зміна траєкторій розв'язків. Зокрема, при певних значеннях I_{app} спостерігається біфуркація Хопфа.

2.3.2 Прямий метод чисельного розв'язування задачі оптимального керування, наведений вище, реалізовано в пакеті Java-класів dyn.Opt [135]. Для використання цього методу в пакеті healthinsurance запускається окремий процес у try-блоці:

```
try {
    Process p = Runtime.getRuntime().exec ("java dyn.Opt");
  }
  catch (java.io.IOException ex) {
    System.err.println("Problems invoking class dyn.Opt: "+ex);
  }
```

В якості прикладу розглядалася задача (2.22) – (2.25). Опис задачі зроблено за допомогою вхідного текстового файлу. Так змінні стану системи визначено командою:

```
state v m n h
змінна керування:
control u
константи:
real    gk gna gl gk gna ia1 xm xn xh vk vna vl I_app
кількість часових вузлів:
nodes = 1000
метод розв'язування задачі нелінійного програмування:
method = dyn_sqp
метод інтегрування системи диференціальних рівнянь:
ode = huen
файл для вихідних даних:
output_file = Hodgkin_Huxley_opt
```

точність методу:

epsilon = 1.0e-4

Система керування (2.22) – (2.25) із значеннями параметрів описується у блоці:

```
dynamic_equation:
  gk=36.
  gna=120.
  gl=0.3
 vna=115
 vk=-12
 vl=10.599
 I_app=9.779638
 ddt v = - gk*n*n*n*n*(v-vk) - gna*m*m*m*h*(v-vna)-gl*(v-vl)+I_app*u
 ddt m = (1-m)*((25-v)/10)/(exp((25-v)/10)-1)-m*4*exp(-v/18)
 ddt n = (1-n)*0.1*((10-v)/10)/(exp((10-v)/10)-1)-n*0.125*exp(-v/80)
 ddt h = 0.07*exp(-v/20)*(1-h)-(h/(1+exp((30-v)/10)))
```

Блок початкових умов (13):

```
initial_condition:
v=0.00002
 m=0.05293
 n=0.317680
 h=0.59612
```

Обмеження типу нерівності:

```
inequality_constraint:
   d = -u   # -u <= 0
   d = u-1  # u <= 1
```

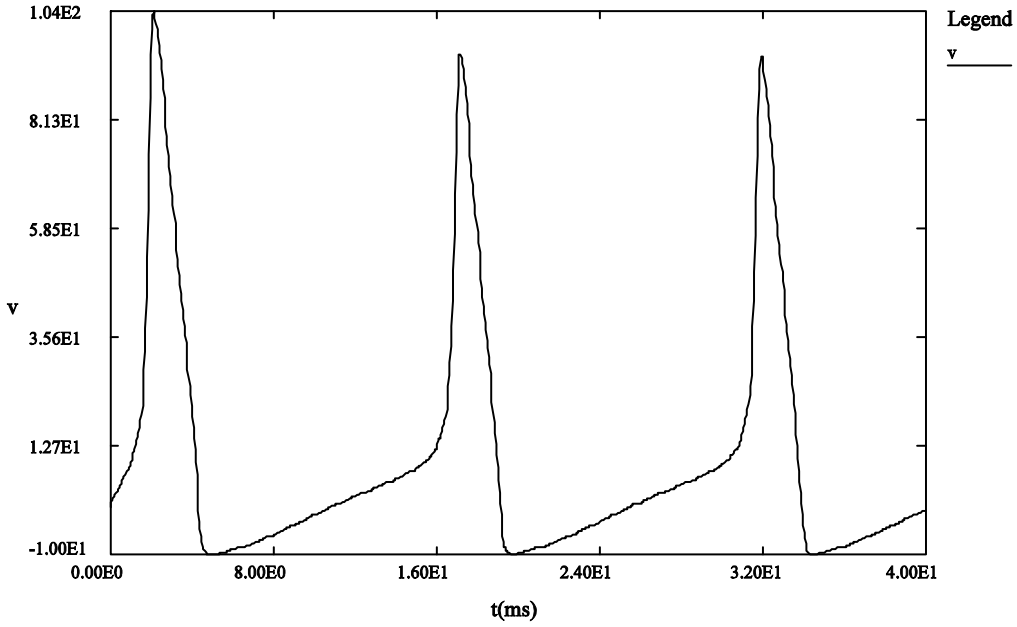
Блок критерія якості:

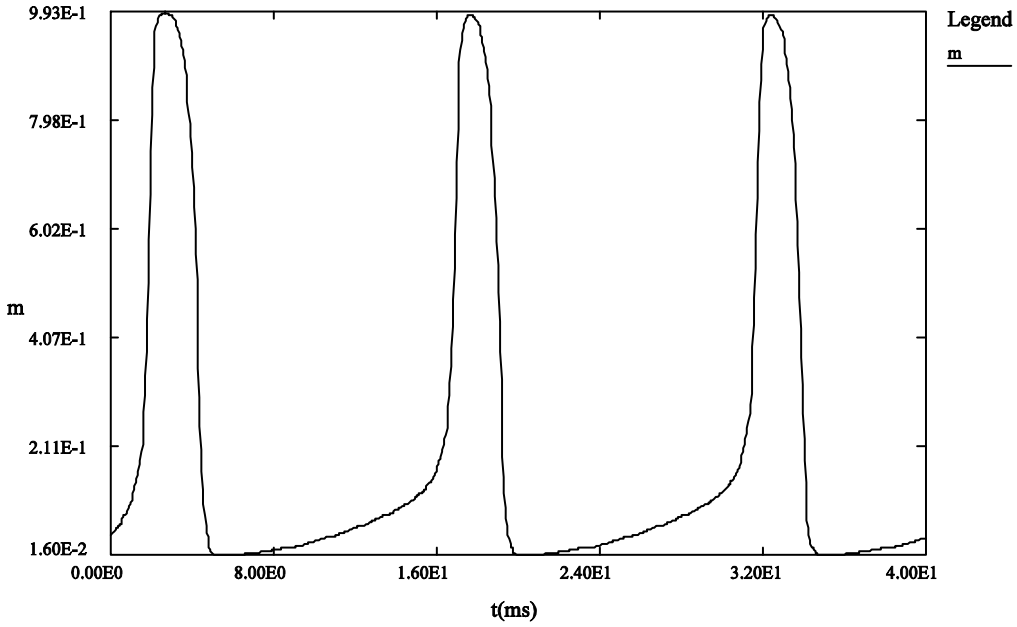
```
cost_functional:
      initial_time = 0.0
```

final_time = 40

L =v*v

Як показано в роботі [135], при значенні прикладеного струму $I_{app} = 9.779638$ в системі (2.18) – (2.121) виникає біфуркація Хопфа. На рисунку 2.8. показано результати чисельного інтегрування системи (2.18) – (2.21) при $u \equiv 0$ та $I_{app} = 9.779638$.





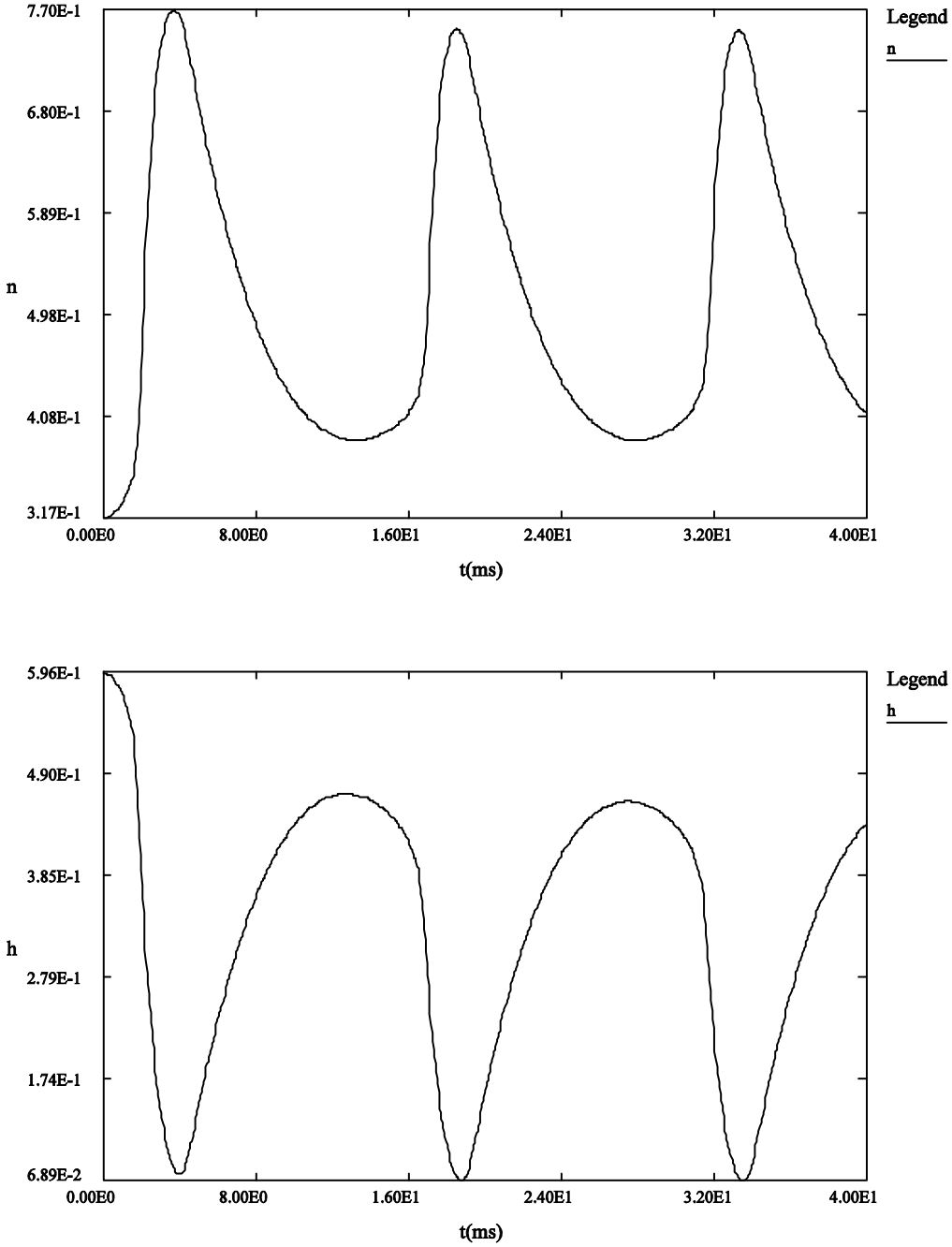


Рис. 2.8. Результати чисельного інтегрування системи

Дані щодо застосування прямого методу до системи керування (2.6) – (2.11) наведені нижче:

$ f = 1.1057781955125538E-8

$ hg_max = 1.349416312231156E-8

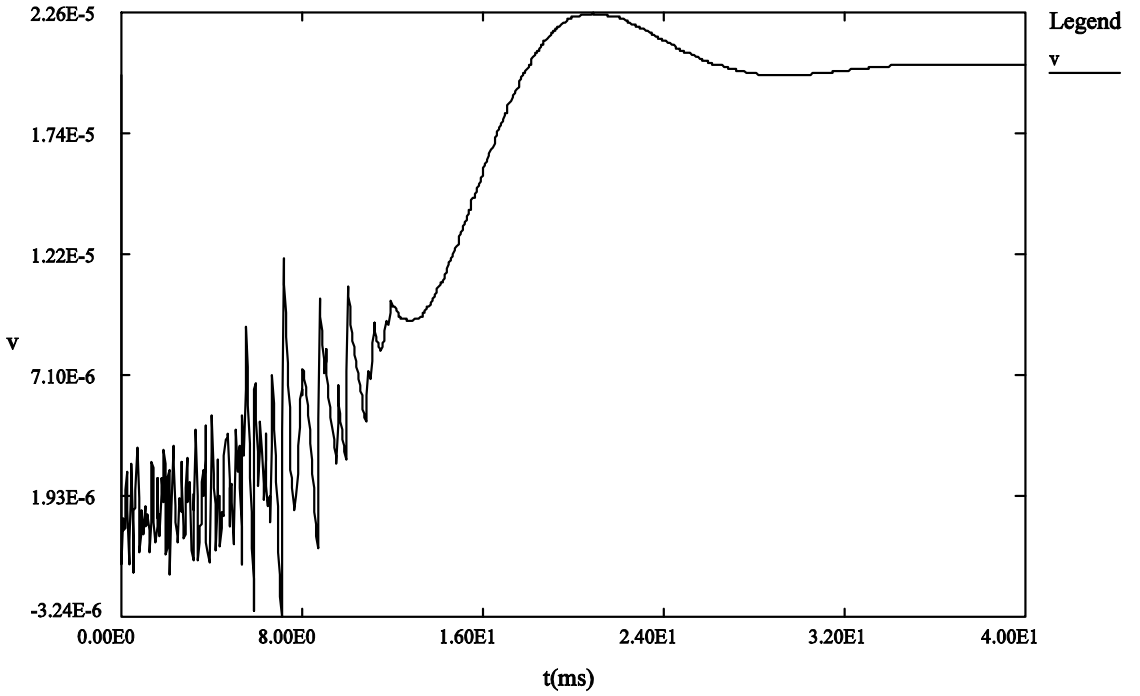
$ |dL| = 0.021991660589692354

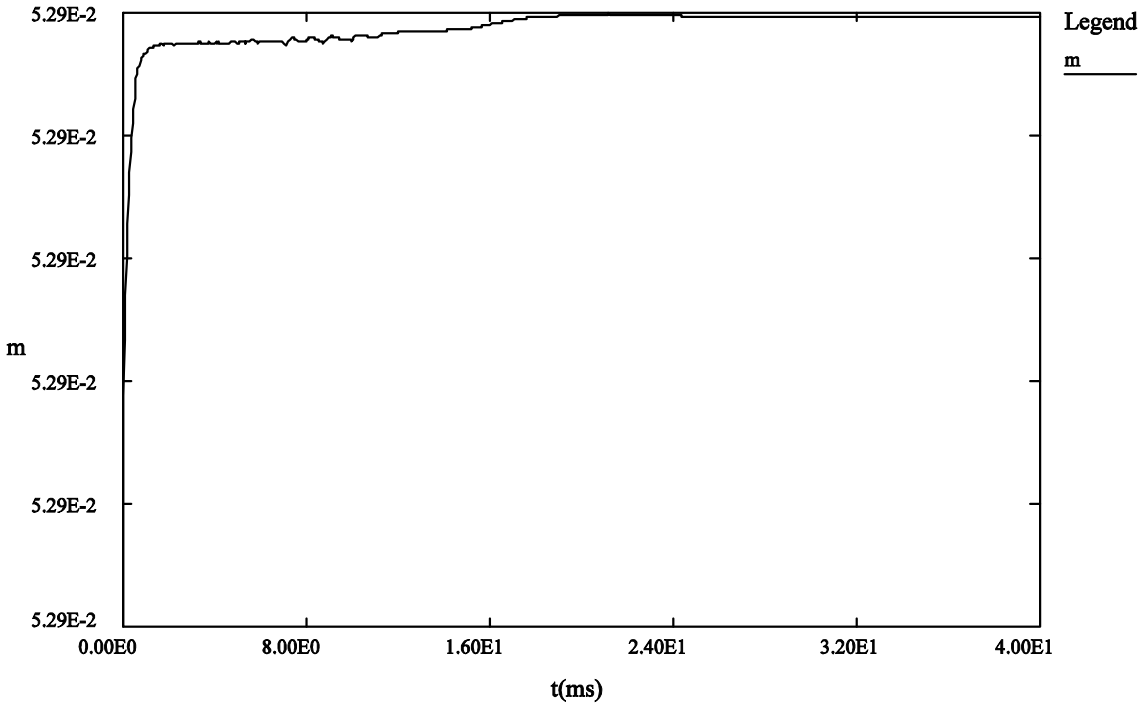
$ nfun = 309345

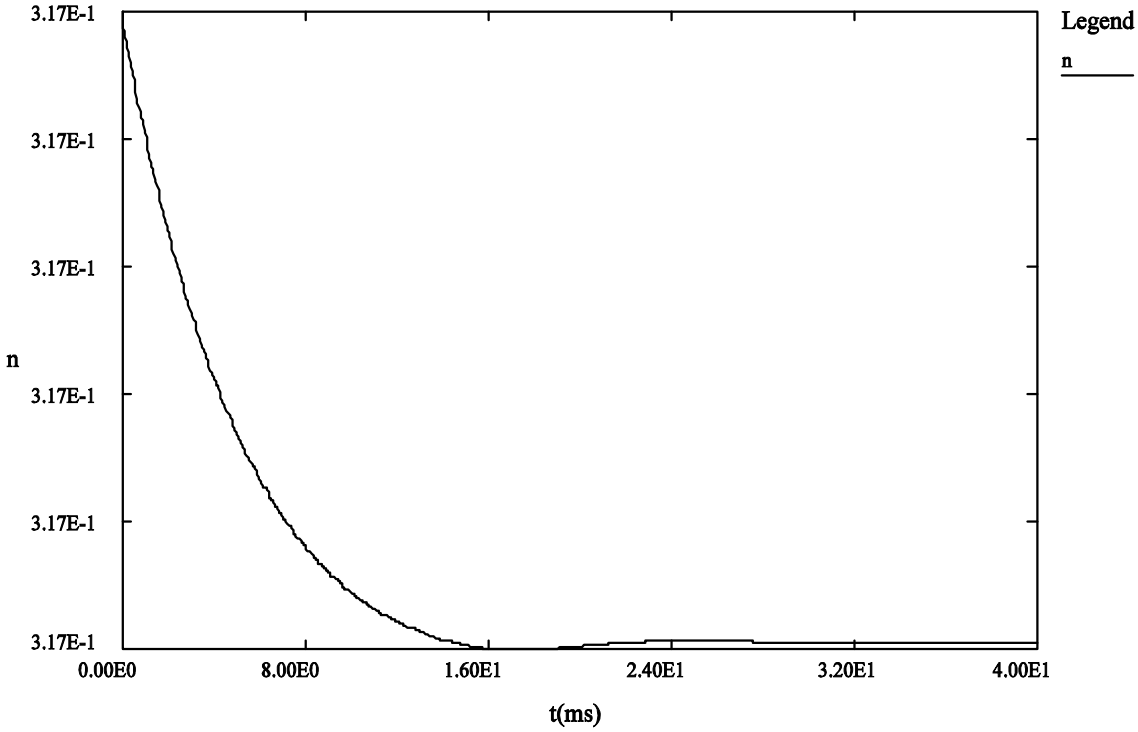
$ ngrad = 308

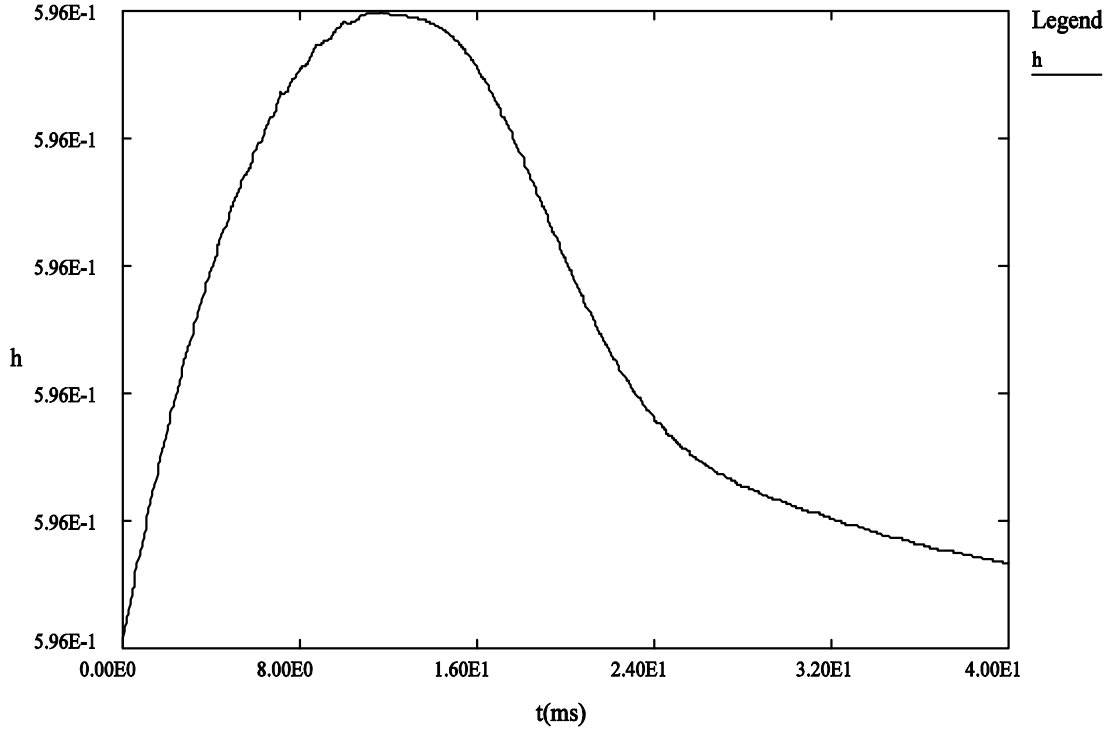
# of function evaluations = 309346

Результати розв’язку задачі нелінійного програмування (2.6) – (2.11) представлено на рис.2.9.






3.17E-1
3.17E-1
3.17E-1
n
3.17E-1
3.17E-1
3.17E-1
0.00E0
8.00E0
1.60E1
2.40E1
3.20E1
4.00E1
t(ms)
Legend
n


5.96E-1
5.96E-1
5.96E-1
h
5.96E-1
5.96E-1
5.96E-1
0.00E0
8.00E0
1.60E1
2.40E1
3.20E1
4.00E1
t(ms)
Legend
h

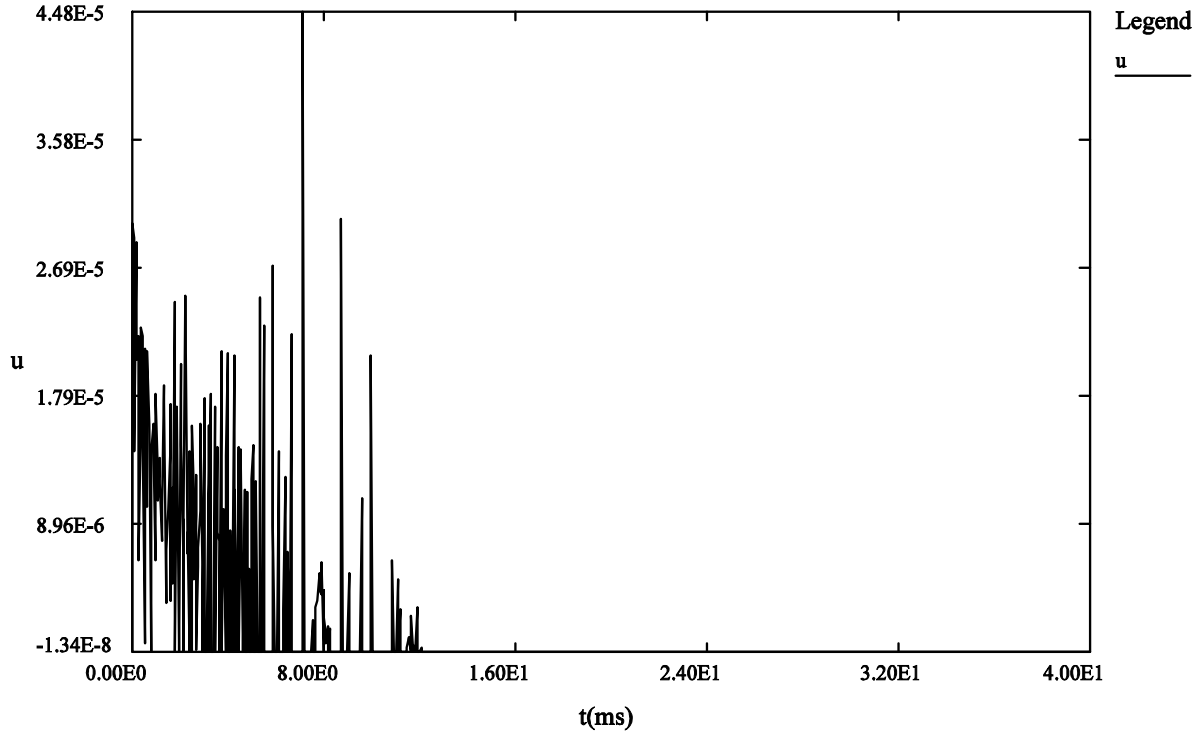


Рис. 2.9. Результати розв'язку задачі нелінійного програмування

Порівнюючи розв'язки на рисунках 2.8 та 2.9 бачимо, що застосування в моделі (2.6) – (2.11) оптимального режиму прикладеного струму дозволяє стабілізувати біфуркацію Хопфа.

Отже, прямий чисельний метод було застосовано для розв'язування задачі оптимального керування на основі моделі Ходжкіна-Хакслі. Показано, що застосування оптимального керування у вигляді прикладеного струму дозволило уникнути небажаних коливань в моделі.

Висновки до другого розділу

Отже, в розділі показано, що для якісного дослідження траєкторій моделі Ходжкіна-Хакслі може бути застосований запропонований і програмно реалізований метод побудови експонент Ляпунова. На основі знаків значень експонент можна роботи висновки про якісну поведінку траєкторії, зокрема стійкість і хаотичність.

З метою зміни якісної поведінки траєкторій моделі Ходжкіна-Хакслі в роботі запропоновано розгляд відповідної системи керування. При цьому для

таких оптимізаційних задач запропоновано в якості керуючих впливів розглядати прикладений струм. Для практичної побудови оптимального керування запропоновано і програмно реалізовано прямий чисельний метод. Таке стабілізаційне керування біфуркацією в системі Ходжкіна-Хакслі може мати важливе застосування при розробці методик лікування так званих «динамічних» захворювань (хвороба Альцгеймера, епілепсія, аритмія і ін.) [159].

Результати другого розділу опубліковано в роботах [77 – 80].

РОЗДІЛ 3
РОЗРОБЛЕННЯ МУЛЬТИВАРІАТИВНИХ МЕТОДІВ ДОСЛІДЖЕННЯ МОДЕЛІ ХОДЖКІНА – ХАКСЛІ У РІЗНИХ РЕЖИМАХ ЇЇ ФУНКЦІОНУВАННЯ, ІНТЕРПРЕТАЦІЯ РЕЗУЛЬТАТІВ МОДЕЛЮВАННЯ

3.1 Розроблення мультиваріативного методу якісного аналізу дослідження нелінійних систем

Якісна теорія диференціальних рівнянь у застосуванні до динамічних систем виявилася вкрай важливою у визначенні топології і структурної стійкості. Теорія бере початок з робіт Пуанкаре, далі отримала розвиток у роботах Біркгофа, Ляпунова, Андронова, Арнольда і ін.

У практичних задачах необхідно з'ясувати характер розв'язку диференціального рівняння, що описує фізичний процес та описати властивості розв'язків, коли незалежна змінна знаходиться на скінченному або нескінченному інтервалі.

Зауважимо, що існує лише кілька відносно простих рівнянь, загальні розв'язки яких можуть бути виражені в термінах інтегралів відомих функцій. Отже, виникає проблема дослідження властивостей розв'язків дифренціального рівняння виходячи безпосереднього із самого рівняння. Оскільки розв'язок диференціального рівняння представляється у вигляді кривої на площині або у просторі, то задача полягає у дослідженні властивостей інтегральних кривих, їх розподілів та їх поведінки в околі сингулярних точок. Наприклад, чи вони знаходяться у обмеженій частині площини, чи вони не містять відгалужень, що прямують до нескінченності, чи деякі із них є замкнутими кривими і т.ін. [136].

Використання диференціальних рівнянь для математичного моделювання зовсім не означає отримання аналітичної формули для їх розв'язків (або апроксимацій). Тим більше, що у більшості випадків таких формул не існує. До

того ж дуже часто формула для явного або наближеного аналітичного розв'язку настільки складна, що вона нічого не зможе сказати про природу розв'язку доти, поки не буде побудований графік [137]. Тому у практичних дослідженнях переважно генерують графіки розв'язків (принаймні їх загальну форму) не вдаючись до отримання розв'язків у вигляді формул [137].

Завданням методу є встановлення механізмів багатопараметричних впливів у системі Ходжкіна-Хакслі.

Загальні ідеї методу були розроблені в роботі [138] для випадку ЗДР. При цьому використовуємо підхід Монте-Карло, який полягає у випадковій генерації параметрів та побудові на їх основі системи Ходжкіна-Хакслі. Далі застосовують алгоритм індукції дерева рішень. Зауважимо, що метод роботи [138] застосовувався лише для дослідження впливу початкових умов на траєкторію системи ДР.

Припускається існування системи на основі (2.1) – (2.4) при початкових значеннях та швидкісних параметрах із заданих інтервалів.

Параметри

$$p \in P = \{(g_K, g_{Na}, g_L, V_K, V_{Na}, V_L, C_m, x_m, x_n, x_h) : g_K^{\min} \le g_K \le g_K^{\max},$$
$$g_{Na}^{\min} \le g_{Na} \le g_{Na}^{\max}, g_L^{\min} \le g_L \le g_L^{\max}, V_K^{\min} \le V_K \le V_K^{\max}, V_{Na}^{\min} \le V_{Na} \le V_{Na}^{\max},$$
$$V_L^{\min} \le V_L \le V_L^{\max}, C_m^{\min} \le C_m \le C_m^{\max}, x_m^{\min} \le x_m \le x_m^{\max}, x_n^{\min} \le x_n \le x_n^{\max},$$
$$x_h^{\min} \le x_h \le x_h^{\max}\} \subset R^{10},$$

а початкові умови

$$(V_0, m_0, n_0, h_0) \in X_0 = \{(V_0, m_0, n_0, h_0) : V_0^{\min} \le V_0 \le V_0^{\max}, m_0^{\min} \le m_0 \le m_0^{\max},$$
$$n_0^{\min} \le n_0 \le n_0^{\max}, h_0^{\min} \le h_0 \le h_0^{\max}\} \subset R^4.$$

Далі випадковим чином генеруватимемо початкові значення та значення швидкісних параметрів, які б належали практично обґрунтованій області. Для кожного з наборів таких параметрів здійснюється інтегрування системи (2.1) –

(2.4) з отриманням відповідних траєкторій. До отриманих результатів далі застосовується алгоритм індукції дерева рішень з метою знаходження певних шаблонів для прийняття рішень.

Отже, в цілому підхід включає такі п’ять кроків.

Крок 1. **Означення класів траєкторій моделі Ходжкіна-Хакслі.** Зазначимо, що в практичних застосуваннях переважно мають справу з набагато складнішими формами поведінки, щоб охарактеризувати їх поняттями «стійка-нестійка» і відповідно вдатися до аналізу власних значень або ж експонент Ляпунова динамічної системи. Тому визначення якісних форм процесу доцільно передати до компетенції експертів. В даному випадку використовуватимемо класи, пов’язані з формами збудливості нейронів: тип I, тип II, тип III.

Для позначення класу траєкторії вводиться атрибут класу C, який приймає одне з 3-х дискретних значень $C \in \overline{1,3}$. На рис. 1 – 3 наведені типові представлення для 3-х класів траєкторій – типів збудливості нейронів:

- тип I

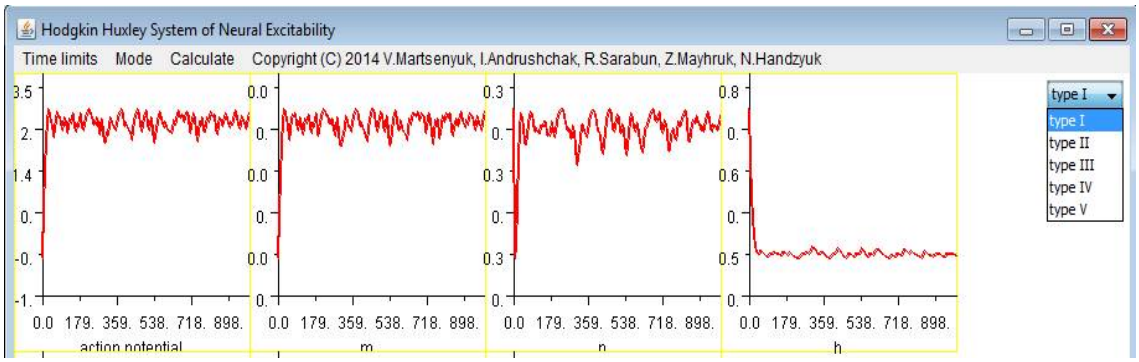


Рис. 3.1. I – клас збудливості нейронів.

- тип II

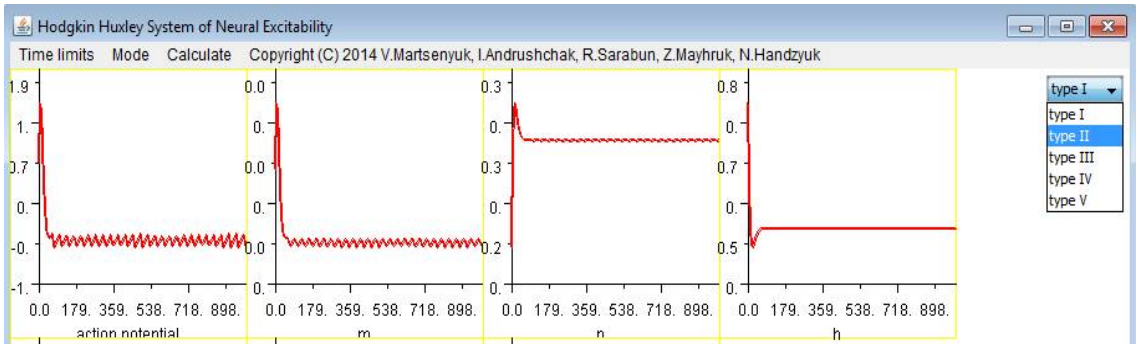


Рис. 3.2. II – клас збудливості нейронів.

- тип III

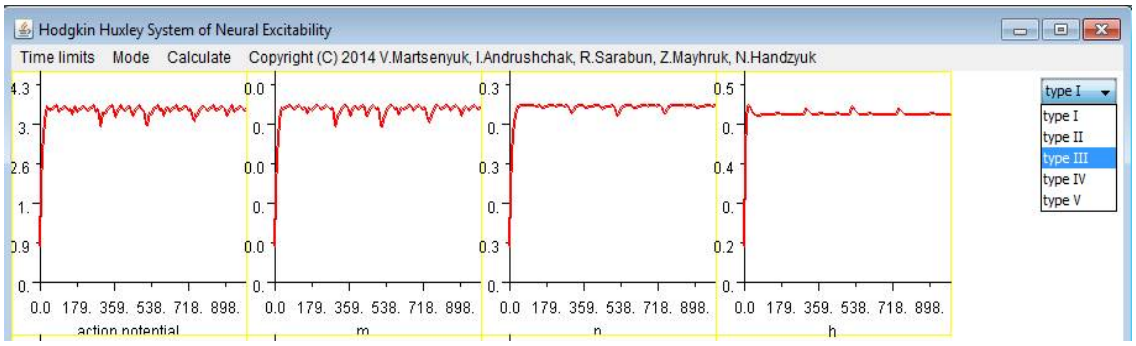


Рис. 3.3. III – клас збудливості нейронів.

Зазначимо, що при цьому визначальним є зміна частоти потенціалу дії у відповідь на зростання сили прикладеного струму.

Крок 2. **Генерація матриці випадкових початкових значень та швидкісних параметрів.** Для того, щоб дослідити весь простір початкових значень та швидкісних параметрів щодо генерації класів траєкторій, визначених на першому кроці, генерується матриця випадкових початкових значень та швидкісних параметрів на основі ймовірнісних розподілів у визначених інтервалах. У даній роботі ми припускаємо, що початкові значення та швидкісні параметри розподілені рівномірно на інтервалах. Кожен стовпчик відповідає множині значень одного параметру – або початкове значення, або швидкісний параметр. Кожен рядок є набором початкових значень та швидкісних параметрів для одного запуску моделі на основі (2.1) – (2.4):

$$M = \begin{pmatrix} V_0^1 & m_0^1 & n_0^1 & h_0^1 & g_K^1 & g_{Na}^1 & g_L^1 & V_K^1 & V_{Na}^1 & V_L^1 & C_m^1 & x_m^1 & x_n^1 & x_h^1 \\ V_0^2 & m_0^2 & n_0^2 & h_0^2 & g_K^2 & g_{Na}^2 & g_L^2 & V_K^2 & V_{Na}^2 & V_L^2 & C_m^2 & x_m^2 & x_n^2 & x_h^2 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ V_0^N & m_0^N & n_0^N & h_0^N & g_K^N & g_{Na}^N & g_L^N & V_K^N & V_{Na}^N & V_L^N & C_m^N & x_m^N & x_n^N & x_h^N \end{pmatrix} \in R^{N\times 14} \quad (3.1)$$

Крок 3. **Запуск моделі і класифікація набору вхідних даних.** Кожен набір початкових значень та швидкісних параметрів, згенерованих на другому кроці, використовуються в якості входу для системи Ходжкіна-Хакслі. Чисельне інтегрування рівнянь здійснюється за допомогою методу Адамса[139]. Вихідні траєкторії класифікуються на основі критеріїв, запропонованих на першому кроці. Виходячи з результатів класифікації наборам початкових значень і швидкісних параметрів приписуються відповідні атрибути класів.

Крок 4. **Побудова матриці залежностей між початковими значеннями та між швидкісними параметрами.** Метод припускає, що для форми траєкторій системи співвідношення між початковими значеннями та між швидкісними значеннями є набагато важливішими порівняно з їх абсолютними значеннями. Тому будується матриця, що включає інформацію у категорованому кодованому вигляді про співвідношення між початковими значеннями та між швидкісними параметрами, згенерованими на кроці 2:

$$D = \begin{pmatrix} m_0 \otimes n_0 & m_0 \otimes h_0 & n_0 \otimes h_0 & g_K \otimes g_{Na} & g_K \otimes g_L & g_{Na} \otimes g_L & V_K \otimes V_{Na} \\ x(m_0^1, n_0^1) & x(m_0^1, h_0^1) & x(n_0^1, h_0^1) & p(g_K^1, g_{Na}^1) & p(g_K^1, g_L^1) & p(g_{Na}^1, g_L^1) & p(V_K^1, V_{Na}^1) \\ x(m_0^2, n_0^2) & x(m_0^1, h_0^1) & x(n_0^1, h_0^1) & p(g_K^1, g_{Na}^1) & p(g_K^1, g_L^1) & p(g_{Na}^1, g_L^1) & p(V_K^1, V_{Na}^1) \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ x(m_0^k, n_0^k) & x(m_0^1, h_0^1) & x(n_0^1, h_0^1) & p(g_K^1, g_{Na}^1) & p(g_K^1, g_L^1) & p(g_{Na}^1, g_L^1) & p(V_K^1, V_{Na}^1) \end{pmatrix}$$

$$\left.\begin{array}{cccccccc} V_K \otimes V_L & V_{Na} \otimes V_L & C_m \otimes 1 & x_m \otimes x_n & x_m \otimes x_h & x_n \otimes x_h & C \\ p(V_K^1, V_L^1) & p(V_{Na}^1, V_L^1) & p(C_m^1, 1) & p(x_m^1, x_n^1) & p(x_m^1, x_h^1) & p(x_n^1, x_h^1) & C_1 \\ p(V_K^1, V_L^1) & p(V_{Na}^1, V_L^1) & p(C_m^1, 1) & p(x_m^1, x_n^1) & p(x_m^1, x_h^1) & p(x_n^1, x_h^1) & C_2 \\ \hdashline p(V_K^1, V_L^1) & p(V_{Na}^1, V_L^1) & p(C_m^1, 1) & p(x_m^1, x_n^1) & p(x_m^1, x_h^1) & p(x_n^1, x_h^1) & C_k \end{array}\right) \in R^{k \times 14} \quad (3.2)$$

Тут $x(u,v) = p(u,v) = \begin{cases} 0, & if \quad u < v \\ 1, & if \quad u = v \\ 2, & if \quad u > v \end{cases}$,

$C_l \in \overline{1,3}$ – значення атрибуту класу, пов'язані з відповідними формами траєкторій.

Отже, на даному кроці чисельні значення початкових значень та швидкісних параметрів трансформуються у категоріальні значення атрибутів наборів навчальних даних. Оскільки ймовірність рівності випадкових чисел дорівнює нулю, то матриця $D \in R^{k \times 14}$ виглядає свого роду «бінаризацією» співвідношень між початковими значеннями та між швидкісними параметрами. Тобто матриця D включатиме лише значення 0 та 2.

Крок 5. **Застосування алгоритму індукції дерева рішень до співвідношень між початковими значеннями та між швидкісними параметрами.** Матриця бінарних співвідношень D, побудована на кроці 4, є набором навчальних даних для алгоритму індукції дерева рішень. Побудоване дерево рішень міститиме перевірку співвідношень між початковими значеннями та швидкісними параметрами у своїх вузлах. В якості листків дерева будуть класи траєкторій моделі $C \in \overline{1,4}$.

Запропонований мультиваріативний метод якісного аналізу моделей електрофізіологічних процесів є підходом, що дозволяє вирішувати задачі класифікації збудливості клітин, які не можуть бути вирішені іншими традиційними методами, наприклад теорії стійкості, або ж граничних циклів.

У цілому метод поєднує у собі підхід Монте-Карло для формування навчальних наборів та класифікаційні алгоритми data mining: метод послідовного покриття з генерацією класифікаційних правил та метод індукції дерева рішень.

Перевагами класифікаційних правил, що можуть будуватися на 5-му кроці алгоритму, є те, що вони відповідають природньому відображенню знань в мисленні людей і є більш виражальними. До того ж алгоритм послідовного покриття є легшим в реалізації та відлагодженні порівняно із рекурсійними алгоритмами дерев рішень, а його обчислювальна складність є простішою порівняно із скінченними автоматами.

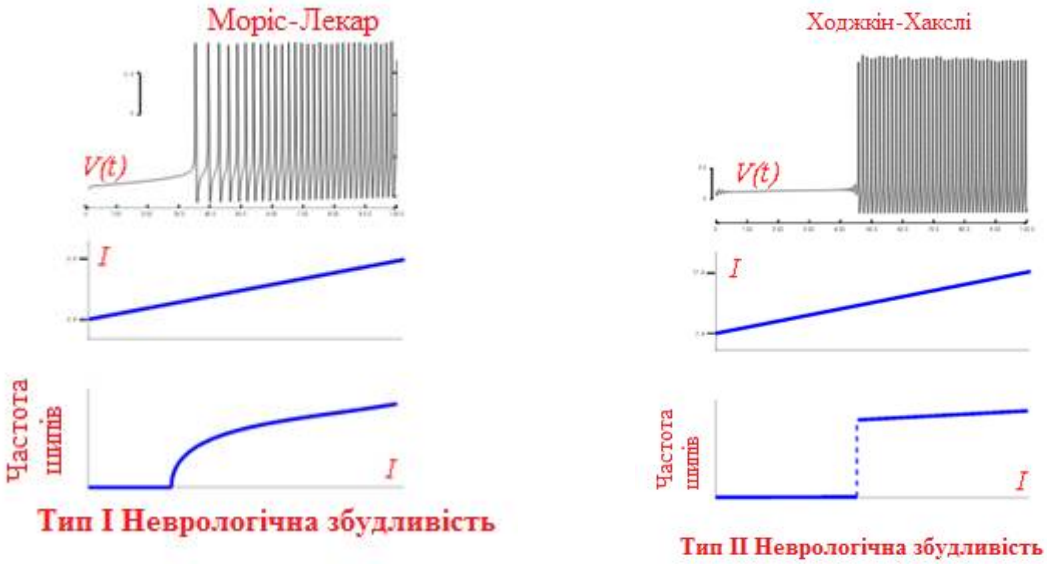
Метод індукції дерева рішень, будучи складнішим в реалізації, допускає наочну візуалізацію та апріорні значення ймовірностей для типу збудливості клітини виходячи із співвідношень між початковими значеннями та швидкісними параметрами у моделі Ходжкіна-Хакслі.

Отже, з метою якісного аналізу траєкторій моделі Ходжкіна-Хакслі розроблено мультиваріативний метод, що включає алгоритми data mining для побудови структур знань – наприклад, дерев рішень або класифікаційних правил. Такий підхід дозволяє здійснити складнішу класифікацію траєкторій порівняно з дослідженнями стійкості, а саме дослідити різні типи збудливості клітинних мембран.

3.2 Спеціалізація алгоритмів технології data mining для класифікації траєкторій динамічної системи Ходжкіна-Хакслі з урахуванням особливостей реальних технічних задач

Мультиваріативний метод якісного аналізу моделі Ходжкіна-Хакслі отримав програмну реалізацію. З цією метою розроблено пакет Java-класів decision_tree.fde.hh [97]. При цьому використовуються такі поняття об’єктно-орієнтованого підходу: «клас» як множина однотипових об’єктів; «об’єкт» як інкапсуляція даних і програмного коду. Під «пакетом» мають на увазі

відображення ієрархічної системи класів у файлову систему комп'ютера. До складу пакету входять класи



а) б)

Рис. 3.4. Класифікація збудливості нейронів (з роботи [140]: а) тип I; б) тип II.

DataManager – клас – менеджер даних для отримання інформації з бази даних через посередництво відповідних класів-слуг;

MultiVariateMethod – клас для реалізації мультиваріативного методу, представленого в роботі – головний клас пакету;

TuplesPeer – клас-слуга для формування і обробки навчальних наборів, що використовуватиметься у класифікаційному алгоритмі.

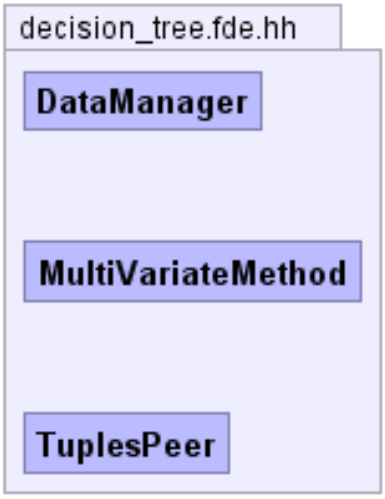


Рис. 3.5. Пакет decision_tree.fde.hh

У класі *MultiVariateMethod* (рис.3.6) здійснюється генерація випадкових значень параметрів (крок 2):

M_x0 = dm.getRandomInitialValues();

M_rateConstants = dm.getRandomRateConstants();

Далі запускається аплет інтегрування системи Ходжкіна-Хакслі. При цьому експерт здійснює вибір форми отриманої траєкторії (крок 3) (рис.3.4). Після цього запускається крок генерації матриці взаємозв'язків параметрів (крок 4). Зауважимо, що послідовність кроків 2-4 може виконуватися як завгодно багато разів. У будь-який момент користувач може запустити алгоритм індукції дерева рішень (крок 5):

dtDecision_tree_HH = new decision_tree.fde.hh.DecisionTree(dmtnRoot, dataManager_FDE, htAttribute_list, "Information gain");

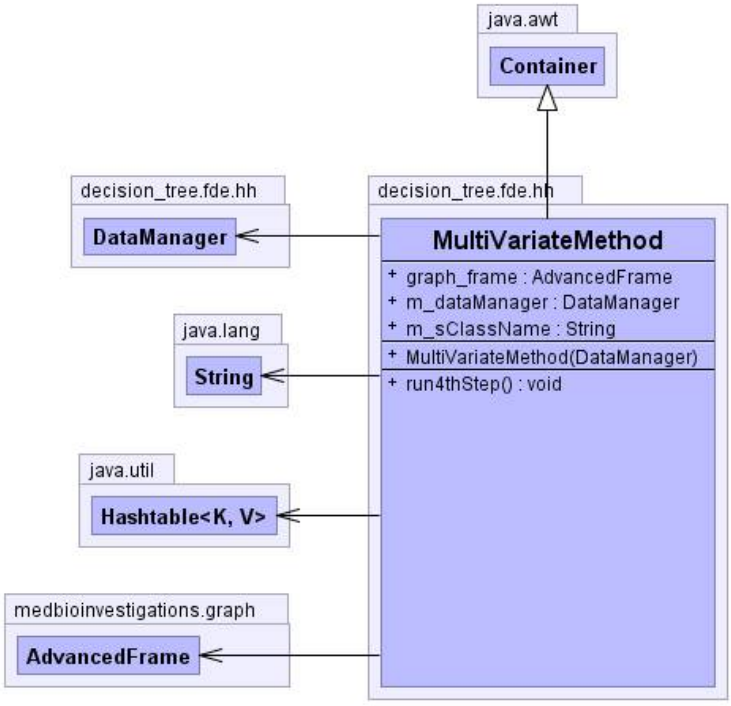


Рис. 3.6. UML-діаграма класу MultiVariateMethod

База даних *hh*, що використовується в пакеті, реалізована в СУБД MySQL. Вона включає такі таблиці (рис.3.5, рис.3.7.):

attribute – опис атрибутів для побудови дерева рішень, тобто взаємозв'язків між початковими значеннями та між швидкісними константами;

categorized_data – навчальні набори, що використовуються в класифікаційному алгоритмі (в даному випадку індукції дерева рішень) і представляють собою матрицю D на четвертому кроці;

init_values_values – матриця згенерованих випадковим чином початкових значень:

$$\begin{pmatrix} V_0^1 & m_0^1 & n_0^1 & h_0^1 \\ V_0^2 & m_0^2 & n_0^2 & h_0^2 \\ \cdots & \cdots & \cdots & \cdots \\ V_0^N & m_0^N & n_0^N & h_0^N \end{pmatrix} \in R^{N\times 4} \qquad (3.3)$$

initial_values – опис початкових значень (включаючи мінімальні та максимальні значення);

parameter_kind – вид параметру;

rate_constants – опис швидкісних констант (включаючи мінімальні та максимальні значення);

rate_constants_values – матриця згенерованих випадковим чином швидкісних констант:

$$\begin{pmatrix} g_K^1 & g_{Na}^1 & g_L^1 & V_K^1 & V_{Na}^1 & V_L^1 & C_m^1 & x_m^1 & x_n^1 & x_h^1 \\ g_K^2 & g_{Na}^2 & g_L^2 & V_K^2 & V_{Na}^2 & V_L^2 & C_m^2 & x_m^2 & x_n^2 & x_h^2 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ g_K^N & g_{Na}^N & g_L^N & V_K^N & V_{Na}^N & V_L^N & C_m^N & x_m^N & x_n^N & x_h^N \end{pmatrix} \in R^{N\times 10} \quad (3.4$$

)

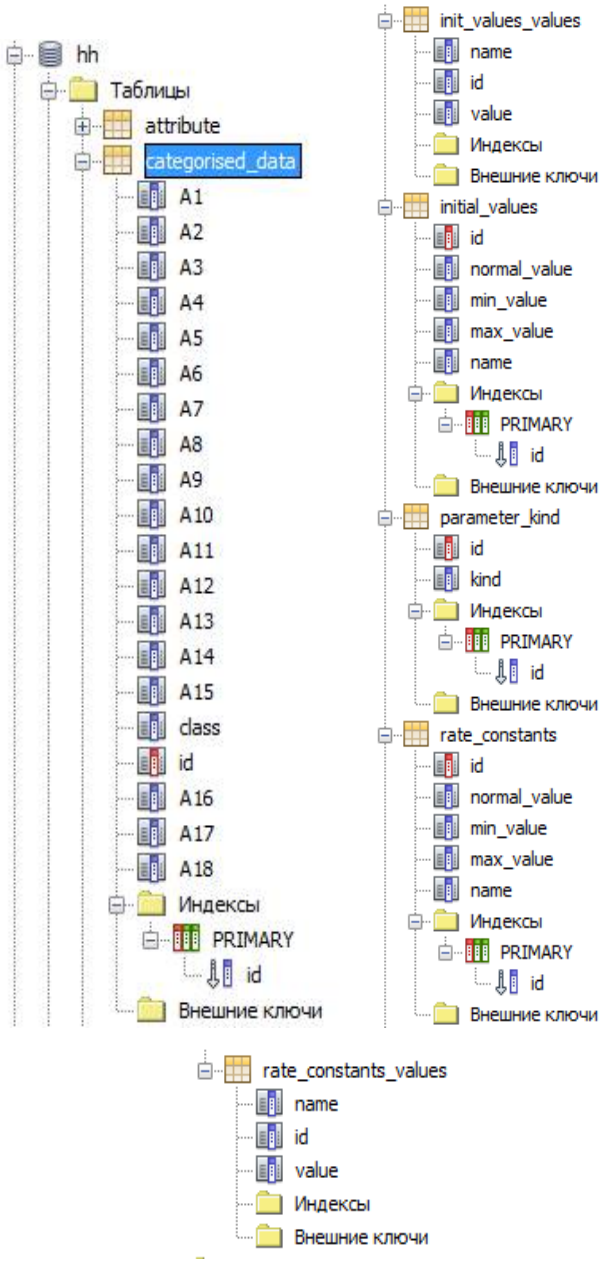


Рис. 3.7. Таблиці бази даних hh

Отже, використовуючи об’єктно-орієнтований підхід, показано шлях практичної реалізації мультиваріативного методу якісного аналізу моделі Ходжкіна-Хакслі.

3.3 Модифікація методу класифікації типів збудливості нервових клітин з побудовою класифікаційних правил та дерев рішень

Попередньо вже вивчалося питання різних типів збудливості нейронів. При збільшенні прикладеного струму збудливі мембрани можуть переходити із стану спокою до повторних спалахів потенціалу дії кількома способами [141]:

- для збудливих нейронів типу I частота потенціалу дії збільшується плавно від початку спалаху, таким чином проходячи увесь діапазон частот;

- для збудливих нейронів типу II спостерігається різкий скачок частоти потенціалу дії для певного порогового значення сили струму. Потенціал дії не індукується для значень частоти, нижчих від порогового;

- відповіддю на стійкі надпорогові імпульси струму є один зубець. Така поведінка отримала назву «збудливість типу III» [142].

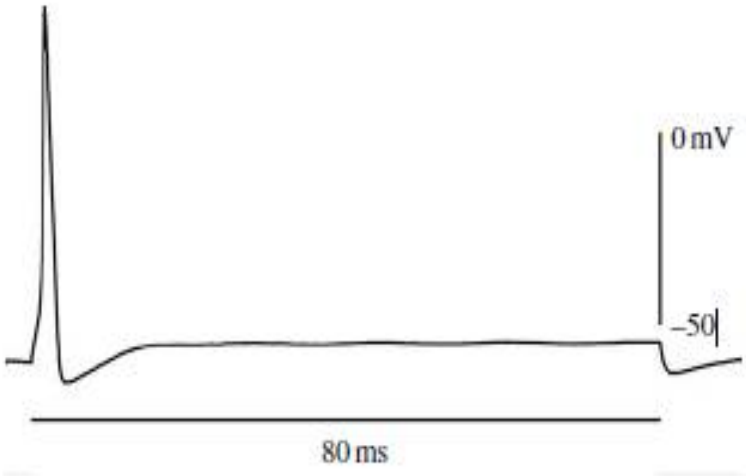


Рис. 3.8. Тип III збудливості нейронів: реакція гігантського аксона кальмара збудливості типу III на тривалому пульсі густини $15\mu A\ cm^{-2}$ струму по амплітуді $T= 8^0C$ (з роботи [142]).

З метою класифікації типів збудливості клітин використаємо алгоритм послідовного покриття, описаний в роботі [143]. Припускаємо, що усі атрибути – категоріальні.

Алгоритм послідовного покриття

Вхідні дані:

D – множина навчальних наборів даних $(A_1^i, A_2^i, ..., A_p^i, C^i)$

Att_vals – множина всіх атрибутів $A_1, ..., A_p$ та їх можливих значень $A_i \in \{a_i^1, a_i^2, ..., a_i^{K_i}\}$

Вихідні дані: *Rule_set* – множина класифікаційних правил.

Метод:

1. Множина класифікаційних правил *Rule_set* = {}

2. Для кожного класу c

3. Розпочати цикл "до"

4. Побудувати нове класифікаційне правило

Rule = Добути_одне_правило (D, Att_vals, c)

5. Вилучити набори навчальних даних з *D*, що покриваються правилом *Rule*

6. Виконувати цикл з кроку 3 до настання *термінальної умови*

7. Додати нове правило до множини класифікаційних правил:

Rule_set = Rule_set + Rule

8. Кінець циклу з кроку 2

9. Множина навчальних правил в *Rule_set*

В основу методу *Добути_одне_правило (D, Att_vals, c* покладена міра приросту інформації для побудови правил логіки першого порядку FOIL (First Order Inductive Learner). Метод є ітераційною процедурою по усіх атрибутах $A_1, ..., A_p$.

Припустимо, що ми вже маємо класифікаційне правило:

R: IF *condition* THEN *class = c*.

Метою кожного кроку $i=\overline{1,p}$ є кон'юнкція умови *condition* за рахунок умови *condition'* вигляду $(A_i = a_i^j)$. Тут $j \in \{1,...,K_i\}$. Тобто нове правило матиме вигляд:

R': IF *condition* AND *condition'* THEN *class = c.*

Згідно методу FOIL *condition'* вибирається з умови мінімізації міри:

$$FOIL\_Gain = pos' \times (\log_2 \frac{pos'}{pos'+neg'} - \log_2 \frac{pos}{pos+neg}) \tag{3.5}$$

Тут *pos(neg)* – число позитивних(негативних) навчальних наборів, що покриваються правилом *R*, *pos'(neg')* – число позитивних(негативних) навчальних наборів, що покриваються правилом *R'*. Під позитивними (негативними) навчальними наборами для певного правила маємо на увазі навчальні набори з умовою консеквенту, які задовольняють (не задовольняють) умови антеседенту правила.

Міра (3.1) сприяє побудові правил, що мають більшу точність і при цьому покривають якомога більше позитивних навчальних наборів.

Математично задача індукції дерева рішень формулюється таким чином. Маємо множину D, що містить N наборів навчальних даних. При цьому кожен i-й набір $(A_1^i, A_2^i, ..., A_p^i, C^i)$ складається з вхідних даних – атрибутів $A_1,..., A_p$ та вихідних даних – атрибуту класу C. Атрибути $A_1,..., A_p$ можуть приймати як чисельні, так і категоріальні значення. Атрибут класу C приймає одне з K дискретних значень: $C \in \{1,...,K\}$. Метою є прогнозування деревом рішень значення атрибуту класу C на основі значень атрибутів $A_1,..., A_p$. При цьому слід максимізувати точність прогнозування атрибуту класу, а саме $P\{C = c\}$ на термінальних вузлах для довільного $c \in \{1,...,K\}$. Алгоритми індукції дерев рішень автоматично розбивають на вузлах значення чисельних атрибутів A_i на два інтервали: $A_i \le x_i$ та $A_i > x_i$, а категоріальних атрибутів A_j – на дві

підмножини: $A_j \in S_j$, $A_j \notin S_j$. Розбиття чисельних атрибутів грунтується, як правило, на мірах на основі ентропії, або індексі Джині [144]. Процес розбиття рекурсивно повторюється до тих пір, поки не спостерігатиметься покращення точності прогнозування. Останній крок включає вилучення вузлів для уникнення оверфітінгу моделі. В результаті ми повинні отримати множину правил, що йдуть від кореня до кожного термінального вузла, містять нерівності для чисельних атрибутів та умови включення для категоріальних атрибутів.

Метою підрозділу є застосувати метод індукції дерева рішень для програмної реалізації в системі електрофізіологічних досліджень.

За основу взято таку рекурсивну процедуру роботи [144].

Метод індукції дерева рішень.

Генерація дерева рішень

Вхідні дані: D – множина навчальних наборів даних $(A_1^i, A_2^i, ..., A_p^i, C^i)$.

Вихідні дані: дерево рішень

Метод:

1. Створити вузол N.

2. Якщо усі набори в D належать до спільного класу C, тоді повернути вузол N як листок із назвою класу C.

3. Якщо список атрибутів (а отже і D) є порожнім, тоді повернути вузол N як листок із назвою найпоширенішого класу в D.

4. Застосувати *Алгоритм відбору атрибуту* із списку атрибутів і для множини D з метою відшукання "найкращого" атрибуту поділу.

5. Вилучити атрибут поділу із списку атрибутів.

6. Для кожної умови поділу *j* для атрибуту поділу розглянемо D_j – множину наборів з *D*, що задовольняють умову поділу *j*.

7. Якщо D_j – порожня, тоді приєднати до вузла N листок під заголовком найпоширенішого класу в D_j, інакше – приєднати до *N* вузол, що

повертається рекурсивним викликом методу *Генерація дерева рішень* з вхідними даними D_j та список атрибутів.

8. Кінець циклу кроку 6.

9. Повернути вузол N.

В основу *Алгоритму відбору атрибуту* на j – му кроці рекурсії покладено такий інформаційний показник:

$$Gain(A_i) = Info(D_j) - Info_{A_i}(D_j). \tag{3.6}$$

Тут

$$Info(D_j) = -\sum_{k=1}^{K} p_k^j \log_2(p_k^j) \tag{3.7}$$

– інформація, потрібна для класифікації набору $(A_1, A_2, ..., A_p)$ в D_j,

$$Info_{A_i}(D_j) = \sum_{l=1}^{K_i} \frac{\#(D_j^l)}{\#(D_j)} Info(D_l) \tag{3.8}$$

– інформація, потрібна для класифікації $(A_1, A_2, ..., A_p)$ в D_j після поділу D_j на підмножини D_j^l відповідно до значень атрибуту A_i.

У формулі (3.26) ймовірність того, що довільний набір з D_j належить множині C_{k,D_j} оцінюється як $p_k^j = \frac{\#(C_{k,D_j})}{\#(D_j)}$, де C_{k,D_j} – множина наборів з D_j, для яких атрибут класу $C = k$. Тут $\#(\bullet)$ – кількість елементів в множині.

У формулі (3.7) $\frac{\#(D_j^l)}{\#(D_j)}$ – оцінка ймовірності того, що довільний набір з D_j належить множині D_j^l, де D_j^l – множина наборів з D_j, для яких атрибут $A_i = a_i^l$. Тут атрибут $A_i \in \{a_i^1, a_i^2, ..., a_i^{K_i}\}$.

Отже, $Gain(A_i)$ оцінює зменшення інформації, необхідної для класифікації довільного набору даних в D_j за рахунок відомого значення атрибуту A_i. Таким чином з наявних атрибутів на кожному вузлі дерева рішень для умови поділу слід відбирати атрибут A_{i^*} з найбільшим значенням $Gain(A_{i^*})$. В результаті такого вибору для завершення процесу класифікації набору даних в D_j вимагатиметься найменше інформації.

Метод індукції дерева рішень може бути модифікований шляхом використання показника відношення приростів інформації.

Традиційний метод індукції дерева рішень ґрунтується на обчисленні показника приросту інформації. Як вже було попередньо зазначено [144], показник приросту інформації $Gain(A_i)$ має ухил в сторону тестів з багатьма виходами. Тобто він надає перевагу атрибутам, які мають дуже велике число значень.

Тому метою даного підрозділу є реалізація алгоритму індукції дерева рішень, який би був позбавлений такого недоліку через використання показника відношення приростів інформації.

Для прикладу розглянемо як атрибут ідентифікатор id з таблиці навчальних даних categorised_data. Поділ набору навчальних даних по даному атрибуту призведе до великої кількості розбиттів (дорівнює кількості пацієнтів), кожне з яких включатиме лише один запис. Оскільки кожне розбиття не потребує подальшого поділу, то інформація, яка необхідна для класифікації даних в D_j на основі цього розбиття становить $Info_{id}(D_j) = 0$ (це можна показати і з формул для обчислення $Info_{A_i}(D_j)$ та $Info(D_j)$ також).

Отже, приріст інформації внаслідок розбиття по цьому атрибуту є максимальним. З іншої сторони, таке розбиття є безглуздим з точки зору класифікації.

З метою подолання такого недоліку в методі C4.5 [144] використовується удосконалений показник на основі приросту інформації, який називається відношення приростів (gain ratio). Він застосовує до приросту інформації свого роду нормалізацію, використовуючи значення "розділяючої інформації", яке визначається за аналогією до $Info(D)$:

$$SplitInfo_{A_i}(D_j) = -\sum_{l=1}^{K_i} \frac{\#(D_j^l)}{\#(D_j)} \log_2 \left(\frac{\#(D_j^l)}{\#(D_j)} \right). \tag{3.9}$$

Таке значення представляє потенційну інформацію, яка ґенерується при поділі набору навчальних даних D_j на K_i розбиттів, що відповідають K_i виходам тесту щодо атрибуту A_i. Зауважимо, що для кожного виходу розглядається кількість записів, що відповідають умові даного виходу відносно загального числа записів в D_j. Це відрізняє його від показника приросту інформації, в якому вимірюється інформація щодо класифікації, яка отримується на основі того ж розбиття. Згідно означення відношення приростів становить:

$$GainRatio(A_i) = \frac{Gain(A_i)}{SplitInfo(A_i)}. \tag{3.10}$$

Атрибут з максимальним значенням відношення приростів обирається як атрибут поділу. Хоча, як зазначають в [144] у випадку, коли інформація поділу наближається до нуля, то значення відношення приростів стає нестійким. Тому слід накладати відповідні обмеження на значення інформації поділу.

В цілому отримуємо таку ж рекурсивну процедуру, як і попередньо, за виключенням того, що на 4-му кроці застосовується *Алгоритм відбору атрибуту* із списку атрибутів для множини D на основі показника відношення приростів інформації з метою відшукання "найкращого" атрибуту поділу.

Отже, запропоновані реалізації алгоритмів data mining, а саме методу послідовного покриття та індукції дерева рішень з метою класифікації типів збудливості клітинних мембран.

Висновки до третього розділу

У розділі запропоновано мультиваріативний метод якісного аналізу моделей електрофізіологічних процесів. Такий підхід дозволяє вирішувати задачі класифікації збудливості клітин, які не можуть бути вирішені іншими традиційними методами, наприклад теорії стійкості, або ж граничних циклів.

Метод поєднує у собі підхід Монте-Карло для формування навчальних наборів та класифікаційні алгоритми data mining: метод послідовного покриття з генерацією класифікаційних правил та метод індукції дерева рішень.

Перевагами класифікаційних правил є те, що вони відповідають природньому відображенню знань в мисленні людей і є більш виражальними. До того ж алгоритм послідовного покриття є простішим в реалізації та відлагодженні порівняно із рекурсійними алгоритмами дерев рішень, а його обчислювальна складність є простішою порівняно із скінченними автоматами.

Метод індукції дерева рішень, будучи складнішим в реалізації, допускає наочну візуалізацію та апріорні значення ймовірностей для типу збудливості клітини виходячи із співвідношень між початковими значеннями та швидкісними параметрами у системі Ходжкіна-Хакслі.

Підхід доведено до програмної реалізації в пакеті Java-класів.

Результати третього розділу опубліковано в роботах [81 – 83].

РОЗДІЛ 4
ПОБУДОВА ПРОГРАМНОЇ СИСТЕМИ МОДЕЛЮВАННЯ ЕЛЕКТРОФІЗІОЛОГІЧНИХ ДОСЛІДЖЕНЬ

4.1 Організація структури Web – інтегрованої програмної системи електрофізіологічних досліджень на основі системи Ходжкіна-Хакслі

4.1.1 Принципи побудови програмного середовища електрофізіологічних досліджень. Загальні принципи до побудови автоматизованої системи керування були запропоновані В. М. Глушковим [17, 28]. З метою розробки програмного середовища електрофізіологічних досліджень застосуємо такі положення:

Положення 1. Принцип нових задач. Згідно цьому принципу застосування програмного середовища електрофізіологічних досліджень до розв’язування задач, що традиційно вже знайшли некомп’ютерний шлях вирішення, є неефективним. Крім того, програмне середовище електрофізіологічних досліджень зорієнтоване на моделювання та аналіз задач медицини, які традиційно не могли бути розв’язані в лабораторних чи клінічних умовах: або внаслідок великої вартості проекту, або великих часових проміжків, або внаслідок відсутності лабораторного обладнання (наприклад, для локальної фіксації біопотенціалу).

Положення 2. Принцип комплексного підходу. Повинна бути проведена структуризація об’єкта керування та системи керування ним. Традиційно, в якості об’єкта керування при проведенні біомедичних досліджень розглядають живий організм. Керуванням відносно нього є різного роду експерименти, що проводяться з метою розв’язання двох класів задач:

– винайдення ефективних методик діагностування;

– пошук оптимальних схем лікування (терапевтичного або хірургічного).

У підрозділі 4.1.2 буде наведена структуризація об’єкта та керування, якої дотримуватимемося.

Положення 3. Принцип максимально доцільної мінімізації проектних рішень. Проект програмного середовища електрофізіологічних досліджень повинен використовуватися при розв’язанні багатьох споріднених задач. Велике значення при цьому має дотримання об’єктно-орієнтованого підходу при розробці концептуальної моделі програмного середовища електрофізіологічних досліджень, що дозволить інкапсулювати методи якісного аналізу (наприклад мультиваріативний метод) з даними експериментальних досліджень. Потенційні задачі, де проект програмного середовища електрофізіологічних досліджень, що розробляється, може бути використаним, – це програмне середовище електрофізіологічних досліджень на основі наукової медичної інформації, організаційні науково-дослідні медичні ПСЕФД, а також технологічні медичні ІКС (клініко-лабораторні дослідження, консультативна комп’ютерна діагностика, постійний моніторинг пацієнта) тощо.

Положення 4. Принцип неперервного розвитку системи. Первинно під цим малося на увазі сповідування модульної процедурно-орієнтованої структури побудови ПСЕФД. Як виявилося, в подальшому в другій половині 1980 рр., ціле покоління АСУ (зокрема і медичного призначення), що ґрунтувалися на модульній організації, не були здатними перенестися на нове апаратне та програмне забезпечення. Як гарант інваріантності ПСЕФД до змін, як у програмному забезпеченні, так і в підходах до моделювання живого організму пропонується об’єктно-орієнтована організація інформаційної моделі ПСЕФД та її складових. Розумне використання таких понять в об’єктно-орієнтованому підході, як абстрактні класи та методи робить можливості ПСЕФД щодо її модернізації та поповнення новими задачами практично невичерпними.

Положення 5. Принцип єдиної інформаційної бази. Полягає в тому, що повинно уникатися дублювання інформації, а накопичена в процесі роботи ПСЕФД інформація повинна використовуватися для розв'язування багатьох задач. Інформація про виконані медико-біологічного дослідження повинна зберігатися у вигляді бази даних (можливо розподіленої), приведеної до відповідної канонічної нормальної форми [147].

Положення 6. Принцип стандартизації систем програмування. Одинакові або подібні задачі повинні розв'язуватися на різній технічній базі. Представлена ПСЕФД орієнтована на Інтернет-програмування. При цьому ядро ПСЕФД повинно розміщуватися на комп'ютері сервері. Користувач ПСЕФД бачить на комп'ютері-клієнті (який може бути найрізноманітнішої конфігурації) лише результати роботи програм ПСЕФД.

Положення 7. Принцип дружнього інтерфейсу при введенні та виведенні інформації. Як і 20 років тому, введення та виведення інформації залишається вузьким місцем комп'ютерів. Та якщо раніше принцип полягав у мінімізації в процесі введення та виведення (тобто щоб його взагалі обминути), то сьогодні ставиться задача надання способу введення-виведення зручності (дружності). Стосовно медичних ПСЕФД, то крім типових технічних проблем, пов'язаних із введенням-виведенням текстової у вигляді форм, графічної, аудіо- та відео-інформації (які на сьогодні долаються за допомогою графічного інтерфейсу та методів цифрової обробки інформації відповідно) долучаються ще чисто психологічні аспекти, пов'язані з особливістю користувача, що працюватиме з програмним середовищем електрофізіологічних досліджень. Користувачі (а це, як правило, фахівці в галузі біологічної та медичної кібернетики, біологи, фізіологи, клініцисти) вирізняються рівнем математичної підготовки і здатністю до сприйняття біологічних закономірностей через аналітичні результати. Це вимагає від ПСЕФД надання можливостей роботи принаймні в трьох режимах: біокібернетика, фізіологія, клініка та розробки додаткових інтерфейсів (наприклад, візуальний конструктор функції).

4.1.2 Концептуальна модель ПСЕФД. Структуризація об'єкта та системи керування. Усі медичні наукові дослідження, що проводяться щодо живого організму, стосуються двох головних напрямків [43]:

Перший напрямок – вивчення клітин та клітинних механізмів взагалі. Сюди входять такі розділи питань.

- Біохімічні реакції. Кінетика білків.
- Клітинний гомеостаз: Вивчення клітинних мембран. Дифузія через мембрани. Активний та пасивний транспорт. Мембранні потенціали. Осмозис. Керування об'ємом клітин.
- Мембранні іонні канали. Моделі електродифузії. Бар'єрні моделі. Мембранні канали.
- Збудливість клітин. Система Ходжкіна-Хакслі.
- Динаміка кальція.
- Міжклітинні взаємодії.
- Серцеві клітини: Серцеві волокна. Тканина міокарда. Волокна Пуркіньє. Синоатрикулярний вузол. Клітини шлуночка.
- Кальцієві хвилі.
- Регуляція функцій клітин. Керування циклом поділу клітини.

Другий напрямок – вивчення органів та їх фізіологічних систем. Тут слід вказати на значний доробок М.М.Амосова [31, 48], що знайшов втілення у блок-схемах кровообігу, зовнішнього дихання та тканинного метаболізму, терморегуляції, водно-сольового обміну та внутрішньої сфери організму людини в цілому, виконаних в термінах теорії автоматичного регулювання. Ми ж вкажемо на такі основні напрямки досліджень, на яких ґрунтуватиметься інформаційна модель ПСЕФД.

- Серце та серцева ритмічність. Сюди стосуються дослідження щодо скалярних та векторних електрокардіограм, пейсмейкерів, серцевих аритмій, дефібриляції.

– М’язи. Теорія перехрестних містків. Моделі Хілла (зв’язок прикладеної сили та швидкості м’язевого скорочення). Моделі Хакслі (спрощена модель перехрестного містка).

4.1.3 Структура інформаційної моделі інтегрованого середовища програмного середовища електрофізіологічних досліджень. В роботі [48] викладено концепції побудови інтегрованого середовища складної ПСЕФД. Дотримуючись їх, інтегроване середовище системних електрофізіологічних досліджень (ІССЕФД) ми пропонуємо розглядати як багатовимірний інформаційний простір, де вводиться п’ять основних інформаційних проекцій, відносно яких потрібно побудувати опис існуючої ПСЕФД, а саме:

1) проекція Топологія наукових напрямків медичних досліджень, яких дотримується ПСЕФД, SDT – проекція (Scientific Directions Topology);

2) проекція Структури програмного забезпечення і Ресурсів даних, які використовуються в системі, SDS – проекція (Software and Data Structures);

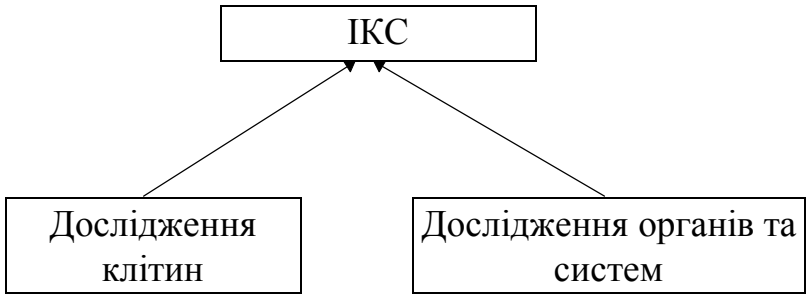
3) проекція Інформаційні профілі користувачів, які взаємодіють з ПСЕФД як безпосередньо в її структурі, так і за її межами (віддалені користувачі), UIP – проекція (User Information Profiles);

4) проекція Опис нових результатів і задач, які виникають при проведенні біомедичних досліджень, NRP – проекція (New Results and Problems);

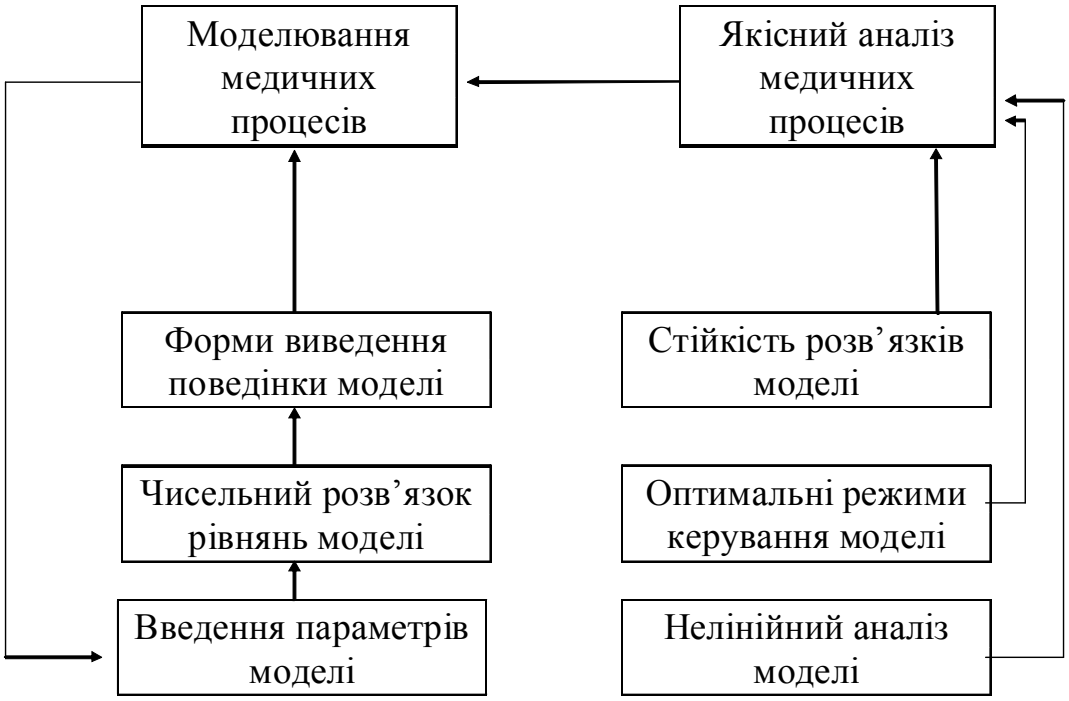
5) проекція Шляхи впровадження нових результатів та розв’язування нових задач для вдосконалення ПСЕФД, NRU – проекція (New Results Upgrade).

Отже, формалізований опис пропонованої ІССЕФД – позначимо його як IMSRE (Integrated Medical System Researches Environment) – можна задати як об’єднання п’яти запропонованих вище проекцій, а саме: $IMSRE = SDT \cup SDS \cup UIP \cup NRP \cup NRU$.

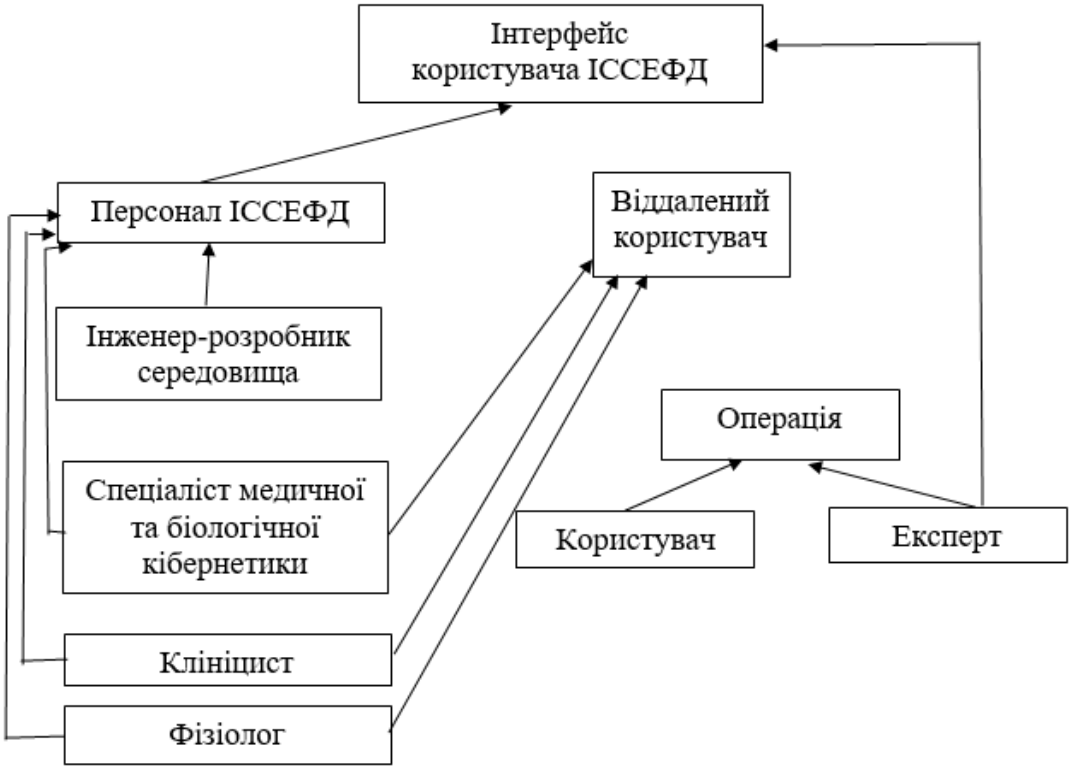
Далі наведено відповідні діаграми абстрактних класів проекцій. Стрілками показано головні напрямки спадщини у класах.



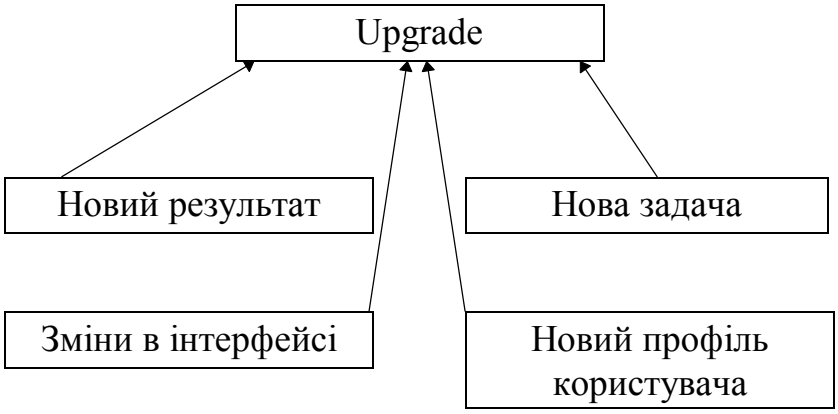
а) SDT – проекція



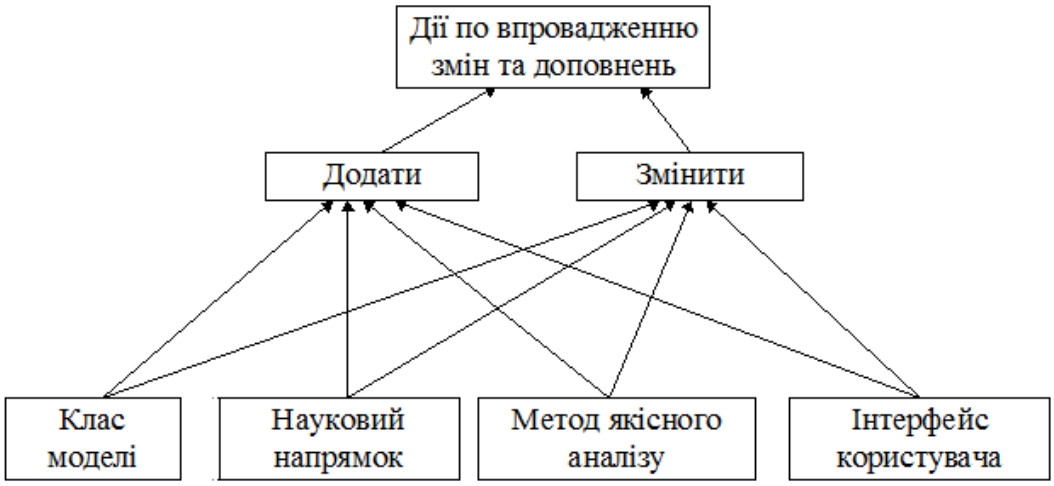
б) SDS – проекція



в) UIP – проекція



г) NRP – проекція



д) NRU – проекція

Рис. 4.1. Діаграми проекцій інформаційного простору програмного середовища електрофізіологічних досліджень.

4.2 Особливості комп'ютерного моделювання у фізіологічних дослідженнях

Моделі необхідні для того, щоб надати науковим ідеям просту і точну форму з метою передбачати події в середовищі, що постійно змінюється. Методи, що розробляються для застосування математичних моделей біологічних систем у комп'ютерних симуляційних дослідженнях, дозволяють висувати гіпотези щодо фізіології в живих організмах. Також стає можливим екстраполювати результати отримані в експериментах на клітинах і тканинах у пробірці на контекст усього організму. Оснащені сучасною обчислювальною технікою математичні моделі, що імітують фізіологічні процеси використовуються для теоретичних перевірок гіпотез щодо впливу локальних фізіологічних ефектів на організм в цілому [39].

Комп'ютерне моделювання, яке застосовується в медицині, поділяється на такі категорії [146]:

- Комп'ютерні текстові симулятори;
- Комп'ютерні графічні симулятори;

- Симулятори з використанням манекенів;

- Симулятори віртуальної реальності.

Розглянемо трохи детальніше кожен з класів.

Текстові симулятори створюють словесний опис ситуації, в якій користувач вибирає одну з декількох визначених відповідей. Грунтуючись на отриманій відповіді, комп'ютер генерує таку ситуацію.

Симулятори з використанням манекенів можуть бути доволі складні та реалістичні, але, зазвичай, завжди дорогі. Передові моделі включають фізичну модель людського тіла і безперервно реєструють сигнали, відбивають фізіологічні параметри, такі як електрокардіограма, артеріальний тиск, капнограмма, пульсоксиметрія і т.д. Сучасні варіанти використовують складні комп'ютерні моделі фізіології людини для автоматичної генерації відповідей манекеном і сигнальними відведеннями. На противагу текстовому і графічному тренажерам, використання манекенів дозволяє розвинути деякі практичні навички, які згодом будуть застосовуватися в клініці.

Технології віртуальної реальності набули популярності, особливо для навчання хірургів. При цьому можливий перехід від двовимірного управління операцією до світу тривимірних віртуальних хворих. Принципова складність полягає в моделюванні тактильних відчуттів. Тим не менше ця проблема активно розробляється в численних науково-дослідних центрах із залученням мікротехнологій [147].

Комп’ютерне моделювання в симуляційних дослідженнях успішно використовуються у фізіології для наукових досліджень з метою кращого зрозуміння взаємодії, що відбувається в складних біосистемах. Такі моделі часто служать як засіб формальної побудови гіпотез щодо пропонованих механізмів фізіологічного функціонування. Таким чином, вдається уникнути надмірних експериментів над лабораторними тваринами [39, 146].

Раціональні дослідження можуть бути виконані з використанням такого алгоритму дій [148, 149]:

Аналіз *in vitro* –> Причинно-наслідковий аналіз –> Аналіз систем –> Компютерна модель –> Дослідження на тваринах.

У запропонованому алгоритмі є постійна взаємодія з інформацією, що отримана із досліджень *in vitro,* теоретичними обґрунтуваннями та узагальненнями на рівні цілого організму. Результати експериментів *in vitro* спочатку аналізуються в залежності від відношення доза-відповідь чи причина-наслідок для органів або клітин. Потім ці відношення екстраполюються на рівень усього організму з використанням математичних моделей. Моделі реалізуються і вивчаються за допомогою комп'ютера в імітаційних дослідженнях з метою передбачити динамічні результати *in vivo* дослідної системи в цілому.

В якості обґрунтування для застосування математичного моделювання в електрофізіології [39, 146, 148] наведемо таке:

1. Математичне моделювання та комп'ютерна симуляція дозволяють теоретично оцінити вплив порушень у «калій-натрієвому насосі» на весь організм.

2. Моделі дозволяють теоретично визначити експериментальні можливості електрофізіологічних досліджень.

3. У деяких випадках тільки комп'ютерний експеримент, що базується на конкретних даних, може бути виставлений на досить тривалий час, що дозволить передбачити сторонні ефекти, які виявляться тільки в майбутньому. Такі довгострокові дослідження небезпечних ефектів часто складні для проведення над тваринами через свою тривалість або через больовий ефект, що заподіюється піддослідним тваринам.

4. Комп'ютерне моделювання надає можливість також розрахувати вплив на такі фізіологічні параметри, які важко або неможливо виміряти безпосередньо.

Для створення функціонуючих комп'ютерних моделей для фізіологічних досліджень потрібно:

1) розробити відповідний математичний апарат для опису змодельованих процесів;

2) реалізувати цю модель на мові програмування або із застосуванням електронних таблиць.

План моделювання повинен включати в себе опис апаратних засобів і програмного забезпечення використаного для розробки моделей, методику виконання моделювання.

Розглянемо докладніше питання, що стосується моделювання за допомогою диференціальних рівнянь моделі Ходжкіна-Хакслі.

Для того, щоб описати детерміновані явища, що змінюються в часі, більшість розробників комп'ютерних моделей у фізіології використовують диференціальні рівняння [149]. Застосування апарату диференціальних рівнянь дозволяє в максимальному ступені "узагальнити" програму, навіть якщо немає необхідності використовувати диференціальні рівняння для моделювання. Програмний код, в якому використовуються диференціальні рівняння, є більш легким для читання, але і вимагає великих обчислювальних затрат.

Лінійні системи можуть мати явні розв’язки, які можуть потім бути модифіковані з використанням лінійних операторів, з метою вирішити проблему аналітично. Головна перевага аналітичного розв’язку рівнянь – швидкість обчислень. Методи розв’язання диференціальних рівнянь типу Ходжкіна-Хакслі значно повільніші, ніж методи для розв’язання рівнянь, заданих у явному вигляді. Оскільки змодельована динамічна система типу Ходжкіна-Хакслі – нелінійна (і ця нелінійність має значення в моделюванні), то безумовно повинен бути використаний апарат нелінійних диференціальних рівнянь. Наслідком цього є різке збільшення обчислювальних витрат.

Вибір методу для розв’язання диференціальних рівнянь залежить від конкретної задачі. Якщо інтервал інтегрування досить великий (як це буває в звичайних випадках моделювання), тоді необхідна точність може бути отримана з використанням чисельних методів розв’язування Рунге-Кутта або

Адамса з адаптацією 4-го порядку. Якщо відношення найбільшої змінної до найменшої досить велике (що має місце в електофізілогічному моделюванні), або якщо є велика зміна швидкості компоненти в межах системи, то ця система вимагає дуже специфічних алгоритмів для її розв’язання, таких як, наприклад, алгоритм Gear або Livermore Solver [39, 147, 148].

4.2.1 Аналіз програмного забезпечення, що використовується у електрофізіологічних дослідженнях. Стандартним способом реалізації математичної моделі на комп'ютері є написання спеціальних програм на мовах BASIC, C / C + +, Delphi або на JavaScript. Однак це досить трудомісткий спосіб, тому що доводиться писати велику кількість програмного коду. До переваг методу можна віднести те, що весь процес моделювання контролюється безпосередньо дослідником [150].

Альтернативний варіант – використання електронних таблиць, які підтримують ітеративне розв’язання рівнянь. Тут світовим стандартом є Microsoft Excel.

Для серйозного моделювання із застосуванням складних математичних викладок більш доцільне використання спеціальних математичних пакетів. Найбільш популярні – Matlab, Mathematica, MathCAD, Maple. Їх особливістю є багатий арсенал математичних методів, комп'ютерна графіка та інтерфейс. Однак і тут треба бути готовим до написання програм на вбудованих мовах програмування.

Ці способи реалізації моделей є досить складними для науковців без спеціальної освіти. Тому розроблено ряд програмних продуктів для конкретних задач електрофізіологічного моделювання. Відповідальність за те, що комп’ютерна реалізація електрофізіологічної моделі не містить помилок, лежить на розробнику моделі. Коли розробник моделі пише свою власну програму, слід обгрунтувати прийнятність алгоритму та кроків програмування. Такі ж питання виникають до щойно створеного комерційного або відкритого програмного забезпечення.

Програмне забезпечення для електрофізіологічного моделювання може включати такі програмні продукти:

Biokmod – програма створена за допомогою інструментів Mathematica для розв'язання систем диференціальних рівнянь, задання коефіцієнтів, з додатками для моделювання лінійних і нелінійних біокінетичних систем. Також включені декілька підручників, в яких йде мова про те, як Mathematica може бути застосована в моделюванні електрофізіологічних досліджень.

ScoP (Simulation Control Program) – розробка Simulation Resources, Inc., Redlands, CA. Є програмою інтерактивного керування конструюванням моделей.

1. Stella (Isee Systems, Lebanon, NH (колишня High Performance Systems Inc.)). Це програмне забезпечення Macintosh з інтерактивним графічним інтерфейсом; надає можливість користувачу генерувати моделі з діаграмами, вимагаючи мінімального знання комп'ютерного програмування [151].

Berkely Madonna (Robert Macey і George Oster, University of California at Berkeley, CA). Це програма для розв'язування диференціальних рівнянь загального призначення. Розроблена в Berkeley при сприянні National Science Foundation і National Institutes of Health. Сьогодні вона використовується в навчальних і комерційних закладах для побудови математичних моделей в наукових дослідженнях і навчанні.

CMATRIX (Robert Ball and Sorell L. Schwartz, Georgetown University, Washington, DC). Це система, що дозволяє користувачу створювати компартментні моделі на основі власного біологічного матеріалу, залишаючи програмному забезпеченню побудову та чисельний розв'язок диференціальних рівнянь.

Як видно з наведеного огляду, програмні продукти, що використовуються, не підтримують веб-технології, що створює певні проблеми з їх переносимістю на нові платформи і ширшим використанням.

4.2.2 Програмне середовище реалізоване у вигляді пакету Java-класів [152 – 156] і складається з папок «classes», «src» і компілятора «compile_it». В «classes» знаходиться запускний файл програми «MedicalBiologicalInvestigations», який запускається будь-яким браузером і папка з відкомпільованими класами. В папці «src» знаходяться вихідні коди пакету «medbioinvestigations». У склад пакету входять такі пакети і класи:

– пакет «fde» містить клас DelaySystemSolution, який призначений для отримання чисельного розв'язку функціонально-диференціальних рівнянь (рівнянь з дискретно або неперервно розподіленим запізненням, інтегро-диференціальних рівнянь).

Представимо абстрактні методи даного класу (Додаток В3). Метод «fcn» описує праві частини рівнянь системи (3.1)-(3.5), за допомогою яких будуються графіки.

Пакет graph містить класи, призначені для графічної візуалізації розв'язків рівнянь.

Головним класом пакету є клас GraphConstruction з наступною декларацією:

abstract public class GraphConstruction extends java.applet.Applet implements Runnable

Клас GraphConstruction є бібліотекою машинної графіки, що має багаті візуальні можливості щодо вивчення поведінки показників на найдрібніших інтервалах зміни, одночасного виведення кількох графіків в одній площині і інші.

Основними елементами класу GraphConstruction є:

- змінна m_Graphics типу Thread для підтримки багатопоточності;
- змінна m_fps типу double для швидкості перемальовування графіків;
- змінна m_nAreaNumber для вказування поточного номера графіка;
- булеві змінні m_bLegend для включення в графіки легенд, m_bCaptions – підписів графіків, m_bPunctureLine – режиму пунктирних ліній;

- змінні m_x0 та m_x1 для визначення часових меж зміни параметрів полісу;

- зв’язаний список m_llCursorLocations типу LinkedList для визначення різних меж областей визначення параметрів;

- метрика шрифта m_fm типу FontMetrics;

- координати вказівника мишки на початку та вкінці перетягування m_dimCursorLocBegin та m_dimCursorLocEnd відповідно (типу Dimension);

- булева змінна m_bDrag для визначення режиму перетягування;

- булева змінна m_bBeginDrag для вказування початку перетягування;

- цілочисельні змінні m_nOffset_x та m_nOffset_y для визначення відступу в кожному графіку для побудови осей абсцис та ординат відповідно;

- змінна m_dStep типу double для встановлення кроку сітки по осі абсцис;

- цілочисельна змінна m_nScale для встановлення щільності шкали;

- змінна m_dimDragEvtLoc типу Dimension для збереження координати, пов’язаної з подією перетягування мишкою;

- змінні m_nGraphWidth та m_nGraphHeight для вказування ширини та висоти відповідно для областей побудови графіків;

- змінна m_nGraphCount для вказування кількості параметрів для побудови графіків;

- булеві змінні m_bMarkGraphs та m_bWholeScreenMode встановлення режимів виведення кількох графіків в одній області та повноекранного графіка відповідно;

- клас-аплет m_graphicalSearchValue типу GraphicalSearchValue для пошуку значення параметра для визначеного значення абсциси;

- об’єкт m_image типу Image для збереження позаекранного зображення розміром m_dimImage (використовується метод подвійної буферизації) та асоційований з ним графічний об’єкт m_g типу Graphics.

Основним методом класу GraphConstruction є метод

```
abstract public double f(int i, double x);
```

через який визначаються значення функцій, що виводяться в графічному вигляді.

Ідея методу подвійної буферизації при виведенні графіків полягає в тому, що формування зображення для виводу здійснюється в методі:

public void update(Graphics g)

При цьому сформоване зображення зберігається в буфері пам'яті в об'єкті m_g. Лише після повного поновлення об'єкту m_g здійснюється виклик методу

public void paint(Graphics g)

з виведенням графічного контексту у вікні аплета. Використання методу подвійної буферизації з попереднім формуванням зображення в буфері пам'яті дозволяє уникнути таких небажаних ефектів із виведенням зображення, як мерехтіння через затримки з розрахунком графічних значень. Це має особливе значення при використанні інтерактивності, як наприклад перетягування мишкою для вказування часових проміжків або ж вказування кількох графіків для виведення в одній графічній площині.

Розрахунок зображення в буфері для виведення графіків кількох параметрів (за припущенням) здійснюється викликом методу:

PaintFewGraphs(m_g, m_nGraphCount, m_nGraphWidth, m_nGraphHeight, m_x0, m_x1);

Розрахунок зображення в буфері для виведення графіків в повноекранному режимі здійснюється викликом методу:

PaintDataFrame(m_g, m_nAreaNumber, 0, 0, size().width, size().height, m_x0, m_x1, Color.red);

Для пошуку моменту часу, коли досягається певне значення змінної викликається метод:

PaintGraphicalSearchValue(m_g, m_graphicalSearchValue.graphsNumber, m_graphicalSearchValue.ordinate, 0, 0, size().width, size().height, m_x0, m_x1);

При перетягуванні мишкою з метою вказування нових часових проміжків для виведення параметрів моделі рисується прямокутник:

```
if (m_bDrag & !(m_bMarkGraphs) & !(m_bWholeScreenMode))
                {
                        m_g.setColor(Color.blue);
                        //draw Rectangle
                        int xPoints[] = {m_dimCursorLocBegin.width, m_dimDragEvtLoc.width, m_dimDragEvtLoc.width, m_dimCursorLocBegin.width, m_dimCursorLocBegin.width};
                        int yPoints[] = {m_dimCursorLocBegin.height, m_dimCursorLocBegin.height, m_dimDragEvtLoc.height, m_dimDragEvtLoc.height, m_dimCursorLocBegin.height};
                        m_g.drawPolygon(xPoints, yPoints, 5);
                        m_g.drawString(Double.toString(m_x0 + (m_dimCursorLocBegin.width - (m_dimCursorLocBegin.width / m_nGraphWidth) * m_nGraphWidth - m_nOffset_x) * m_dStep), m_dimCursorLocBegin.width + 1, m_dimCursorLocBegin.height + 1);
                        m_g.drawString(Double.toString(m_x0 + (m_dimDragEvtLoc.width - (m_dimCursorLocBegin.width / m_nGraphWidth) * m_nGraphWidth - m_nOffset_x) * m_dStep), m_dimDragEvtLoc.width + 1, m_dimDragEvtLoc.height + 1);
                }
```

Вкінці методу update здійснюється примусове перерисовування зображення шляхом виклику paint(g).

В методі PaintFewGraphs здійснюється побудова графіків n параметрів. При цьому для побудови кожного окремого графіка викликається метод:

PaintDataFrame(g, nFunctionCounter, nGraphWidth * i, nGraphHeight * j, nGraphWidth, nGraphHeight, x0, x1, Color.red)

Для одночасного виведення кількох графіків в одному графічному полі викликається метод

PaintFewDataFrame(g, ((n) % (size().width / m_nGraphWidth)) * m_nGraphWidth, ((n) / (size().width / m_nGraphWidth)) * m_nGraphHeight, m_nGraphWidth, m_nGraphHeight, x0, x1)

У випадку режиму пошуку моменту часу для досягнення певного значення параметру тут здійснюється виклик методу

PaintGraphicalSearchValue(g, m_graphicalSearchValue.graphsNumber,

m_graphicalSearchValue.ordinate, ((n) % (size().width / m_nGraphWidth)) * m_nGraphWidth , ((n) / (size().width / m_nGraphWidth)) * m_nGraphHeight, m_nGraphWidth, m_nGraphHeight, x0, x1)

В якості прикладу побудови графіків для зміни значень параметрів розглянемо метод

private void PaintFewDataFrame(Graphics g, int nLeftBound, int nUpperBound, int nWidth, int nHeight, double x0, double x1)

призначений для виведення кількох графіків в одній графічній області.

Спочатку розраховується крок сітки по осі абсцис:

m_dStep = (x1 - x0) / (nWidth - m_nOffset_x);

Потім оцінюється множина значень, а саме визначаються мінімальні та максимальні значення по усіх фазових змінних, що одночасно виводитимуться. Визначається крок сітки по осі ординат:

double h = (nHeight - m_nOffset_y) / (ymax - ymin);

Далі здійснюється креслення осей, нанесення шкал, зображення легенди і креслення графіків. При цьому графік будується в ході цикла:

```
for (int ix = 0 ; ix < nWidth - m_nOffset_x; ix++)
                    {
                        xStart = nLeftBound + m_nOffset_x + ix;
                        yStart = nUpperBound + (int)(nHeight - m_nOffset_y
- (f(fl.Data(), x0 + ix * m_dStep) - ymin) * h);
                        xEnd = nLeftBound + m_nOffset_x + ix + 1;
```

```
                    yEnd = nUpperBound + (int)(nHeight - m_nOffset_y
- (f(fl.Data(), x0 + (ix + 1) * m_dStep) - ymin) * h);
                    if(! m_bPunctureLine)
                    {
                        //g.drawLine(xStart, yStart, xEnd, yEnd); //
ordinary line
                        drawBoldLine(g,xStart, yStart, xEnd, yEnd);
                    }
                    else
                    {
                        dLengthRemainder = drawPunctureLine(g,
dLengthRemainder, xStart, yStart, xEnd, yEnd, 5 * (fl.Data() - 1), cl.Data());
                    }
                }
```

4.2.2.1 Програмна реалізація мультиваріативного методу для системи Ходжкіна-Хакслі. Для реалізації методу розроблено пакет Java-класів rule.model. До складу пакету входять класи (рис.4.2):

beans-класи Attribute, Attribute_for_list для роботи з даними відповідних таблиць та Rule – для представлення правил. SQL – запити щодо отримання потрібних даних реалізовано в класах AttributeListPeer та TuplesPeer.

У класі Rule_set зберігається набір навчальних правил. До того ж даний клас безпосередньо реалізує алгоритм послідовного покриття. Клас містить члени: менеджер даних m_dataManager, хеш-таблиці наборів навчальних даних m_htTuples, усіх атрибутів з їх можливими значеннями m_htAtt_vals та безпосередньо множину правил m_htRule_set.

У конструкторі класу Rule_set здійснюється побудова хеш-таблиць m_htTuples та m_htAtt_vals, а також застосування алгоритму послідовного

покриття – через виклик методу Sequential_covering(m_htTuples, m_htAtt_vals). Отримана множина правил виводиться в текстовий файл.

Клас Rule призначений для зберігання окремих правил. Його членами класу є дві хеш-таблиці: m_htAntecedent – для зберігання антеседенту правила та m_htConsequent – для консеквенту. За допомогою методу

public void conjunctCondition(Attribute_for_list attribute, String sAttribute_value)

здійснюється кон'юнкція нової умови до правила. За допомогою методу

public Rule copy()

створюється «глибока» копія правила. При цьому використовується протокол JOS (Java Object Serialization).

Підрахунок кількості позитивних та негативних навчальних наборів здійснюється у методах класу TuplesPeer.

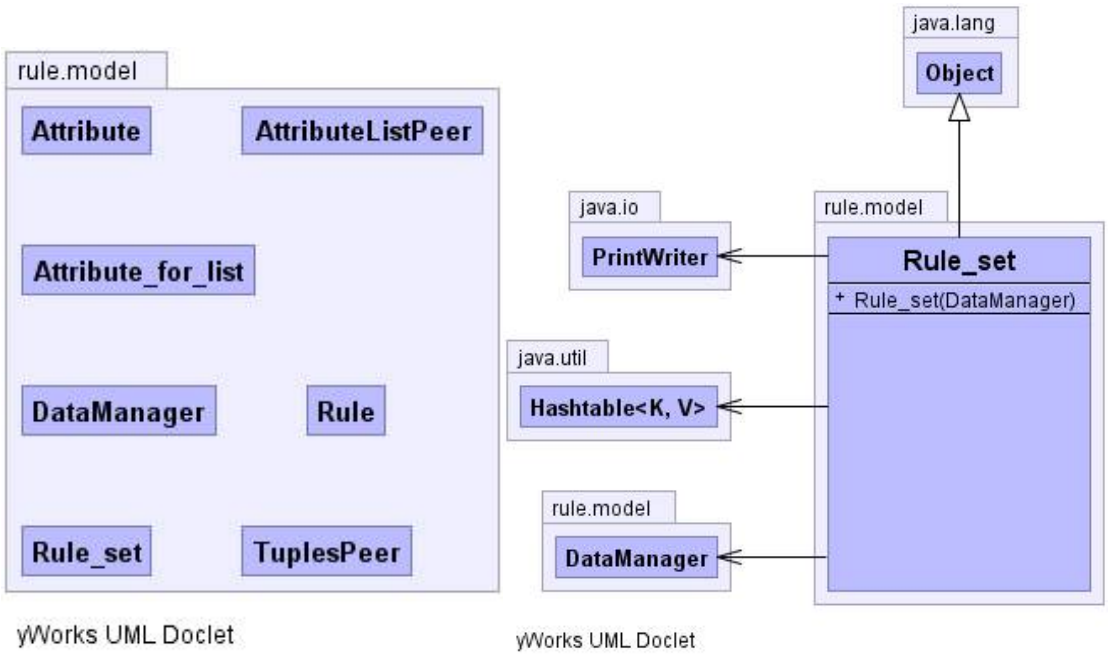


Рис. 4.2. Пакет rule.model

У класі fde.hh.MultiVariateMethod (рис.4.3) здійснюється генерація випадкових значень параметрів (крок 2):

M_x0 = dm.getRandomInitialValues();

M_rateConstants = dm.getRandomRateConstants();

Далі запускається аплет інтегрування системи Ходжкіна-Хакслі. При цьому експерт здійснює вибір форми отриманої траєкторії (крок 3). Після цього запускається крок генерації матриці взаємозв'язків параметрів (крок 4). Зауважимо, що послідовність кроків 2-4 може виконуватися як завгодно багато разів. У будь-який момент користувач може запустити алгоритм послідовного покриття (крок 5):

rule.model.Rule_set rule_set = new rule.model.Rule_set(rule_dataManager, sql0, rules_file_url)

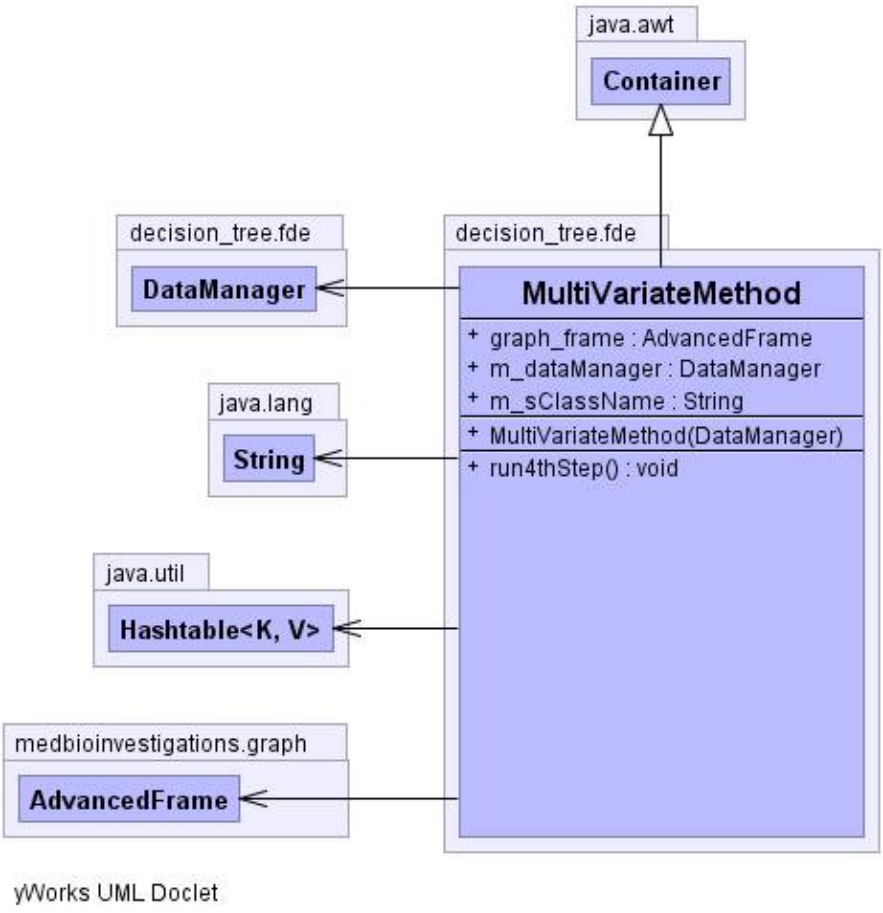


Рис. 4.3. UML – діаграма класу MultiVariateMethod

База даних fde, що використовується в пакеті, реалізована в СУБД MySQL.

Вона включає такі таблиці (рис. 3.7):

attribute – опис атрибутів для побудови класифікаційних правил, тобто взаємозв'язків між початковими значеннями та між швидкісними константами;

categorized_data – навчальні набори, що використовуються в класифікаційному алгоритмі (в даному випадку алгоритму послідовного покриття) і представляють собою матрицю D на четвертому кроці;

init_values_values – матриця згенорованих випадковим чином початкових значень:

$$\begin{pmatrix} V_0^1 & m_0^1 & n_0^1 & h_0^1 \\ V_0^2 & m_0^2 & n_0^2 & h_0^2 \\ \dots & \dots & \dots & \dots \\ V_0^N & m_0^N & n_0^N & h_0^N \end{pmatrix} \in R^{N\times 4}$$

initial_values – опис початкових значень (включаючи мінімальні та максимальні значення);

parameter_kind – вид параметру;

rate_constants – опис швидкісних констант (включаючи мінімальні та максимальні значення);

rate_constants_values – матриця згенерованих випадковим чином швидкісних констант:

$$\begin{pmatrix} g_K^1 & g_{Na}^1 & g_L^1 & V_K^1 & V_{Na}^1 & V_L^1 & C_m^1 & x_m^1 & x_n^1 & x_h^1 \\ g_K^2 & g_{Na}^2 & g_L^2 & V_K^2 & V_{Na}^2 & V_L^2 & C_m^2 & x_m^2 & x_n^2 & x_h^2 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ g_K^N & g_{Na}^N & g_L^N & V_K^N & V_{Na}^N & V_L^N & C_m^N & x_m^N & x_n^N & x_h^N \end{pmatrix} \in R^{N\times 10}$$

4.2.3 Особливості програмної реалізації системи Ходжкіна-Хакслі. Пакет rule.model використаний для класу систем Ходжкіна-Хакслі. Для цього модель на основі функціонально-диференціальних рівнянь повинна бути реалізована у вигляді відповідного пакету Java-класів. Прикладом такого пакету

у випадку системи Ходжкіна-Хакслі є пакет medbioinvestigations.hodgkin_hyxley (рис. 3.7).

Для інтеграції з пакетом decision_tree.fde.hh в класі Hodgkin_HuxleyGraph, що здійснює графічну візуалізацію моделі, поряд з існуючим додано новий конструктор, що використовує посилання за значенням на інстанцію класу MultiVariateMethod. У цьому конструкторі додатково створюється об'єкт класу JComboBox, що дозволяє вибирати форму траєкторії та запускати виконання 4-го кроку методу:

```
String[] classStrings = {"subclinical","chronic","acute","lethal"};
JComboBox m_jcbClassName = new JComboBox(classStrings);

m_jcbClassName.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e)  {
        JComboBox jcbClass = (JComboBox) e.getSource();
        m_sClassName = (String)jcbClass.getSelectedItem();
        mvm.m_sClassName = m_sClassName;
        ((AdvancedFrame)getParent()).dispose();
        mvm.run4thStep();
    }
});
```

Також слід внести відповідні зміни у таблиці бази даних hh в такому порядку:

- описати усі початкові значення та швидкісні константи в таблицях initial_values та rate_constants відповідно;

- описати залежності між початковими значеннями та між швидкісними константами, які досліджуватимемо, у таблиці attribute;

- у таблиці categorised_data створити поля у відповідності з даними таблиці attribute.

Метод реалізовано в середовищі розробки Netbeans на мові програмування Java. Базу навчальних даних розгорнуто на сервері MySQL. На рис.4.4 представлено концептуальну модель. У класі DecisionTree безпосередньо реалізовано метод індукції дерева рішень. У клас DataManager надходять виклики від DecisionTree на виконання запитів до бази даних mysql щодо отримання навчальних даних.

База даних mysql складається з двох таблиць – таблиці attribute, призначеної для зберігання інформації про атрибути та таблиці categorized_data – для наборів навчальних даних. Структура таблиць на мові SQL для Прикладу наведена нижче:

```
CREATE TABLE mysql.attribute (
        id integer not null unique,
        attribute_name varchar(25),
        attribute_field_name varchar(25),
    primary key (id)
  ) ENGINE=InnoDB;
CREATE TABLE mysql.categorised_data (
        id integer not null unique,
        A1 varchar(12),
        A2 varchar(8),
        A3 varchar(7),
        A4 varchar(7),
        A5 varchar(7),
        class varchar(8),
    primary key (id)
  ) ENGINE=InnoDB;
```

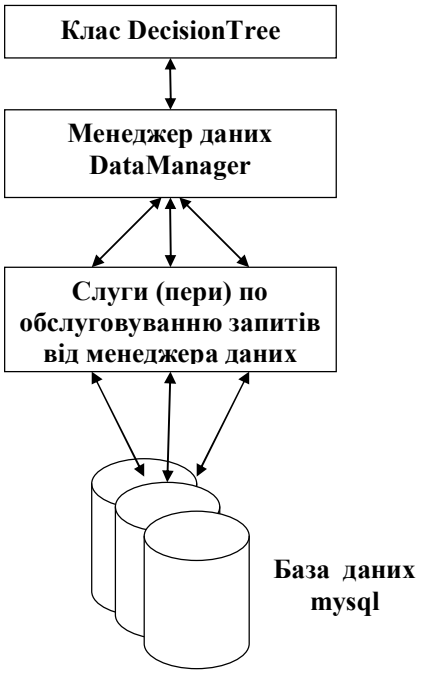


Рис. 4.4. Концептуальна модель інформаційної системи з індукцією дерева рішень

Програмні класи проекту включено до пакету decision_tree.model. Сюди входять beans-класи Attribute, Attribute_for_list та CategorisedData для роботи з даними відповідних таблиць. SQL-запити щодо отримання відповідних даних, включаючи розрахунки інформаційних показників реалізовано в класі AttributeListPeer.

Клас DecisionTree є нащадком класу DefaultTreeModel пакету javax.swing.tree. Він має два елементи класу: m_dataManager – менеджер даних та m_htAttribute_list – хеш-таблиця із списком атрибутів.

Хеш-таблиця із списком атрибутів (у методах класу DecisionTree виступає під назвою htAttribute_list) створюється для кожного вузла дерева рішень. Вона має два призначення – поряд із списком включених для даного вузла атрибутів зберігати умови поділу (splitting conditions), які перейшли до даного вузла від вузлів-батьків. Кожен вузол дерева рішень є об’єктом класу

DefaultMutableTreeNode. В якості об’єкта кожен вузол зберігає об’єкт класу NodeObject, декларація якого наведена нижче:

```
class NodeObject {
   Attribute attribute;
   Hashtable htAttribute_list;
   String splitting_criterion;
   String sLabel;
   public String toString() {
      if (splitting_criterion.matches("")) { return sLabel; }
      else return "if '" + splitting_criterion + "' then '" + sLabel + "'"; }
}
```

Тут attribute – атрибут, який повертається методом Attribute_selection_method, splitting_criterion – умова поділу, яка переходить від батьківського вузла, sLabel – надпис на вузлі. Хеш-таблиця htAttribute_list використовується для побудови наборів навчальних даних D_j для кожного із вузлів і має таку структуру:

Тип ключа	int
Тип об’єкта	Attribute_for_list
Структура об’єкта	Attribute attribute; Hashtable htSplitting_outcomes; String splitting_criterion; boolean included;

Тут included – булева змінна-прапорець належності атрибуту attribute до списку атрибутів даного вузла. Можна показати, що коли included=true, то вузол з назвою attribute є для даного вузла дочірнім (на певному нижчому рівні ієрархії). У випадку, коли атрибут attribute не входить до списку атрибутів для даного вузла (included=false), то вузол з назвою attribute є батьківським (на

певному рівні ієрархії), а в змінній splitting_criterion зберігається умова поділу, якій підлягає даний вузол відносно батьківського вузла attribute.

Хеш-таблиця htSplitting_outcomes містить усі можливі наслідки (умови поділу) щодо атрибуту attribute.

Метод Generate_decision_tree є безпосередньою реалізацією методу індукції дерева рішень.

Заголовок методу має вигляд:

private DefaultMutableTreeNode Generate_decision_tree (Hashtable htAttribute_list, DefaultMutableTreeNode dmtnSubroot, String splitting_criterion).

За аргументи у методі використано кореневий вузол дерева, список пов'язаних з ним атрибутів: htAttribute_list та умову поділу splitting_criterion. В якості значення метод повертає дочірній вузол типу: DefaultMutableTreeNode. Шляхом рекурсивного виклику методу Generate_decision_tree будується дерево рішень.

З метою візуалізації представлення дерева використано клас javax.swing.JTree. При цьому дерево рішень створюється виводиться за допомогою операторів:

dtDecision_tree = new DecisionTree(dmtnRoot, dataManager, htAttribute_list);

jTree1.setModel(dtDecision_tree).

4.2.3.1 SQL-реалізація розрахунку інформаційних показників приросту інформації. Ключовим в реалізації методу Attribute_selection_method є розрахунок інформаційних показників $Info(D_j)$ та $Info_{A_i}(D_j)$ на j-му кроці рекурсії для атрибута A_i. Так показник $Info(D_j)$ розраховується методом:

public double getInfo$D$ (DataManager dataManager, Hashtable htAttribute_list).

Зауважимо, що множина наборів D_j тут описується хеш-таблицею htAttribute_list, з якої отримуємо перелік включених атрибутів sAttribute_list та умов поділу sConditions.

Мова структурованих запитів SQL має досить гнучкі засоби для реалізації алгоритмів в галузі машинних знань. Так використавши вкладені запити, псевдоніми та агрегативні функції можна розрахувати $Info(D_j)$ в результаті виконання такого SQL-запиту:

String sql = "select -SUM((Alias1.Ci/Alias2.D)*(LOG(Alias1.Ci/Alias2.D)/LOG(2))) from " +

"(select SUM(1) as Ci from (select " + sAttribute_list + ",class from categorised_data " + (sConditions.matches("")?"": " where " + sConditions) + ")Alias3 group by Alias3.class)Alias1, " +

"(select SUM(1) as D from (select " + sAttribute_list + " from categorised_data " + (sConditions.matches("")?"": " where " + sConditions) + ")Alias4)Alias2".

Показник $Info_{A_i}(D_j)$ розраховується методом:

public static double getInfo_AD(DataManager dataManager, int i, Hashtable htAttribute_list).

Остаточно $Info_{A_i}(D_j)$ обчислюється в результаті виконання такого SQL-запиту:

String sql = "select SUM((Alias1.Dj/Alias2.D)*Alias3.InfoDj) from " + "(select SUM(1) as Dj from (select * from categorised_data" + (sConditions.matches("")?"": " where " + sConditions) + ")Alias6 group by Alias6." + ((Attribute_for_list)htAttribute_list.get(A)).attribute.getAttributeFieldName() + ")Alias1, " + "(select SUM(1) as D from (select " + sAttribute_list + " from categorised_data " + (sConditions.matches("")?"": " where " + sConditions) + ")Alias7)Alias2, " + "(select -SUM((Alias4.Ci/Alias5.D)*(LOG(Alias4.Ci/Alias5.D)/LOG(2))) as InfoDj from " + "(select SUM(1) as Ci from (select " + sAttribute_list + ",class from categorised_data " + (sConditions.matches("")?"": " where " + sConditions) + ")Alias8 group by Alias8.class)Alias4, " + "(select SUM(1) as D from (select " + sAttribute_list + " from categorised_data " + (sConditions.matches("")?"": " where " + sConditions) + ")Alias9)Alias5)Alias3".

4.2.3.2 SQL-реалізація розрахунку відношення інформаційних показників приростів інформації. При реалізації методу Attribute_selection_method слід розрахувати інформаційні показники $Info(D_j)$, $Info_{A_i}(D_j)$ та $SplitInfo_{A_i}(D_j)$ на j-му кроці рекурсії для атрибута A_i.

Множина наборів D_j тут описується хеш-таблицею htAttribute_list, з якої отримуємо перелік включених атрибутів sAttribute_list та умов поділу sConditions.

Використавши вкладені запити, псевдоніми та агрегативні функції можна розрахувати показник інформації поділу $SplitInfo_{A_i}(D_j)$ в результаті виконання такого запиту:

sql = "select -SUM((Alias1.Dj/Alias2.D)*(LOG(Alias1.Dj/Alias2.D)/LOG(2))) from (select SUM(1) as Dj from (select * from categorised_data " + (sConditions.matches("")?"": " where " + sConditions) + ")Alias3 group by Alias3." + ((Attribute_for_list)htAttribute_list.get(A)).attribute.getAttributeFieldName() + ")Alias1, (select SUM(1) as D from (select " + sAttribute_list + " from categorised_data " + (sConditions.matches("")?"": " where " + sConditions) + ")Alias4)Alias2".

4.3 Оцінювання обчислювальної складності з метою якісного аналізу моделі Ходжкіна-Хакслі

4.3.1 Проблема обчислювальної складності алгоритму індукції дерева рішень. Оцінка складності виконання алгоритму. Як показано в роботі [157, 158] обчислювальна складність для індукції дерева рішень для множини з n навчальних наборів, що використовують p атрибутів, $T(n)$ виражається таким рекурентним співвідношенням:

$$T(n) = \begin{cases} b \equiv const, & if \quad n = 1 \\ 2T(n/2) + pn, & if \quad n > 1 \end{cases} \tag{4.1}$$

Зауважимо, що дане співвідношення відображає оптимальне значення обчислювальної складності, яке має місце у випадку добре збалансованої множини навчальних наборів. У такому разі з (4.1) маємо:

$$T(n) = pn \sum_{i=0}^{\log_2 n} 1^i = pn \log_2 n . \tag{4.2}$$

На основі моделі (2.1) – (2.4) було проведено чисельний експеримент з метою з'ясування узгодженості часу побудови дерева рішень з оцінкою (4.2). Використовувалась система на основі процесора Celeron(R) Dual-Core CPU T3300 @ 2.00 GHz та 2 Гб RAM. Результати представлені на рис. 4.5.

З рисунка 4.5. бачимо певний розкид в часі індукції дерева рішень залежно від величини n, що по'язано в першу чергу із незбалансованістю множини навчальних наборів. Дійсно, застосування методу Монте-Карло гарантує рівномірне представлення усіх початкових значень та швидкісних констант. У той же час це не гарантує рівномірного представлення значень атрибуту класу, від чого залежить виконання порядок редукційного алгоритму індукції дерева рішень.

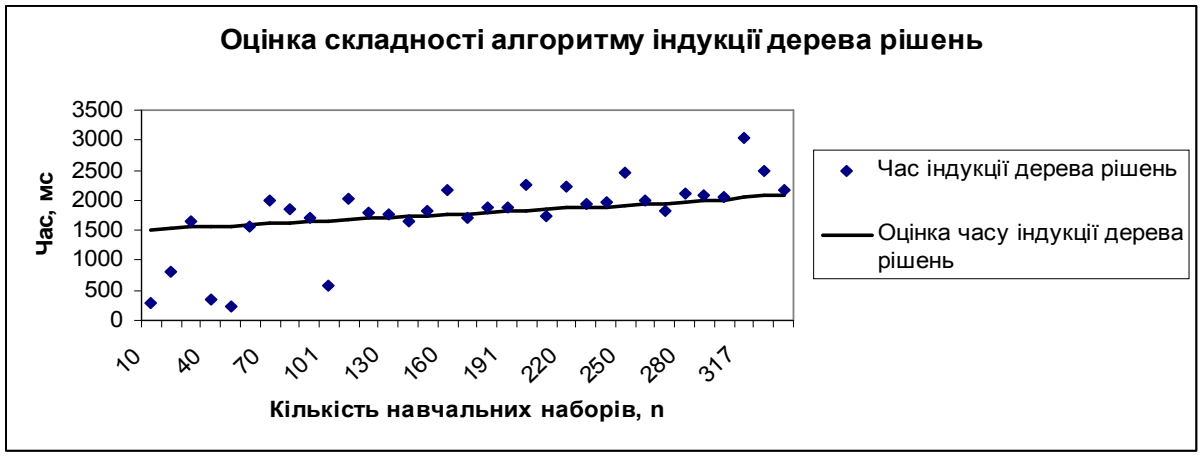


Рис. 4.5. Графік оцінки складності алгоритму індукції дерева рішень

Побудоване дерево рішень для $n=17$ наведене на рисунку 4.6. Час побудови дерева рішень 1402 мс.

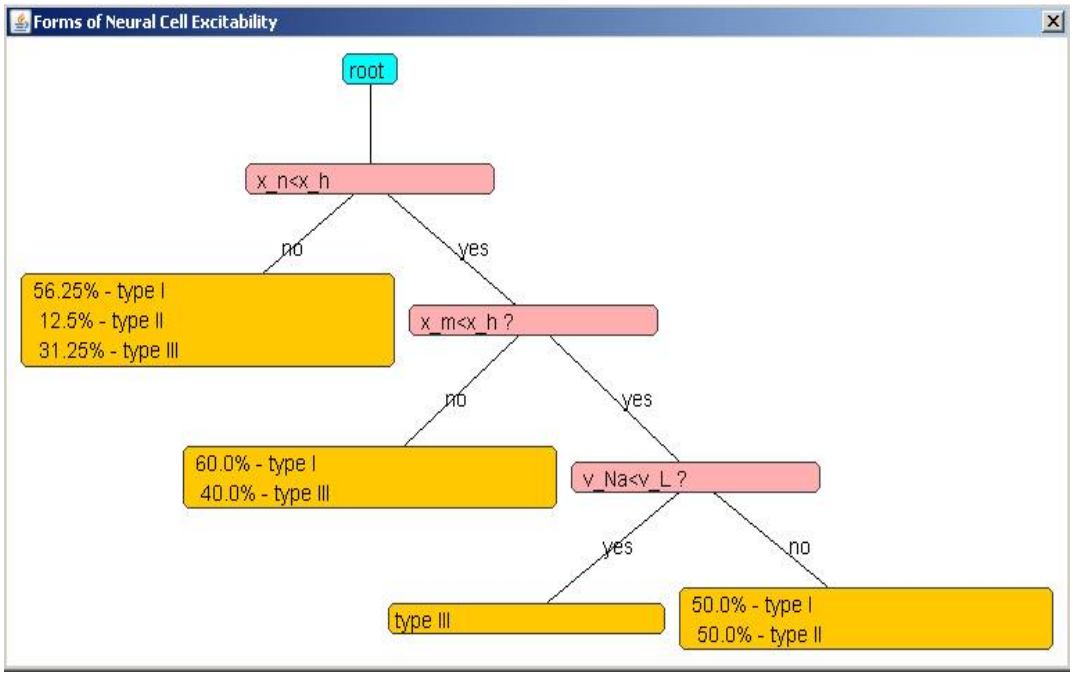


Рис. 4.6. Дерево рішень

Експеримент проводили також змінюючи кількість атрибутів p. Дерева рішень, побудовані для кожного значення p, наведені в додатку.

На рис.4.7. і 4.8. наведені оцінки часу індукції дерева рішень згідно (4.2).

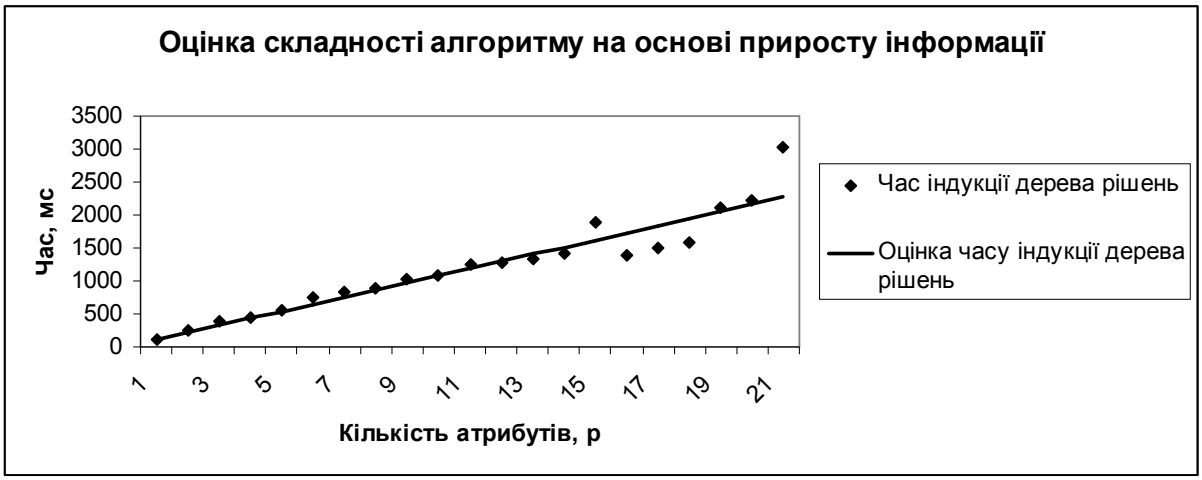


Рис. 4.7. Графік оцінки складності алгоритму на основі приросту інформації

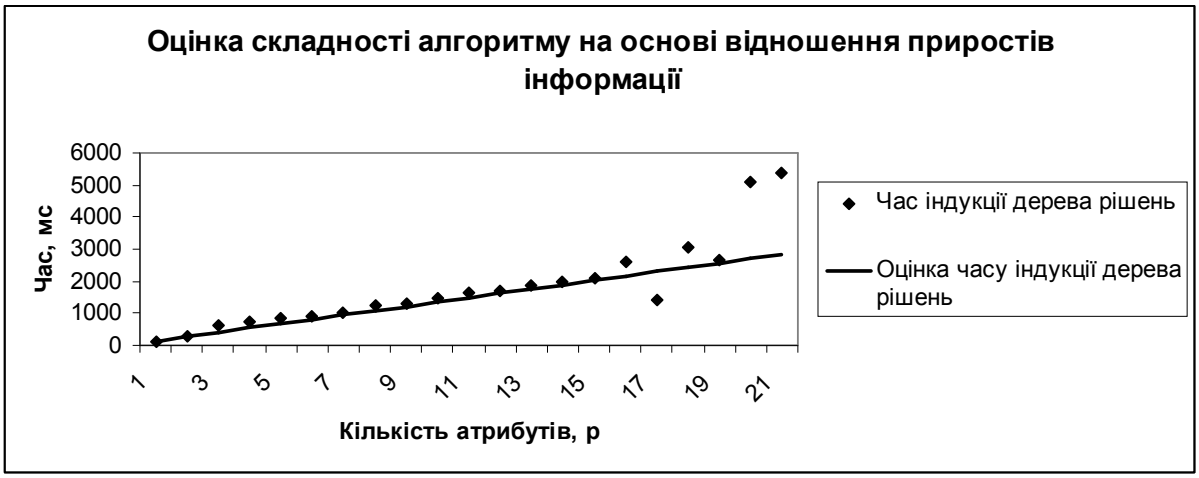


Рис. 4.8. Графік оцінки складності алгоритму на основі відношення приростів інформації

Таблиця 4.1. Залежність часу індукції дерева рішень від кількості атрибутів p (алгоритм на основі приросту інформації).

p	Час індукції дерева рішень, мс	p	Час індукції дерева рішень, мс
1	122	11	1238
2	252	12	1282
3	392	13	1342
4	450	14	1424
5	550	15	1885
6	762	16	1402
7	820	17	1502
8	900	18	1585
9	1030	19	2110
10	1080	20	2212
-	-	21	3033

Таблиця 4.2. Залежність часу індукції дерева рішень від кількості атрибутів p (алгоритм на основі відношення приростів інформації).

p	Час індукції дерева рішень, мс	p	Час індукції дерева рішень, мс
1	100	11	1623
2	290	12	1724
3	610	13	1847
4	740	14	1981
5	832	15	2092
6	930	16	2595
7	1002	17	1390
8	1242	18	3033
9	1321	19	2652
10	1472	20	5096
-	-	21	5380

4.3.2 Оцінювання складності виконання алгоритму на основі методу послідовного покриття. З аналізу алгоритму послідовного покриття бачимо, що обчислювальна складність визначається добутком кількості можливих значень атрибуту класу $K = 3$ (кількість ітерацій зовнішнього циклу) та обчислювальної складності процедури *Добути_одне_правило (D, Att_vals, c)*, яка виконується всередині кожного циклу.

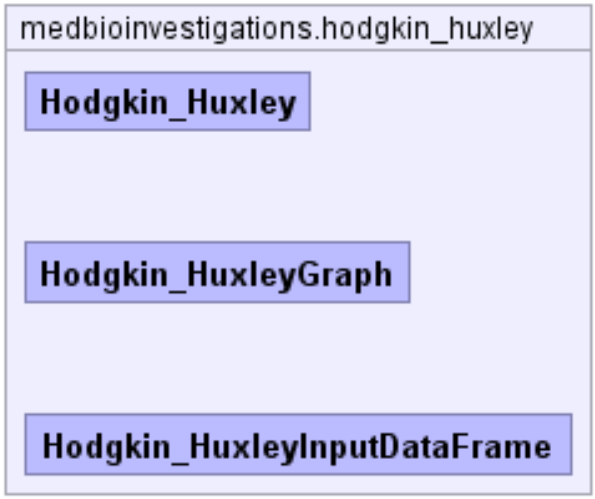


Рис. 4.9. Пакет medbioinvestigations.hodgkin_huxley

Процедура *Добути_одне_правило (D, Att_vals, c)* включає виконання $p = 18$ ітерацій. На кожній ітерації для певного атрибуту A_i проводиться розрахунок міри $FOIL\_Gain$ для кожного з $K_i = 2$ значень атрибуту. Тобто внутрішнє тіло цикла в процедурі *Добути_одне_правило (D, Att_vals, c)* виконується $\sum_{i=1}^{p} K_i = 36$ разів. Міра $FOIL\_Gain$ обчислюється в результаті 3-х SQL-запитів, складність яких можна оцінити величиною $O(\log(N))$ (див. документацію до MySQL 5.0 – http://dev.mysql.com/doc/refman/5.0/en/select-speed.html). Отже, вцілому процедура *Добути_одне_правило (D, Att_vals, c)* має обчислювальну складність $O\left(\sum_{i=1}^{p} K_i \times \log(N)\right) = O(36 * \log(N))$.

Підсумовуючи, маємо обчислювальну складність всього алгоритму послідовного покриття порядку

$$O\left(K \times \sum_{i=1}^{p} K_i \times \log(N)\right) = O(108 * \log(N)) = O(\log(N)). \qquad (4.3)$$

На основі моделі (2.1) – (2.4) було проведено чисельний експеримент з метою з'ясування узгодженості часу побудови набору класифікаційних правил з оцінкою (4.3). Використовувалась система на основі процесора Celeron(R) Dual-Core CPU T3300 @ 2.00 GHz та 2 Гб RAM. Результати представлені на рис. 4.3.

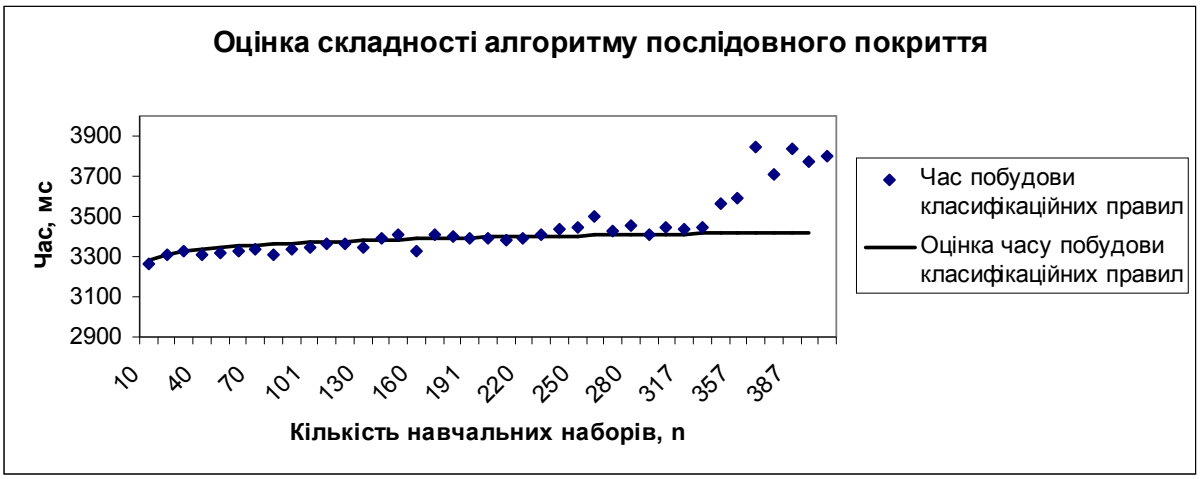


Рисунок 4.10. Графік оцінки складності алгоритму послідовного покриття

З рисунка 4.10. бачимо значне відхилення від оцінки часу побудови класифікаційних правил при $N > 350$, що по'язано із збільшенням обчислювальних ресурсів.

4.3.2.1 Оптимізація побудови копій класифікаційних правил. Як зазначалося вище, з метою створення «глибоких» копій об'єктів в програмі використано технологію Java Object Serialization (JOS). Це – загальний підхід, який полягає в тому, що відбувається записування об'єкта до масиву з використанням ObjectOutputStream а згодом створення копії об'єкта за допомогою ObjectInputStream. В результаті створюється повністю окремий об'єкт з повністю відмінними об'єктами, на які він посилається. Саме такий підхід первинно було використано в програмній реалізації алгоритму.

На жаль тут виникаються проблеми, а саме:

- даний метод працює лише, коли об'єкти, що копіюються, а також об'єкти, на які йдуть прямі або непрямі посилання, підтримують серіалізацію. Тобто вони реалізують інтерфейс java.io.Serializable. На щастя досить лише декларації implements java.io.Serializable;

- технологія Java Object Serialization є повільною і її використання для створення глибокої копії вимагає як серіалізації так і десеріалізації;

- реалізація потоку байтового масиву, що входить до пакету java.io розроблена для досить загального використання для даних різних розмірів і для

забезпечення безпеки в багатопотокових середовищах. Ці характерні особливості, однак, уповільнюють ByteArrayOutputStream і меншою мірою ByteArrayInputStream.

З метою вирішення певних з перелічених проблем (особливо третьої) використовуємо підхід, запропонований у роботі: [158] і який полягає в альтернативних реалізаціях класів ByteArrayOutputStream та ByteArrayInputStream, що робить три простих оптимізації:

- ByteArrayOutputStream за припущенням починається з 32-х байтного масиву для виводу. Далі при запису контенту до потоку розмір масиву при потребі збільшується (або до затребованого розміру, або розмір просто подвоюється). Отже, первинний розмір масиву в 32 байти означає, що створюється багато малих масивів, які потім копіюються і зрощуються при записі даних. Отже, є проста оптимізація – створити масив з більшим початковим розміром;

- усі методи класу ByteArrayOutputStream є синхронізованими. В цілому це правильно, але можна бути певними, що лише один потік має доступ до ByteArrayOutputStream. Вилучення синхронізації дасть певне пришвидшення. Методи класу ByteArrayInputStream залишаються й надалі синхронізованими;

- метод toByteArray() створює і повертає копію байтового масиву з потоку. Це проста ідея, яка полягає в тому, що в противному випадку, коли ми створюємо інший окремий байтовий масив для копіювання в нього, то йде сповільнення зарахунок виконання додаткової роботи.

Таким чином даний альтернативний підхід було використано в методі обчислення міри private double FOIL_Gain(…), а саме копія правила створюється викликом спеціально створеного методу:

rule_prime = (Rule)rule.copyOptimized();

Провівши чисельний експеримент (рис.4.8) встановлено значення оцінки часу виконання алгоритму:

$$21. \times K \times \sum_{i=1}^{p} K_i \times \log(N) + 1550 \quad (4.5)$$

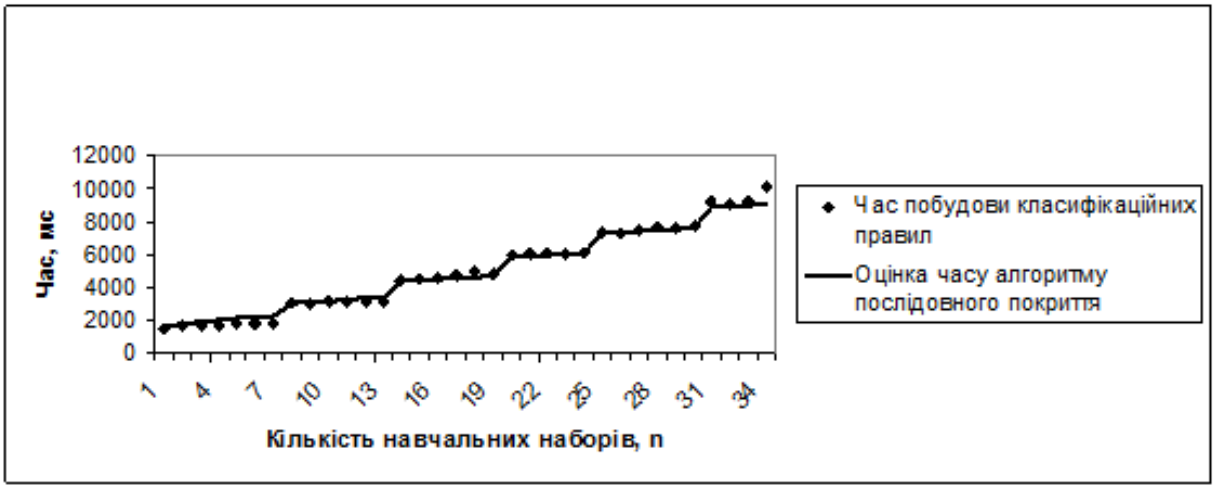


Рис. 4.11. Графік порівняння результатів чисельного експерименту на основі оптимізованого копіювання правил з оцінкою складності алгоритму (3)

Отже, вцілому бачимо певну оптимізацію часу виконання алгоритму (рис.4.12), яка відчутніше проявляється при збільшенні обсягу наборів навчальних даних.

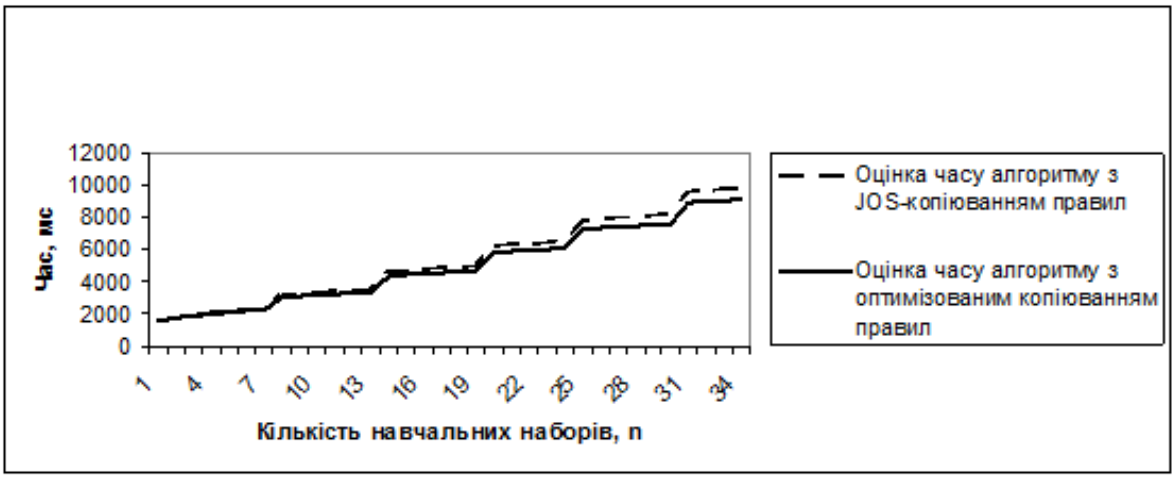


Рис. 4.12. Грфік порівняння оцінок складності алгоритмів послідовного покриттям (JOS-технологія та оптимізоване копіювання правил)

Остаточно побудований набір класифікаційних правил для $n = 397$ наведений нижче.

IF x_m<x_h AND x_n>x_h THEN class=type I

IF n_0<h_0 AND x_n<x_h THEN class=type II

IF v_Na>v_L AND x_m>x_h AND x_n<x_h THEN class=type III

Ряд умов в антецеденті мають певний біологічний зміст.

Отже, існує чимало реалізацій алгоритму послідовного покриття побудови класифікаційних правил, зокрема у складі програм Mathcad, StatSoft Statistica і ін. Суттєвою перевагою даної реалізації є те, що алгоритм розвинено до Інтернет-проекту, де до формування навчальних наборів залучаються експерти з Інтернет-доступом, що значно розширює та балансує множину навчальних наборів.

Висновки до четвертого розділу

Розроблено програмне середовище дослідження розвитку електрофізіології на основі системи Ходжкіна-Хакслі у вигляді бібліотеки Java-класів. В перспективі таке програмне середовище повинне бути орієнтоване на побудову і дослідження електрофізіологічних моделей в усіх аспектах фізичного впливу на біологічну тканину. Здійснено апробацію побудованої математичної моделі шляхом порівняння чисельних розрахунків та експериментально отриманих даних.

Результати четвертого розділу опубліковано в роботі [81 – 83].

ВИСНОВКИ

У дисертаційній роботі розв’язано важливе наукове завдання розробки методів і засобів математичного та комп'ютерного моделювання, обчислювальних методів, призначених для створення апаратно-програмних

засобів моделювання й обчислення для електрофізіологічних процесів збудливості клітин:

1. Розвинуто теорію математичного моделювання електрофізіологічних процесів збудливості клітин шляхом розробки нових алгоритмів дослідження стійкості (дослідження нелінійної динаміки в системі Ходжкіна-Хакслі на основі методу експонент Ляпунова), керування нелінійною динамікою (встановлено необхідні умови оптимальності для керування біфуркацією, що виникає в системі Ходжкіна-Хакслі при зміні прикладеного струму), а також мультиваріативних методів якісного аналізу систем Ходжкіна-Хакслі.

2. Розвинуто та ефективно використано методи обчислювальної математики стосовно вирішення проблем створення і дослідження нових обчислювальних методів і алгоритмів, що враховують особливості електрофізіологічних процесів збудливості клітин, забезпечивши створення ефективних програмних засобів комп'ютерної реалізації систем типу Ходжкіна-Хакслі. Розроблено прямий чисельний метод оптимального керування біфуркацією, що виникає в системі Ходжкіна-Хакслі при зміні прикладеного струму.

3. Розвинуто теорію побудови програмних систем моделювання, а також систем, методів та засобів напівнатурного моделювання електричної активності клітинних мембран, включаючи інформаційні технології їх використання при проведенні електрофізіологічних досліджень. Розроблено програмне середовище якісного аналізу моделі електричної активності нервових клітин.

4. Розроблено новий метод організації та оптимізації процесів моделювання збудливості клітин на основі системи Ходжкіна-Хакслі, включаючи процеси підготовки первинної інформації, визначення складу та структури, настроювання та верифікації, перевірки та забезпечення якості комп’ютерних моделей, дослідження моделей для різних типів збудливості клітин, інтерпретації результатів моделювання. Розроблено та досліджено оцінки обчислювальної складності для мультиваріативних методів якісного

аналізу системи Ходжкіна-Хакслі з метою класифікації різних типів збудливості нервових клітин із застосуванням алгоритмів послідовного покриття та індукції дерева рішень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Б. Н. Бублик, В. Я. Данилов та А. Г. Наконечный, *Некоторые задачи наблюдения и управления в линейных системах.* Київ, Україна: УМК ПО, 1988, с. 190.

2. А. П. Власюк та П. М. Мартинюк, *Математичне моделювання консолідації ґрунтів при фільтрації сольових розчинів в неізотермічних умовах.* Рівне, Україна: Вид-во НУВГП, 2008.

3. Ф. Г. Гаращенко та В. В Пічкур, *Прикладні задачі теорії стійкості.* Київ, Україна: Навчальний посібник. ВПЦ ”Київський університет”, 2014, с. 125.

4. Ф. Г. Гаращенко, В. Т. Матвієнко, В. В Пічкур та І. І. Харченко, *Диференціальні рівняння, варіаційне числення та їх застосування.* Київ, Україна: Навчальний посібник, ВПЦ ”Київський університет”, 2015, с. 271.

5. Ф. Г. Гаращенко та Л. А. Панталієнко *Аналіз та оцінка параметричних систем.* Київ; МСДО України, 1995, с.140.

6. Б. Н. Бублик, Ф. Г. Гаращенко та Н. Ф. Кириченко, *Структурно-параметрическая оптимизация и устойчивость динамики пучков.* Київ, Україна: Наук.думка, 1985, с. 304.

7. О. М. Башняков, Ф. Г. Гаращенко та В. В. Пічкур, *Практична стійкість та структурна оптимізація динамічних систем.* Київ, Україна: ВПЦ ”Київський університет”, 2000, с. 97.

8. Ф. Г. Гаращенко, Н. Ф. Кириченко та інші, *Сучасні методи та інформаційні технології математичного моделювання, аналізу і оптимізації складних систем*: Монографія. Київ, Україна: ВПЦ ”Київський університет”, 2006, с. 200.

9. Ф. Г. Гаращенко та Л. А. Панталієнко, *Аналіз та оцінка параметричних систем.* Київ, Україна: МСДО України, 1995, с. 140.

10. Ю. М. Ермольев, В. П. Гуленко та Т. И. Царенко, *Конечно-разностный метод в задачах оптимального управления.* Київ, Україна: Наук.думка, 1978, с. 164.

11. С. И. Доценко, О. К. Закусило, ”Задача оптимального вибора с ”заминированними” обьектами”, *Кибернетика и вычислительная техника,* Вып.170, с. 51 – 58, 2012.

12. М. З. Згуровский та В. С. Мельник, *Нелинейный анализ и управление бесконечномерными системами.* Київ, Україна: Наук. думка, 1999, с. 630.

13. M. Z. Zgurowsky and N. D. Pankratova, *System Analysis: Theory and Applications*. Springer, 2007, p. 448.

14. В. М. Кунцевич та М. М. Лычак, ”Об оптимальном управлении динамическими объектами в условиях неопределенности”, *Автоматика,* №8, с. 35 – 45, 1978.

15. В. М. Кунцевич та М. М. Лычак, *Синтез оптимальных и адаптивных систем управления. Игровой поход.* Київ, Україна: Наук. думка, 1985, с. 245.

16. В. М. Кунцевич та А. В. Кунцевич, ”Активная идентификация и управление при ограниченных шумах (возмущениях)”, *Кибернетика и системный анализ, №*1, с. 147 – 157, 2000.

17. В. М. Кунцевич та А. А. Чикрий, ”Развитие теории автоматического управления в Институте кибернетики имени В. М. Глушкова и институтах Кибцентра НАН Украины”, *Проблемы управления и информатики*, № 4, с. 7 – 21, 2003.

18. А. В. Кунцевич, ”Анализ и синтез оптимальных дискретных систем управления при ограниченных помехах”, *Кибернетика и системный анализ,* № 5, с. 21 – 27, 1996.

19. V. M. Kuntsevich, A. V. Kuntsevich, *Robust stability of linear discrete-time dynamic systems.* Modeling Techniques for Uncertation Systems. Boston; Basel; Berlin: Birkhauser, 1992, p. 85 – 204.

20. V. M. Kuntsevich, *Set-valued estimation of state and parameter vectors within adaptive control systems.* Bounding Approaches to System Identification. Mew York; London: Plenum Press, 1996, p. 565.

21. С. И. Ляшко, *Обобщенное управление линейными системами.* Київ, Україна: Наукова Думка, 1998, с. 500.

22. S. I. Lyashko, *Generalized Optimal Control of Linear Systems with Distributed Parameters.* Dordrecht, Boston, London:Kluwer Academic Publishers, 2002, p. 455.

23. С. И. Ляшко, Д. А. Номировский, Ю. И. Петунин та В. В. Семенов, *Двадцатая проблема Гильберта (обобщенные решения операторных уравнений).* Москва, Санкт-Петербург: Диалектика, 2009, с. 190.

24. О. Г. Наконечний, О. М. Трофимчук, І. В. Трофімова та Д. І. Чер, *Моделювання та аналіз глобальних біосферних процесів.* Київ, Україна: ВПЦ ”Київський університет”, 2002, с. 93.

25. А. Г. Наконечный, *Минимаксное оценивание функционалов от решений вариационных уравнений в Гильбертовых пространствах.* Київ, Україна: КГУ, 1985, с. 83.

26. Ю. М. Онопчук, ”Математическое моделирование гипоксии при анемии”, *Компьютерная математика,* №1, с. 113 – 121, 2007.

27. П. В. Білошицький, О. М. Ключко, Ю. М Онопчук та А. З. Колчинська ”Результати вивчення вищої нервової діяльності українськими вченими в Приельбруссі” // Вісник НАУ, № 2, с. 105 – 115, 2009.

28. І. В. Сергієнко, І*нформатика в Україні: Становлення, розвиток, проблеми.* Київ, Україна: Наук. думка, 1999, с. 354.

29. Ю. Г. Кривонос, И. И. Матичин та А. А. Чикрий, *Динамические игры с разрывными траекториями.* Київ, Україна: Наукова думка, 2005, с. 220.

30. А. А. Чикрий та С. Д. Эйдельман, ”Обобщенные матричные функции Миттаг-Леффлера в игровых задачах для эволюционных уравнений дробного порядка”, *Кибернетика и системный аналіз,* №3, с. 3 – 32, 2000.

31. Н. М. Амосов, *Кибернетика биологическая.* Киев, Украина: Гл.ред. УСЭ, Энциклопедия кібернетики, 1974, т.1, с. 123 – 125.

32. Н. М. Амосов, *Моделирование сложных систем.* Киев, Украина: Наук. думка, 1968, с. 88.

33. Н. М. Амосов, ”Регулирующие системы организма”, *Энциклопедия кібернетики,* Киев, Украина: Гл. ред. УСЭ, т.1, с. 123 – 125, 1974.

34. Ю. Г. Антомонов, ”Моделирование биологических систем”, *Справочник* Київ, Україна: Наук. думка, с. 260, 1977.

35. Ю. Г. Антомонов *Системы. Сложность. Динамика.* Киев, 1969.

36. Ю. Г. Антомонов *Принципы нейродинамики.* Киев, Украина: Наукова думка. 1974.

37. Ю. Антомонов, Т. Попов та А.Котова. *Приложна биокибернетика.* София: Медицина и физкултура, 1978.

38. Ю. Антомонов*, А.Котова, та В.Зотов Основы биологической информатики:* в 2 ч., Киев: Медэкол. 1996.

39. Н. Бейли, *Математика в биологии и медицине*. М.: Мир, 1970, с. 328.

40. Р. Ледли та Л. Ластед, *Медицинская диагностика и современные методы выбора решения.* М.: Мир Математические проблемы в биологии под ред. Р.Беллмана, 1966, с. 141 – 198.

41. І. М. Ляшенко, А. П. Мукоєд*, Моделювання біологічних та екологічних процесів.* Київ, Україна: ВПЦ "Київський університет", 2002, с. 340.

42. В. П. Марценюк, *Системний аналіз та теорія прийняття рішень в біомедицині.* Львів: Ліга-Прес, В зб. Інформаційна підтримка охорони здоров’я, біомедичних досліджень та освіти, с. 59 – 68, 2002.

43. В. П. Марценюк, Н. О. Кравець, А. С. Сверстюк, ”Інформаційна система медико-біологічних досліджень: проект на основі Web-технологій”, *Укр. журнал телемедицини та мед .телематики,* т.1, №1, с. 57 – 60, 2003.

44. В. П. Марценюк, ”Об устойчивости в модели иммунной защиты с учетом нарушения функционирования органа-мишени: метод вырожденных

функционалов Ляпунова", *Кибернетика и системный аналіз,* № 1, с. 153 – 164, 2004.

45. Г. И. Марчук, *Математические модели в иммунологии.* М.: Наука, Главная редакция физико-математической литературы, 1980, с. 264.

46. Н. И. Нисевич, Г. И. Марчук, *Математическое моделирование вирусного гепатита.* Москва: Наука, 1981, с. 352.

47. Л. Гласс та М. Мэки, От часов к хаосу: Ритмы жизни: Пер. с англ., М.: Мир, 1991, с. 248.

48. О. А. Минцер., В. Н. Молотков, Б. Н. Угаров и др., *Биологическая и медицинская кибернетика.* Справочник. К.: Наукова думка, 1989, с. 375.

49. J. M. Murray, *Mathematical Biology.* New York: Springer Verlag, 1989, p. 432.

50. О. Г. Наконечний, О. М. Трофимчук, І. В. Трофімова та ін., *Моделювання та аналіз глобальних біосферних процесів.* Київ, Україна: ВПЦ Київський університет, 2002, с. 93.

51. Ю. М. Онопчук, "Математическое моделирование гипоксии при анемии", *Компьютерная математика,* №1, с. 113 – 121, 2007.

52. Ю. И. Петунин, *Приложение теории случайных процессов в биологии и медицине.* Київ, Україна: Наукова думка, 1981, с. 320.

53. A. Klyushin, S. I. Lyashko, D. A. Nomirovsky, Yu. I. Petunin and V. V. Semenov, *Generelized solutions of operator equations and extreme elements.* New York, Dordrecht, Heidelberg, London: Springer, 2012, p.200.

54. М. Б. Славин, *Методы системного анализа в медицинских исследованиях.* М.: Медицина, 1989, с. 304.

55. J. Keener and J. Sneyd, *Mathematical Physiology.* New York: Springer Verlag, p. 767, 1998.

56. M. Pardalos, V. Yatsenko and S. P. Nair, "Optimization in Control and Learning in Coupled Map Lattice Systems", *Journal of Optimization Theory and Applications,* Springer, № 3, pp. 533 – 547, 2007.

57. Я. П. Драгана, Л. С. Сікора, Б. І. Яворський, *Основи сучасної теорії стахостичних сигналів: енергетична концепція, математичний апарат, фізичне тлумачення.* Львів, Україна: Центр стратегічних досліджень екобіотехнічнах систем, 1999, с. 133.

58. Я. П. Драґан, *Моделі сигналів у лінійних системах.* Київ, Україна: Наукова думка, 1972, с. 302.

59. Я. П. Драґан, *Структура и представление моделей стохастических сигналов.* Київ, Україна: Наукова думка, 1980, с. 384.

60. Я. П. Драган та И. Н. Яворский, *Ритмика морского волнения и подводные акустические сигналы*. Київ, Україна: Наукова думка, 1982, с. 248.

61. В. О. Омельченко, В. М. Безрук, Я. П Драган, О. О. Колесников та А. В. Омельченко, *Імовірнісні моделі випадкових сигналів і полів у прикладах і задачах.* Навч. посібник, Київ, Україна: Ін-тут системних досліджень освіти, 1996, с. 272.

62. Я. Драган, Л. Сікора та И. Яворский, *Основи сучасної теорії стохастичних сигналів: енергетична концепція, математичний аппарат, фізичне тлумачення.* Львів, Україна: Центр стратегічних досліджень еко- біо-технічних систем, 1999, с. 133.

63. С. Лупенко та А. Горкуненко, ”Математичне моделювання циклічних економічних явищ на базі циклічного випадкового процесу для задач автоматизованого аналізу та прогнозу”, *на всеукр. наук. конф.,* Тернопіль, Україна, 2009, с. 12.

64. С. А. Лупенко, ”Детермнированные и случайные циклические функции как модели колебательных явлений и сигналов: определение и классификация”, *Электронное моделирование. Ин-т проблем моделирования в энергетике им. Г.Е. Пухова*, т. 28, № 4,с. 29 – 45, 2006.

65. С. А. Лупенко, ”Статистичні методи обробки циклічного випадкового процесу”, *Електроніка та системи управління,* Київ, Україна: Національний авіаційний університет, № 2 (8), с. 59 – 65, 2006.

66. М. П. Приймака, Б. Г. Марченко та М. В. Приймак, *Побудова моделі та аналіз стохастично періодичних навантажень енергосистем.* Київ, Україна: ІЕД НАН України, Праці Інституту електродинаміки, 1999, Вип.1, с. 129 – 153.

67. М. В. Приймак, ”Марківські періодичні процеси”, *Вісник Тернопільського державного технічного університету,* т.8, №3, с. 17 – 21, 2003.

68. М. В. Приймак, ”Дискретні періодичні білі шуми з неперервними розподілами і можливості їх імітаційного моделювання”, *Відбір і обробка інформації,* Вип.18(94), с.17 – 21, 2003.

69. М. В. Приймак , І. О. Боднарчук, С. А. Лупенко, ”Умовно періодичні випадкові процеси із змінним періодом”, *Вісник ТДТУ*, т.10, №2, 2005.

70. М. В. Приймак, ”Періодичні функції із змінним періодом”, *Матеріали одинадцятої наук конф. ТДТУ ім. Івана Пулюя,* Тернопіль, 2007, с. 70.

71. Л. М. Щербака, Б. Г. Марченко та Л. Н. Щербак, *Линейные случайные процессы и их приложения.* Київ, Україна: Наукова думка, с. 144, 1975.

72. В. П. Бабак, С. М. Маєвський та Л. М. Щербак, *Основи побудови систем аналізу сигналів у неруйнівному контролі.* Київ, Україна: Либідь, 1993, с. 200.

73. Б. Г. Марченкоа та Л. М. Щербак, *Основи теорії вимірювань.* Київ, Україна: ІЕД НАН України, Праці Ін-ту електродинаміки. Електроенергетика, 1999, с. 221 – 230.

74. Б. Г. Марченко та Л. М. Щербак, ”Сучасна концепція побудови теорії вимірювань”, *Доповіді НАН України*, №10, с. 85 – 88, 1999.

75. С. А. Лупенко та Л. М. Щербак, ”Конструктивна математична модель сигналів серця в технічних системах кардіометрії”, *Вимірювальна та обчислювальна техніка в технологічних процесах.* Хмельницьк, Україна: Навчальна книга, 2000, №2, с. 133 – 136.

76. Б. І. Яворський, *Математичні основи радіоелектроніки.* Частина І. Навчальний посібник, Тернопіль, Україна: ТНТУ, 2008, с. 182.

77. V. P. Martsenyuk, I. Ye. Andrushchak, N. M. Gandzyuk, N.Ya. Klymuk, O. M. Kuchvara and Z.V. Mayhruk, ”Decision support system for medical system research” Proceedings – Applied Papers, Edited by: Hajkova, E; Michalek, J; Nakonechnyi, O; Neubauer, J. Internat. Conf. *Problems of decision making under uncertainties,* Brno, Crech Republic, 2012, pp.73 – 78. ***(індексовано - Web of Science, Accession Number: WOS:000394076800012).***

78. В. П. Марценюк та З. В. Майхрук, ”Побудова оптимального керування біфуркацією в моделі Ходжкіна-Хакслі на основі принципу максимуму”, *Математичне та комп’ютерне моделювання.* Сер.: технічні науки, Кам’янець-Подільськ, №.9, с. 78 – 90, 2013.

79. З. В. Майхрук, ”Програмна реалізація чисельного методу оптимального керування біфуркацією в моделі Ходжкіна-Хакслі”, Вісник.ХНУ. Сер.: технічні науки, Хмельницький, №1, с. 186 – 194, 2014. ***(індексується у Index Copernicus, Googel Scholar, Polish Scholarly).***

80. В. П. Марценюк та З. В. Майхрук, ”Дослідження нелінійної динаміки в моделі Ходжкіна-Хакслі на основі експонент Ляпунова”, Наукові праці. Сер.: комп’ютерні технології, Миколаїв, Чорноморськ, вип. 225, т. 237, с. 62 – 65, 2014.

81. В. П. Марценюк та З. В. Майхрук, ”Якісний аналіз системи Ходжкіна-Хакслі електричної активності аксона на основі класифікаційних правил”, Вісник ТНТУ, Тернопіль, № 4, т. 76, с. 223 – 233, 2014.

82. В. П. Марценюк та З. В. Майхрук, ”Мультиваріативний метод якісного аналізу моделі Ходжкіна-Хакслі з індукцією дерева рішень”, *Вісник Київського університету,* сер.: фіз.-мат. наук, №1, с. 145 – 153, 2015.

83 В. П. Марценюк та З. В. Майхрук, ”Алгоритми якісного аналізу моделі Ходжкіна-Хакслі активності аксона”, *Клінічна інформатика і телемедицина,* Харьків, №12, т. 11, с. 43 – 49, 2015. ***(індексується у Index Copernicus).***

84. V. Martsenyuk, K. Warwas, K. Augustynek, A. Klos-Witkowska, V. Karpinskyi, N. Klymuk, Z. Mayhruk, *On Multivariate Method of Qualitative Analysis*

of Hodgkin-Huxley Model with Decision Tree Induction, International Conference on Control, Automation and Systems. HICO, Gyeongju, Korea, 2016, p. 489 – 494. IEEE Catalog Number: CFP1610D-USB, ISBN: 978-89-93215-12-0 ISSN: 2093 - 7121. ***(індексовано - Web of Science, Accession Number: WOS:000401800200075).***

85. І. Є. Андрущак, І. С. Гвоздецька, Н. Я. Климук, О. М. Кучвара, З. В. Майхрук та ін., ”Системно аналітичні методи дослідження медико-біологічних процесів”, доп. наук.-практ. конф. Здобутки клінічної та експериментальної медицини, Тернопіль, 2011, с. 168 – 169.

86. В. П. Марценюк та З. В. Майхрук, ”Якісний аналіз моделі Ходжкіна-Хакслі електричної провідності біологічних клітин”, на XVIII міжнарод. конф. *Problems of Decision Making under Uncertainties*, Ялта, 2011, с. 115 – 117.

87. В. П. Марценюк та З. В. Майхрук,”Экспериментальный метод анализа модели патологического процесса с помощью чисельных характеритик”, на міжн. наук.-практ. конф. *Біофізичні стандарти та інформаційні технології у медицині,* Одеса, 2011, с. 56.

88. В. П. Марценюк та З. В. Майхрук, ”Дослідження біфуракцій в моделі імунного захисту”, на міжн. конф. *Проблеми прийняття рішень в умовах невизначеності,* Мукачів, 2012, с. 165 – 167.

89. В. П. Марценюк та З. В. Майхрук, ”Експериментальний метод аналізу моделі патологічного процесу за допомогою чисельних характеристик”, на *міжн. наук. конф. ім. акад. М. Кравчука,* Київ, 2012, с. 296.

90. V. P. Martsenyuk and Z. V. Mayhruk, ”Decision support system for medical system research”, Internat. *Conf. Problems of decision making under unctrtainties,* Brno, Crech Republic, 2012, pp. 84 – 85.

91. В. П. Марценюк та З. В. Майхрук,”Вивчення біфуркацій на основі дослідження квазіполінома в моделях імунного захисту” міжн. конф. *Прийняття рішень в умовах невизначеності,* Ялта-Форос, 2013, с. 100.

92. В. П. Марценюк та З. В. Майхрук, ”Моделі та методи нелінійного аналізу для процесів популяційної динаміки”, міжн. конф. *Проблеми прийняття рішень в умовах невизначеності,* Східниця, 2013, с. 168.

93. В. П. Марценюк та З. В. Майхрук, ”Дослідження нелінійної динаміки медико-біологічних дисипативних систем”, міжн. конф. *Dynamical Systems Modeling and Stabitify Investigation,* Київ, 2013, с. 205.

94. В. П. Марценюк та З. В. Майхрук, ”Чисельний метод побудови оптимального керування в моделі Ходжкіна-Хакслі електричної активності клітинної мембрани”, наук.-практ. конф. *Науково-технічний прогрес і оптимізація технологічних процесів створення лікарських препаратів,* Тернопіль, 2013, с. 208 – 209.

95. В. П. Марценюк, І. Є. Андрущак, А. С. Сверстюк, О. А. Багрій-Заяць, О. М. Кучвара, Н. М. Гандзюк, З. В. Майхрук та ін., ”Проблеми системного аналізу і прийняття рішень в медико-біологічних дослідженнях”, *конф. до 90-річчя з дня народження академіка В. М. Глушкова,* Київ, 2013, с. 151 – 154.

96. V. P. Martsenyuk, Z. V. Mayhruk, ” Investigation of nonlinear dynamics in model of Hodgkin-Huxley based on lyapunovs exponent”, Internat. Conf. *Problems of decision making under uncertainties,* Brno, Crech Republic, 2014, p.73 – 78.

97. В. П. Марценюк, З. В. Майхрук ”Мультиваріативний метод дослідження збудливості нейрона з індукцією дерева рішень”, Комп'ютерна програма *ПСДМРПУОДР. № 57280,* 20.11.2014.

98. В. П. Марценюк, І. Є. Андрущак, Р. О. Сарабун, Н. М. Гандзюк, З. В. Майхрук ”Система підтримки прийняття рішень в системних медичних дослідженнях”, Комп'ютерна програма *№ 57045,* 17.10.2014.

99. A. Hodgkin and A. Huxley, *A quantitative description of membrane current and its application to conduction and excitation in nerve.* J. Physiol, vol.117, pp. 500-544, 1952.

100. J. Rinzel, *Electrical excitability of cells, theory and experiment: review of the Hodgkin-Huxley foundation and an update,* Bulletin of Mathematical Biology, vol. 52, pp. 5 – 23, 1990.

101. A. L. Hodgkin and A. F. Huxley, *Currents carried by sodium and potassium ions through the membrane of the giant axon of Loligo,* J. Physiol. vol. 116, pp. 449 – 472, 1952a.

102. A. L. Hodgkin and A. F. Huxley, *The components of membrane conductance in the giant axon of Loligo,* J. Physiol. vol. 1I6, pp. 473 – 496, 1952b.

103. A. L. Hodgkin, *The dual effect of membrane potential on sodium conductance in the giant axon of Loligo,* J. Physiol. vol.116, pp. 497 – 506, 1952c.

104. A. L. Hodgkin and A. F. Huxley, *A quantitative description of membrane current and its application to conduction and excitation in nerve,* J. Physiol.(L.), vol.117, pp. 500 – 544, 1952d.

105. A. L. Hodgkin, A. F. Huxley and B. Katz, *Measurements of current-voltage relations in the membrane of the giant axon of Loligo,* J. Physiol. (L.), vol.116, pp. 424 – 448, 1952.

106. A. L. Hodgkin, and B. Katz, *The effect of sodium ions on the electrical activity of the giant axon of the squid,* J. Physiol. (L.), vol.108, pp. 37 – 77, 1949.

107. W. Nernst and A. Lessing, *Ueber die Wanderung galvanischer Polarisation durch Platin - und Palladiumplatten,* Nachr. Kgl. Ges. Wiss. Gott, Sitzung 22, pp. 146 – 160, Februar 1902(2).

108. W. H. Nernst, *Zur Kinetik der Lųsung befindlichen Kųrper: Theorie der Diffusion,* Z. Phys. Chem.3, pp. 613 – 37, 1888.

109. W. H. Nernst, *Die elektromotorische Wirksamkeit der Ionen,* Z. Phys. Chem.4, pp.129 - 81, 1889.

110. R. FitzHugh, *Thresholds and plateaus in the Hodgkin-Huxley nerve equations,* J. Gen. Physiol., 43, p. 867, 196.

111. R. FitzHugh, *Impulses and physiological states in theoretical models of nerve membran.* Biophys. J. 1, pp. 445 – 466, 1961.

112. R. FitzHugh, Chapter,1 Schwan, ed. *Biological Engineering*. McGraw-Hill Book Co., N.Y., pp. 1 – 85, 1961.

113. J. Rinzel, *On repetitive activity in nerve.* Federation Proc. 37, 1978, pp. 2793 – 2802.

114. W. C. Troy, *Bifurcation phenomena in FitzHugh's nerve conduction equations.* J. Math. Anal. & Appl. 54, 1976, pp. 678 – 690.

115. K. S. Cole, H. A. Antosiewicz and P. Rabinowitz, "Automatic computation of nerve excitation", *J. Soc. Indust.* Appl. Math. 3, pp. 153 – 171, 1955.

116. N. H. Saban and R. A. Spangler, "Repetitive response of the Hodgkin-Huxley model for the squid giant axon", *J. Theor. Biol.* 29: pp. 155 – 171, 1970.

117. J. Nagumo, S. Arimoto and S. Yoshizawa, "An active pulse transmission line simulating axon", *Proc. IRE 50.,* pp. 2061 – 2071, 1962.

118. H. P. McKean, "Nagumo's equation", *Advances in Mathematics 4*, pp. 209 – 223, 1970.

119. E. Coraboeuf, "Ionic basis of electrical activity in cardiac tissues", *J. Physiol, Vol*. 234(2), pp. H101-H116, 1978.

120. D. Noble, "The surprising heart: A review of recent progress in cardiac electrophysiology", *J. Physiol. 353*, pp. 129 – 162, 1984.

121. D. Noble, "A modification of the Hodgkin-Huxley equations applicable to Purkinje fibre action and pace-maker potentials", *Journal of Physiology,* 160, pp. 317 – 352, February, 1962.

122. K. Yanagihara and H. Irisawa, *Inward current activated during hyperpolarization in the rabbit sinoatrial node cell.* Pflugers Arch., 385, pp. 11 – 19, 1980.

123. G. Beeler and H. Reuter, "Reconstruction of the action potential of ventricular myocardial fibres", *J. Physiol,* vol. 268, pp. 177 – 210, 1977.

124. R. Mac Allister, D. Noble and H. R.W. Tsien, "Reconstruction of electrical activity of cardiac Purcinje fibres", *J. Physiol,* vol. 215, pp. 1 – 59, 1975.

125. A. Wolf, B. J. Swift, H. L. Swinney and J. A. Vastano, ”Determining lyapunov exponents from a time series”, *Physica D: Nonlinear Phenomena,* vol. 16 Issue 3, pp. 285 – 317, July 1985.

126. G. Benettin,. L. Galgani, A. Giorgilli and J.-M.Strelcyn, *Lyapunov Characteristic Exponents for Smooth Dynamical Systems and for Hamiltonian Systems; a Method for Computing all of them*, Part II: Numerical Application, Meccanica 15 (1980).

127. I. Shimada and T. Nagashima, *Numerical Approach to Ergodic Problem of Dissipative Dynamical Systems,* Prog. Theor. Phys. 61, 1979, p. 1605.

128. R. Eichhorn, J. S. Linz and P. Hanggi, ”Transformation invariance of Lyapunov exponents”, *Chaos, Solitons and Fractals 12*, pp. 1377 – 1383, 2001.

129. D. L. Lukes, ”Differential Equations: Classical to Controlled”, *Academic Press,* New York, 1982.

130. J. Macki and A Strauss, *Introduction to Optimal Control Theory.* Springer-Verlag, New York, 1982.

131. W. H. Fleming and R. W. Rishel, ”Deterministic and Stochastic Optimal Control”, Springer Verlag, New York, 1975.

132. J. T Betts, ”Practical Methods for Optimal Control Using Nonlinear Programming”, SIAM Society for Applied and Industrial Mathematics, Philadelphia, 2001.

133. M. E. Brandt and G. Chen, ”Bifurcation control of two nonlinear models of cardiac activity”, IEEE Trans. Circ. Syst., Issue 44, pp. 1031 – 1034, 1997.

134. D L. Ding and C. Hou, ”Stabilizing control of Hopf bifurcation in the Hodgkin-Huxley model via washout filter with linear control term*”, Nonlinear Dyn 60,* pp. 131 – 139, 2010.

135. B. C. Fabien B, *Some Tools for the Direct Solution of Optimal Control Problems, Advances in Engineering Software.* vol. 29, pp. 45 – 61, 1998.

136. A. D. Alexandrov and A. N. Kolmogorov, *Mathematics: Its Content and Meaning.* Courier Dover Publications, p. 1120, 1999.

137. Lea A. Segel *Mathematical Models in Molecular and Cellular Biology.* CUP Archive, p. 757, 1980.

138. Y. Koch, T. Wolf, P. K. Sorger, R. Eils, B. Brars, *Decision-Tree Based Model Analysis for Efficient Identification of Parameter Relations Leading to Different Signaling States.* PLOS ONE, vol. 8, 2013, iss. 12, e82593.

139. Э. Хайрер, С. Нерсетт та Г. Ваннер, *Решение обыкновенных дифференциальных уравнений. Нежесткие задачи.* М.: Мир, с. 512, 1990.

140. E. M. Izhikevich, ”Neural Excitability, Spiking and Bursting”, *International Journal of Bifurcation and Chaos,* vol. 10, №6, pp. 1171 – 1266, 2000.

141. A. L. Hodgkin, ”The local electric changes associated with repetitive action in a non-modulated axon”, *J. Physiol.,* 107, pp. 165 – 181, 1948.

142. J. R. Clay et al., *A simple modification of the Hodgkin and Huxley equations explains type 3 excitability in squid giant axon.* J. R. Soc. Interface 5, pp. 1421 – 1428, 2008.

143. J. Han and M. Kamber, ”DataMning: Concepts and Techniques”, *Morgan Kaufmann,* San Francisco, 2001.

144. V. Marzeniuk, A. Nakonechny, ”Investigation of Delay System with Piece-Wise Right Side Arising in Radiotherapy”, *WSEAS Transactions on Mathematics,* Issue 1, Volume 3, January, pp. 181 – 187, 2004.

145. В. П. Марценюк *Медична інформатика. Проектування та використання баз даних.* Україна, Тернопіль: Укрмедкнига, с. 178, 2001.

146. Р. Беллман. *Математические методы в медицине.* Москва: Мир, с. 262, 1987.

147. L. Wang, ”Computer-simulated pharmacology experiments for undergraduate pharmacy students: experience from an Australian university”, *Indian Journal Of Pharmacology,* 2001.

148. R. L. Summers, S. M. Hudson and J.P. Montana, *Computer simulation studies and biomedical research*. Animal Welfare Information Center Newsletter 6 (2–4): Winter, 1995/1996.

149. Д. Я. Хусаїнов та І. В. Мусатенко, *Диференціальні рівняння.* Київ. Україна, ВПЦ Київський університет с. 159, 2010.

150. П. В. Соловьев, *FORTRAN для персонального компьютера.* Москва: Справочное пособие, 1991.

151. D. B. Menzel, R. L. Wolpert, J. R. Boger et al., ”Resources available for simulation in toxicology: specialized computers, generalized software and communication networks”, *Drink Water and Health 8*, pp. 229 – 254, 1987.

152. М. Томас, П. Пател та А. Хадсон, *Секреты программирования для Internet на Java,* Перев. с англ. СПб: Питер, с. 640, 1997.

153. Д. Флэнэген, *Java in a Nutshell*, Перев. с англ., Киев: Bhv, 1998, с.720.

154. Eckel B. Thinking in Java, Second Edition, Мейнджер Джейсон. *JAVA: Основы программирования,* Пер. с англ. С. Бойко под ред. Я. Шмидского, К.: BNV, 1999, с. 1997.

155. Джон Родли, *Создание JAVA-апплетов.* К.: Серия: Программирование для Internet, 1996.

156. R. Latkowski, ”On Decomposition for incomplete data Fund”, *Informa*, 54, pp. 1 – 16, 2003.

157. R. Latkowski, M. Micolajczyk, ”Data decomposition and decision rule joining for classification of data with missing values”, In: Proceeding of 4-th *international Conference on Rough Sers and Current Trends in Computiong, Lecture.* Notes in Actificial. Intelligence 3066, Springer-Verlag, Heidelberg, Berlin, pp. 254 – 263, 2004.

158. http://javatechniques.com/blog/faster-deep-copies-of-java-objects/

159. Chaos: An Interdisciplinary Journal of Nonlinear Science 25, 097603 (2015); До: http://dx.doi.org/10.1063/1.4915529

ДОДАТОК А1

«Затверджую»
Проректор з наукової роботи
ДВНЗ «Тернопільський державний
медичний університет
ім. І.Я. Горбачевського МОЗ України»
проф. І.М. Кліш _____
«18» лютого 2015 р.

АКТ
впровадження в науковий процес

1. Найменування пропозиції для впровадження:
Метод керування біфуркацією в електрофізіологічній моделі Ходжкіна-Хакслі на основі принципу максимуму.

2. Розроблювачі та заклад, де розроблено, поштова адреса:
Марценюк Василь Петрович, Майхрук Зоряна Василівна, Тернопільський державний медичний університет імені І.Я. Горбачевського, майдан Волі, 1, м. Тернопіль, 46001.

3. Джерело інформації:
Марценюк В.П. Побудова оптимального керування біфуркацією в моделі Ходжкіна-Хакслі на основі принципу максимуму. /В.П. Марценюк, З.В. Майхрук// Математичне та комп'ютерне моделювання. Серія: Технічні науки: зб. наук. пр. – Кам'янець-Подільський : Кам'янець-Подільський нац. ун-т, 2014. – Вип. № 6. – С. 131-142.

4. Ким впроваджено:
Кафедра фізіології Тернопільського державного медичного університету ім. І.Я. Горбачевського.

5. Термін впровадження: 2014 – 2015 р.

6. Висновок по впровадженню:
Запропоновану математичну модель Ходжкіна-Хакслі можна використати в науковому процесі на кафедрі фізіології.

Відповідальний за впровадження:
Завідувач кафедри фізіології,
д.м.н., професор *С. Н.Вадзюк*

ДОДАТОК А2

«Затверджую»
Перший проректор
Тернопільського державного
медичного університету
імені І.Я.Горбачевського
проф. Ковальчук О.Л. _____
«22» квітня 2015 р.

АКТ
впровадження у навчальний процес

1. ***Найменування пропозиції для впровадження:***
Модель розвитку загального патологічного утворення на основі моделі Ходжкіна-Хакслі.

2. ***Розроблювачі та заклад, де розроблено, поштова адреса:***
Марценюк Василь Петрович, Майхрук Зоряна Василівна, Тернопільський державний медичний університет імені І.Я. Горбачевського, майдан Волі, 1, м. Тернопіль, 46001.

3. ***Джерело інформації***:
Марценюк В.П. Дослідження нелінійної динаміки в моделі Ходжкіна-Хакслі на основі експонент Ляпунова /В.П.Марценюк, З.В.Майхрук// Наукові праці/ Чорноморського державного університету імені Петра Могили комплексу «Києво-Могилянська академія»: наук. журн.- Миколаїв. Серія, Комп'ютерні технології - 2014. – Вип. № 225., Том 237– С. 62-65.

4. ***Де впроваджено:***
Тернопільський державний медичний університет імені І.Я. Горбачевського, Інститут моделювання та аналізу патологічних процесів, кафедра патологічної фізіології.

5. ***Термін впровадження:*** 2014р.

6. ***Висновок по впровадженню***:
Запропоновану математичну модель розвитку загального патологічного утворення на основі моделі Ходжкіна-Хакслі можна використати в навчальному процесі на кафедрі патологічної фізіології при вивченні курсу «Патологічна фізіологія».

Відповідальний за впровадження:
Завідувач кафедри паталогічної фізіології,
д.м.н., професор ***Ю.І.Бондаренко***

ДОДАТОКАЗ

«ЗАТВЕРДЖУЮ»
Проректор з науково-педагогічної роботи Одеського національного медичного університету
проф. Бажора Ю.І.

«__» _____ 2015 р.

АКТ впровадження

1. Найменування пропозиції для впровадження:
Мультиваріативний метод якісного аналізу моделі Ходжкіна-Хакслі з індукцією дерева рішень

2. Розроблювачі та заклад, де розроблено, поштова адреса: Марценюк Василь Петрович, Майхрук Зоряна Василівна, Тернопільський державний медичний університет імені І.Я. Горбачевського, майдан Волі, 1, м.Тернопіль, 46001.

3. Джерело інформації: Марценюк В.П. Якісний аналіз моделі Ходжкіна-Хакслі електричної провідності біологічних клітин./ В.П.Марценюк, З.В. Майхрук //Матеріали XVIII міжнародної конференції [«Проблеми прийняття рішень в умовах невизначенності»], (Ялта, 19-23 вересня 2011)/С.115-117

4. Ким впроваджено: Кафедра біофізики, інформатики та медичної апаратури Одеського національного медичного університету.

5. Термін впровадження: 2014 – 2015 р.

6. Загальна кількість спостережень: 170.

7. Ефективність впровадження у відповідності з критеріями в джерелі інформації. Мультиваріативний метод якісного аналізу дозволяє будувати дерево рішень для класифікації типу збудливості нервової клітини. Дослідження показує, що дана кількість спостережень є достатньою для побудови «стабільної» структури дерева рішень.

Відповідальний за впровадження:
Завідувач кафедри біофізики, інформатики та медичної апаратури
д.м.н., професор
Годлевський Л.С.

ДОДАТОКА4

«Затверджую»
Проректор з наукової
роботи Луцького
національного технічного
університету
к.е.н., доцент
Талах В.І.

АКТ

впровадження в науковий процес результатів дисертаційної роботи «Моделі та методи нелінійного аналізу електрофізіологічної системи Ходжкіна-Хакслі» Майхрук Зоряни Василівни

Дійсний акт складений в тому, що результати теоретичних і практичних досліджень дисертаційної роботи «Моделі та методи нелінійного аналізу електрофізіологічної системи Ходжкіна-Хакслі» Майхрук Зоряни Василівни, які стосувалися розробки алгоритму і його програмної реалізації для мультиваріативного методу якісного аналізу динамічної моделі електричної активності клітин, передані на кафедру "Комп'ютерних технологій" для використання при вирішенні задач класифікації в складних динамічних системах.

Завідувач кафедри комп'ютерних технологій
д. т. н., професор І. Є. Андрущак

Додаток Б**1**

УКРАЇНА

ДЕРЖАВНА СЛУЖБА ІНТЕЛЕКТУАЛЬНОЇ ВЛАСНОСТІ УКРАЇНИ

СВІДОЦТВО

про реєстрацію авторського права на твір

№ 57045

Комп'ютерна програма "Система підтримки прийняття рішень в системних медичних дослідженнях"

(вид, назва службового твору)

Автор(и) **Марценюк Василь Петрович, Андрущак Ігор Євгенович, Гандзюк Надія Михайлівна, Сарабун Роман Олегович, Майхрук Зоряна Василівна**

(повне ім'я, псевдонім (за наявності))

Авторські майнові права належать **Марценюк Василь Петрович, Майдан Волі, 1, м. Тернопіль, 46001; Андрущак Ігор Євгенович, майдан Волі, 1, м. Тернопіль, 46001; Гандзюк Надія Михайлівна, майдан Волі, 1, м. Тернопіль; Сарабун Роман Олегович, майдан Волі, 1, м. Тернопіль, 46001; Майхрук Зоряна Василівна, майдан Волі, 1, м. Тернопіль, 46001; Державний вищий навчальний заклад "Тернопільський державний медичний університет ім. І. Я. Горбачевського Міністерства охорони здоров'я України", майдан Волі, 1, м. Тернопіль, 46001**

(повне ім'я фізичної та/або повне офіційне найменування юридичної особи, адреса)

17.10.2014

Дата реєстрації

Державна служба інтелектуальної власності України
Ідентифікаційний код 37552556

Голова Державної служби інтелектуальної власності України
М.В. Ковіня

Додаток Б**2**

УКРАЇНА

ДЕРЖАВНА СЛУЖБА ВЛАСНОСТІ УКРАЇНИ ІНТЕЛЕКТУАЛЬНОЇ

СВІДОЦТВО

про реєстрацію авторського права на твір

№ 57280

Комп'ютерна програма "Мультиваріативний метод дослідження збудливості нейрона з індукцією дерева рішень"

(вид, назва службового твору)

Автор(и) **Марценюк Василь Петрович, Майхрук Зоряна Василівна**

(повне ім'я, псевдонім (за наявності))

Авторські майнові права належать **Марценюк Василь Петрович, Майдан Волі, 1, м. Тернопіль, 46001; Майхрук Зоряна Василівна, майдан Волі, 1, м. Тернопіль, 46001; Державний вищий навчальний заклад "Тернопільський державний медичний університет ім. І. Я. Горбачевського Міністерства охорони здоров'я України", майдан Волі, 1, м. Тернопіль, 46001**

(повне ім'я фізичної та/або повне офіційне найменування юридичної особи, адреса)

20.11.2014

Дата реєстрації

Державна служба інтелектуальної власності України Ідентифікаційний код 31552556

М.П.

Голова Державної служби інтелектуальної власності України
А.Г.Жарінова

ДОДАТОК В

Таблиця 1.1. Концентрації іонів для більшості серцевих клітин

Іони	**Позаклітинні (mM)**	**Внутрішньоклітинні (mM)**	**Потенціал Нернста (mV)**
Na+	145	15	60
Cl-	100	5	-80
K+	4.5	160	-95
Ca^2	1.8	0.0001	130
H+	0.0001	0.0002	-18

Таблиця 1.2. Константи швидкостей α і β для моделей.

	C1	C2	C3	C4	C5	C0
m	0	-	0/1	-1	-15	-48
m	0	-	-0.12	-1	5	-8
α_h	0.17	-20	0	0	-	-90
β_h	1	∞	0	1	-10	-42
α_n	0	-	0.0001	-1	-10	-50
β_n	0.0002	-80	0	0	-	-90

Таблиця 1.3. Значення α і β для моделі MNT

	C_1	**C_2**	**C_3**	**C_4**	**C_5**	**V_0**
α m	0	—	1	1	-10	-37
β m	40	-17.8	0	0	—	-62
α h	1.209×10^{-3}	-6.534	0	0	—	-20
β h	1	∞	0	1	-10	-30
α p	0	—	-2.25×10^{-4}	-1	13.3	-40
β d	0	—	-4.21×10^{-3}	-1	2.5	5

α f	0	—	3.55×10^{-4}	-1	5.633	-20

Таблиця 1.4.

	C_1	**C_2**	**C_3**	**C_4**	**C_5**	**V_0**
α_m	0	—	1	1	-10	-37
β_m	40	-17.8	0	0	—	-62
α_h	1.209×10^{-3}	-6.534	0	0	—	-20
β_h	1	∞	0	1	-10	-30
α_p	0	—	-2.25×10^{-4}	-1	13.3	-40
β_d	0	—	-4.21×10^{-3}	-1	2.5	5
α_f	0	—	3.55×10^{-4}	-1	5.633	-20

Таблиця 1.5. Значення α і β для моделей Белер-Ройтер

	C_1	**C_2**	**C_3**	**C_4**	**C_5**	**V_0**
α_m	0	-	1	-1	-10	-47
β_m	40	-17.86	0	0	-	-72
α_h	0.126	-4	0	0	-	-77
β_h	1.7	∞	0	1	-12.2	-22.5
α_j	0.055	-4	0	1	-5	-78
β_j	0.3	∞	0	1	-10	-32
α_d	0.095	-100	0	1	-13.9	5
β_d	0.07	-58.5	0	1	20	-44
α_f	0.012	-125	0	1	6.67	-28
β_f	0.0065	-50	0	1	-5	-30
α_x	0.0005	12	0	1	17.5	-50
β_x	0.0013	-16.67	0	1	-25	-20

Таблиця 1.6. Струми для різних іонних моделей

	Струми	Іон	Канали
	I_{Na}	Na^+	m, h
	I_{K_1}	K^+	
	I_{K_2}	K^+	n
	I_{An}	Cl^-	
MNT	Струми	Іон	Канали
	I_{Na}	Na^+	m, h
	I_{si}	Na^{2+}	d, f
	I_{K_1}	K^+	
	I_{K_2}	K^+	s
	I_{x_1}	K^+	X_1
	I_{x_2}	K^+	X_2
	I_{Cl}	Cl^-	q, r
	$I_{Na,b}$	Na^+	
	$I_{Cl,b}$	Cl^-	
YNI	Струми	Іон	Канали
	I_{Na}	Na^+	m, h
	I_s		d, f
	I_K	K^+	P
	I_1	leak	
	I_h		q
BR	Струми	Іон	Канали
	I_{Na}	Na^+	m, h, j
	I_s	Ca^{2+}	d, f
	I_K	K^+	
	I_X	K^+	x

ДОДАТОК Г

Лістинги Java-класів програмного середовища дослідження розвитку загального патологічного утворення на основі моделі Ходжкіна-Хакслі

```
#
# Hodgkin_Huxley optimal control
state v m n h
control u
real    gk gna gl gk gna ia1 xm xn xh vk vna vl I_app
nodes = 1000
method = dyn_sqp
ode = huen
output_file = D:\My_code\Klymuk\HealthInsurance\Hodgkin_Huxley_opt.stcl
epsilon = 1.0e-9

initial_condition:

        v=0.00002
        m=0.05293
        n=0.317680
        h=0.59612

dynamic_equation:
     gk=36.
     gna=120.
     gl=0.3
          vna=-115
          vk=12
          vl=-10.599
```

```
        I_app=7

        ddt v = - gk*n*n*n*n*(v-vk) - gna*m*m*m*h*(v-vna)-gl*(v-
vl)+I_app
        ddt m = (1-m)*((25-v)/10)/(exp((25-v)/10)-1)-m*4*exp(-v/18)
        ddt n = (1-n)*0.1*((10-v)/10)/(exp((10-v)/10)-1)-n*0.125*exp(-v/80)
        ddt h = 0.07*exp(-v/20)*(1-h)-(h/(1+exp((30-v)/10)))

    inequality_constraint:

        d = -u   # -u <= 0
        d = u-1  # u <= 1

    cost_functional:

        initial_time = 0.0
        final_time = 20
        L =u*u
```

```
package medbioinvestigations.hodgkin_huxley;
import medbioinvestigations.graph.*;
//import AdvancedFrame;
import java.awt.GridLayout;
import java.awt.Button;
import java.awt.Event;

/*
 *
 * Hodgkin_HuxleyInputDataFrame
```

```
 *
 */
public class Hodgkin_HuxleyInputDataFrame extends AdvancedFrame
{
                    LabelAndTextField m_gk = new
LabelAndTextField("gk", "36.");
                    LabelAndTextField m_gna = new
LabelAndTextField("gna", "120.");
                    LabelAndTextField m_gl = new
LabelAndTextField("gl", "0.3");
                    LabelAndTextField m_cm = new
LabelAndTextField("cm", "20.");
                    LabelAndTextField m_vk = new
LabelAndTextField("vk", "-12.");
                    LabelAndTextField m_vna = new
LabelAndTextField("vna", "115.");
                    LabelAndTextField m_vl = new
LabelAndTextField("vl", "10.6");
                    LabelAndTextField m_ia1 = new
LabelAndTextField("ia1", "2000.");
                    LabelAndTextField m_t1 = new
LabelAndTextField("t1", "5.");
                    LabelAndTextField m_dt1 = new
LabelAndTextField("dt1", "0.2");
                    LabelAndTextField m_ia2 = new
LabelAndTextField("ia2", "0.");
                    LabelAndTextField m_t2 = new
LabelAndTextField("t2", "7.");
```

```
                    LabelAndTextField m_dt2 = new
LabelAndTextField("dt2", "0.2");
                    LabelAndTextField m_xm = new
LabelAndTextField("xm", "1.");
                    LabelAndTextField m_xn = new
LabelAndTextField("xn", "1.");
                    LabelAndTextField m_xh = new
LabelAndTextField("xh", "1.");
                    LabelAndTextField m_v0 = new
LabelAndTextField("v0", "0.");
                    LabelAndTextField m_n0 = new
LabelAndTextField("n0", "0.317680");
                    LabelAndTextField m_m0 = new
LabelAndTextField("m0", "0.052934");
                    LabelAndTextField m_h0 = new
LabelAndTextField("h0", "0.596110");

                    LabelAndTextField m_dTau = new
LabelAndTextField("dTau_8", "5.");
                    LabelAndTextField m_x0 = new
LabelAndTextField("t0", "0.");
                    LabelAndTextField m_x1 = new
LabelAndTextField("T", "40.");

        private Button m_ShowGraphs = new Button("Show graphs...");

        public Hodgkin_HuxleyInputDataFrame()
        {
                    show();
```

```
                    hide();
                    setTitle("Input data for Hodgkin-Huxley model ... - "
+ getTitle());
                    resize(insets().left + insets().right  + 600,
                                        insets().top    +  insets().bottom  +
400);
                    setLayout(new GridLayout(6,4));
                    add(m_gk);
                    add(m_gna);
                    add(m_gl);
                    add(m_cm);
                    add(m_vk);
                    add(m_vna);
                    add(m_vl);
                    add(m_ia1);
                    add(m_t1);
                    add(m_dt1);
                    add(m_ia2);
                    add(m_t2);
                    add(m_dt2);
                    add(m_xm);
                    add(m_xn);
                    add(m_xh);
                    add(m_v0);
                    add(m_n0);
                    add(m_m0);
                    add(m_h0);
                    add(m_dTau);
                    add(m_x0);
```

```
                    add(m_x1);
                    add(m_ShowGraphs);

                    show();
            }
            public boolean action(Event event, Object obj)
            {
                Object oTarget = event.target;
                if (oTarget instanceof Button)
                {
                    Button buttonTarget = (Button)oTarget;
                    String sButtonString = buttonTarget.getLabel();
                    if (sButtonString.compareTo("Show graphs...") == 0)
                    {
                        //////////GRAPHS///////////////
                        AdvancedFrame graph_frame = new
AdvancedFrame();
                        graph_frame.show();
                        graph_frame.hide();
                        graph_frame.resize(graph_frame.insets().left +
graph_frame.insets().right  + 900,
                                           graph_frame.insets().top +
graph_frame.insets().bottom + 1500);
                        //////
                        AdvancedMenu menu = new
AdvancedMenu(graph_frame);
                        menu.CreateMenu();
                        //////
                        try{
```

```
                    graph_frame.m_applet = new
Hodgkin_HuxleyGraph(
                        m_gk.v, m_gna.v, m_gl.v, m_cm.v, m_vk.v,
m_vna.v,
                        m_vl.v, m_ia1.v, m_t1.v, m_dt1.v, m_ia2.v,
m_t2.v, m_dt2.v,
                        m_xm.v, m_xn.v, m_xh.v, m_v0.v, m_n0.v,
m_m0.v, m_h0.v,
                        m_dTau.v, m_x0.v, m_x1.v);
                    graph_frame.add("Center", graph_frame.m_applet);
                    graph_frame.m_applet.init();
                    graph_frame.m_applet.start();
                    graph_frame.show();
                    }
                    catch(Exception e)
                    {
                        //graph_applet.stop();
                    }
                    return true;
                    ////////////////////////////////////
                }
            }
            return true;
        }
    }
```

package medbioinvestigations.hodgkin_huxley;

```
import medbioinvestigations.graph.GraphConstruction;
import java.awt.*;
```

```
import medbioinvestigations.graph.LabelAndTextField;
import medbioinvestigations.graph.*;
import decision_tree.fde.hh.MultiVariateMethod;
import javax.swing.JComboBox;
import java.awt.event.*;

/*
 *
 * Hodgkin_HuxleyGraph
 *
 */
public class Hodgkin_HuxleyGraph extends GraphConstruction
{
        Hodgkin_Huxley SampleSystemSolution;
       public String m_sClassName = "";
        private double m_dHmax;
        // Parameters of the system
        private double m_gk;
        private double m_gna;
        private double m_gl;
        private double m_cm;
        private double m_vk;
        private double m_vna;
        private double m_vl;
        private double m_ia1;
        private double m_t1;
        private double m_dt1;
        private double m_ia2;
        private double m_t2;
```

```
    private double m_dt2;
    private double m_xm;
    private double m_xn;
    private double m_xh;
    private double m_v0;
    private double m_n0;
    private double m_m0;
    private double m_h0;
    private double m_dTau; ////////////////////////////////

    public Hodgkin_HuxleyGraph(double gk, double gna, double gl,
            double cm, double vk, double vna, double vl, double ia1,
            double t1, double dt1, double ia2, double t2, double dt2,
            double xm, double xn, double xh, double v0, double n0,
            double m0, double h0,
            double arg_m_dTau, double x0, double x1, final
MultiVariateMethod mvm)
    {
        super();
        //m_x0=0;
        //m_x1 =2000.0;
        m_nScale =30;
        m_bLegend =true;
        m_bCaptions =true;
        m_nGraphWidth =190;
        m_nGraphHeight =200;
        // IMPORTANT!!! number of  graphs
        m_nGraphCount =7;
        // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
m_bPunctureLine =true;
//m_dDelay =15.;
//m_dHmax = 5.;  // preferable
m_dDelay = 0.;
m_dHmax = arg_m_dTau;
m_gk = gk;
m_gna = gna;
m_gl = gl;
m_cm = cm;
m_vk = vk;
m_vna = vna;
m_vl = vl;
m_ia1 = ia1;
m_t1 = t1;
m_dt1 = dt1;
m_ia2 = ia2;
m_t2 = t2;
m_dt2 = dt2;
m_xm = xm;
m_xn = xn;
m_xh = xh;
m_v0 = v0;
m_n0 = n0;
m_m0 = m0;
m_h0 = h0;
m_dTau = arg_m_dTau;
m_x0 = x0;
m_x1 = x1;
```

```
                ////////////// class name ////////////////////
                String[] classStrings = {"type I","type II","type III","type IV", "type
V"};
                JComboBox m_jcbClassName = new JComboBox(classStrings);

                m_jcbClassName.addActionListener(new
java.awt.event.ActionListener() {
                    public void actionPerformed(ActionEvent e)  {
                        JComboBox jcbClass = (JComboBox) e.getSource();
                        m_sClassName = (String)jcbClass.getSelectedItem();
                        mvm.m_sClassName = m_sClassName;
                        ((AdvancedFrame)getParent()).dispose();
                        mvm.run4thStep();
                    }
                });

                ////////////////////////////////////////////////
                m_p.add(m_jcbClassName);
            }

        // constructor for non-multivariate method
            public Hodgkin_HuxleyGraph(double gk, double gna, double gl,
                    double cm, double vk, double vna, double vl, double ia1,
                    double t1, double dt1, double ia2, double t2, double dt2,
                    double xm, double xn, double xh, double v0, double n0,
                    double m0, double h0,
                    double arg_m_dTau, double x0, double x1)
            {
                    super();
```

```
//m_x0=0;
//m_x1 =2000.0;
m_nScale =30;
m_bLegend =true;
m_bCaptions =true;
m_nGraphWidth =190;
m_nGraphHeight =200;
// IMPORTANT!!! number of  graphs
m_nGraphCount =7;
// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
m_bPunctureLine =true;
//m_dDelay =15.;
//m_dHmax = 5.;  // preferable
m_dDelay = 0.;
m_dHmax = arg_m_dTau;
m_gk = gk;
m_gna = gna;
m_gl = gl;
m_cm = cm;
m_vk = vk;
m_vna = vna;
m_vl = vl;
m_ia1 = ia1;
m_t1 = t1;
m_dt1 = dt1;
m_ia2 = ia2;
m_t2 = t2;
m_dt2 = dt2;
m_xm = xm;
```

```
            m_xn = xn;
            m_xh = xh;
            m_v0 = v0;
            m_n0 = n0;
            m_m0 = m0;
            m_h0 = h0;
            m_dTau = arg_m_dTau;
            m_x0 = x0;
            m_x1 = x1;
    }

public double f(int i, double x)
    {
            switch(i)
            {
            case 1: return SampleSystemSolution.ylag(i, x);
            case 2: return SampleSystemSolution.ylag(i, x);
            case 3: return SampleSystemSolution.ylag(i, x);
            case 4: return SampleSystemSolution.ylag(i, x);
            case 5: return SampleSystemSolution.ia(x);
            case 6: return SampleSystemSolution.ina(x);
            case 7: return SampleSystemSolution.ik(x);
            default: return 0.;
            }

}

    public String GraphCaptions(int i)
    {
```

```
                switch(i)
                {
                case 1: return  "action potential";
                case 2: return  "m";
                case 3: return  "n";
                case 4: return  "h";
                case 5: return  "applied current";
                case 6: return  "Na current";
                case 7: return  "K current";
                default: return  "No name";
                }

        }

        synchronized public void init()
        {
                String param;
                // □□□□□□□□□ □□□□□□ □□□□□□
                Font f = getFont();
                m_fm = getFontMetrics(f);

                //      □□□□□      □□□□□□□□      □□□□□
□□□□□□□□□□□□□□□□□
                m_bBeginDrag = true;

                m_bDrag = false;

                m_bWholeScreenMode = false;
```

```
                // □□□□□□□□□ □□□□□□□ □□ □□□□ □□□ □□□□ x □□ y
                m_nOffset_x = m_nGraphWidth / 8;
                if (m_bLegend)
                {
                        m_nOffset_y = m_nGraphHeight / 5;
                }
                else
                {
                        m_nOffset_y = m_nGraphHeight / 8;
                }

                //  □□□□□□□□□  □□"□□□□□□  □□□□□□ □□□ □□□□□□□□□□ □□□□□□□
                if (m_x0 > m_x1)
                {
                        m_x0 = m_x0 + m_x1;
                        m_x1 = m_x0 - m_x1;
                        m_x0 = m_x0 - m_x1;
                }

                ////////////IMPOTRANT !!!!!!!!//////////////////////////////////////
                // here dimension of system is 4 !!!
                //SampleSystemSolution = new Hodgkin_Huxley(m_nGraphCount,
                SampleSystemSolution = new Hodgkin_Huxley(4,
                        m_x0, m_x1, m_dDelay, m_dHmax, m_gk, m_gna, m_gl, m_cm, m_vk, m_vna,
                        m_vl, m_ia1, m_t1, m_dt1, m_ia2, m_t2, m_dt2,
```

```
            m_xm, m_xn, m_xh, m_v0, m_n0, m_m0, m_h0,
            m_dTau);

        ////////////////////////////////////////////////////////////////////

        m_x0 = m_x0 - m_dDelay;

        BoundsLocation bl = new BoundsLocation();
        bl.m_dLeftBound = m_x0;
        bl.m_dRightBound = m_x1;
        m_llCursorLocations.Add(bl);

        ColorList.Add(Color.yellow);
        ColorList.Add(Color.darkGray);
        ColorList.Add(Color.gray);
        ColorList.Add(Color.lightGray);
        ColorList.Add(Color.magenta);
        ColorList.Add(Color.pink);
        ColorList.Add(Color.cyan);
        ColorList.Add(Color.black);
        ColorList.Add(Color.orange);
        ColorList.Add(Color.green);
        ColorList.Add(Color.blue);
        ColorList.Add(Color.red);

    }
}
```

package medbioinvestigations.hodgkin_huxley;

```
import medbioinvestigations.fde.DelaySystemSolution;

/*
 *
 * Hodgkin_Huxley model
 *
 */
public class Hodgkin_Huxley extends DelaySystemSolution
{
        // Parameters of the system
        double m_gk;
        double m_gna;
        double m_gl;
        double m_cm;
        double m_vk;
        double m_vna;
        double m_vl;
        double m_ia1;
        double m_t1;
        double m_dt1;
        double m_ia2;
        double m_t2;
        double m_dt2;
        double m_xm;
        double m_xn;
        double m_xh;
        double m_v0;
        double m_n0;
        double m_m0;
```

```
    double m_h0;
    double m_dTau;
   double x1;

    public Hodgkin_Huxley(int arg_n, double arg_x0, double arg_x1, double
dDelay, double hmax,
        double gk, double gna, double gl, double cm, double vk, double
vna,
        double vl, double ia1, double t1, double dt1,
        double ia2, double t2, double dt2, double xm,
        double xn, double xh, double v0, double n0,
        double m0, double h0, double dTau)
    {
        // GENERAL INITALIZATION ////////////////////

        n = arg_n;
        x0 = arg_x0;
        //double x1 = arg_x1;
      x1 = arg_x1;

        xstor = new double[mxst + 1];
        ystor = new double[Math.max(n,nn) + 1][mxst + 1];
        c1 = new double[Math.max(n,nn) + 1][mxst + 1];
        c2 = new double[Math.max(n,nn) + 1][mxst + 1];
        c3 = new double[Math.max(n,nn) + 1][mxst + 1];
        c4 = new double[Math.max(n,nn) + 1][mxst + 1];

        ////////////////////////////////
        m_gk = gk;
```

```
m_gna = gna;
m_gl = gl;
m_cm = cm;
m_vk = vk;
m_vna = vna;
m_vl = vl;
m_ia1 = ia1;
m_t1 = t1;
m_dt1 = dt1;
m_ia2 = ia2;
m_t2 = t2;
m_dt2 = dt2;
m_xm = xm;
m_xn = xn;
m_xh = xh;
m_v0 = v0;
m_n0 = n0;
m_m0 = m0;
m_h0 = h0;
m_dTau = dTau;

////////////////////////////////
last = 0;
double x;
double  h = 0.5;  //initial integration step size
y = new double[Math.max(n, nn) + 1];
for(int i = 1; i <= Math.max(n, nn); i++)
{
        if(i <= n)
```

```
                {
                        y[i] = phi(i, x0);
                }
                else
                {
                        y[i] = 0;
                }
        }
        /*for (double i = (int)x0; i <= (int)x1; i += dDelay)
        {
                x = i - dDelay;
                double xend = (double)(i);
                double eps = 1.e-6;
                retard(n, x, y, xend, eps, hmax, h);

        } */

        for (double i = x0; i <= x1; i += hmax)
        {
                x = i - hmax;
                double xend = i;
                double eps = 1.e-6;
                if
(((x<=m_t1)&(xend>m_t1+dt1))|((x<=m_t2)&(xend>m_t2+dt2)))
                {
                        for (double j = x; j <= xend; j += hmax/40.)
                        {
                                double t = j - hmax/40.;
                                double tend = j;
```

```
                        retard(n,t,y,tend, eps, hmax/60., hmax/60.);
                    }
                }
                else retard(n, x, y, xend, eps, hmax, h);
            }

        }

        ////////////////////////////////////////////////////////////////
        // This method return value of system right-side f(x,y(x),y(x-tau))
        // for given (x,y)    ////////////////////////////////////////////
        ////////////////////////////////////////////////////////////////
        synchronized public double[] fcn(double x, double y[])
        {
            double[] dRight_side = new double[Math.max(n, nn) + 1];

            // Hodgkin-Huxley modelling
            double v_ik = m_gk*Math.pow(y[3],4)*(y[1]-m_vk);
            double v_ina = m_gna*Math.pow(y[2],3)*y[4]*(y[1]-m_vna);
            double v_il = m_gl*(y[1]-m_vl);
            dRight_side[1] = (-v_ik-v_ina-v_il+ia(x))/m_cm;
            dRight_side[2] = (am(y[1])*(1.-y[2])-bm(y[1])*y[2])*m_xm;
            dRight_side[3] = (an(y[1])*(1.-y[3])-bn(y[1])*y[3])*m_xn;
            dRight_side[4] = (ah(y[1])*(1.-y[4])-bh(y[1])*y[4])*m_xh;
            ////////////////////////////////

            return dRight_side;
        }
```

```
public double phi(int i, double x)
{
	switch (i)
	{
		case 1:
			return m_v0;
		case 2:
			return m_m0;
		case 3:
			return m_n0;
		case 4:
			return m_h0;
		default:
			return 1.;
	}
}


double am(double v)
{
	return 0.1*(25.-v)/(Math.exp((25.-v)/10.)-1.);
}

double bm(double v)
{
	return 4.*Math.exp(-v/18.);
}
```

```
double ah(double v)
{
        return 0.07*Math.exp(-v/20.);
}

double bh(double v)
{
        return 1./(Math.exp((30.-v)/10.)+1);
}

double an(double v)
{
        return 0.01*(10.-v)/(Math.exp((10.-v)/10.)-1.);
}

double bn(double v)
{
        return 0.125*Math.exp(-v/80.);
}

public double ik(double t)
{
        return m_gk*Math.pow(ylag(3,t),4)*(ylag(1,t)-m_vk);
}

public double ina(double t)
{
        return m_gna*Math.pow(ylag(2,t),3)*ylag(4,t)*(ylag(1,t)-m_vna);
}
```

```
  public double il(double t)
  {
        return m_gl*(ylag(1,t)-m_vl);
  }

  public double ia_repetitive_firing(double t)
  {
   double result = i(t, m_ia1,m_t1,m_dt1);
   for(double j=x0; j<x1; j=j+(m_t2-m_t1))
   {
      result = result + i(t, m_ia2,j,m_dt2);
   }
   return result;
  }

 public double ia_simplified(double t)
  {
        return i(t, m_ia1,m_t1,m_dt1)+i(t, m_ia2,m_t2,m_dt2);
  }

 public double ia_incrising(double t)
  {
        return  ((m_ia2-m_ia1)/(x1-x0))*t  +  m_ia1-((m_ia2-m_ia1)/(x1-
x0))*x0;
  }

 public double ia_constant(double t)
  {
```

```
            if(t<x0) return 0.;
            else return m_ia2;
          }

        public double ia(double t)
          {
            if(t<x0) return 0.;
            else return ((m_ia2-m_ia1)/(x1-x0))*t + m_ia1-((m_ia2-m_ia1)/(x1-
x0))*x0;
          }

          double i(double t, double amp, double t0, double dt)
          {
                return amp*(heav(t-t0)-heav(t-t0-dt));
          }

          double heav(double t) // Heavisade's function
          {
                if (t >= 0)
                      return 1.;
                else return 0.;
          }
      }
```

package medbioinvestigations.graph;

```
      /*
       *
       * GraphConstruction
       *
```

```
 */

import java.applet.*;
import java.awt.*;

//////////////////////////////////////////////////////

//////////////////////////////////////////////////////

abstract public class GraphConstruction extends java.applet.Applet implements
Runnable
{

        public abstract String GraphCaptions(int i);

        Thread m_Graphics = null;
        double m_fps = .1;
        private int m_nAreaNumber;
        public double m_dDelay;
        public boolean m_bLegend;
        public boolean m_bCaptions;
        public boolean m_bPunctureLine;
        public double m_x0;     //  □□□   □□□□   □□□□□□
□□□□□□□□□□□
        public double m_x1;     //  □□□□□   □□□□   □□□□□□
□□□□□□□□□□□
```

```
        public LinkedList m_llCursorLocations = new LinkedList(); //
□□"□□□□□□      □□□□□□      □□□      □□□□□□□□
□□□□□□□□□□
        public FontMetrics m_fm;          //□□□□□□□ □□□□□□
        private Dimension m_dimCursorLocBegin;  // □□□□□□□□□□
□□□□□□□ □ □□□□□□□ □□□□□□□□□□□□□□□
        private Dimension m_dimCursorLocEnd;  // □□□□□□□□□□
□□□□□□□ □ □□□□ □□□□□□□□□□□□□□□
        public boolean m_bDrag; //    □□□□ TRUE - □□ □ □□□□□
□□□□□□ - □□□□□□□□□□□□□□□
        public boolean m_bBeginDrag;  // □□□□ TRUE - □□
□□□□□□□ □□□□□□□□□□□□□□□
        public int m_nOffset_x;
        public int m_nOffset_y;
        public double m_dStep; // m_dStep - □□□□ □□□□ □□ x
        public int m_nScale; // □□□□□□□ □□□□□
        public Button m_button = new Button("Back");
        public Button m_buttonFewGraphs = new Button("Mark few graphs");
        public Button m_buttonWholeScreen = new Button("Whole screen");
       //!!!!!!!!!!!!!!!!!!!!!!!!!!!public - not private Control Panel
        public Panel m_p = new Panel();
       //!!!!!!!!!!!!!!!!!!!!!!!!!!
        private Dimension m_dimDragEvtLoc;
        public int m_nGraphWidth;
        public int m_nGraphHeight;
        public int m_nGraphCount;
        public boolean m_bMarkGraphs;
        public boolean m_bWholeScreenMode;
```

```
        public GraphicalSearchValue m_graphicalSearchValue = new
GraphicalSearchValue(false, 1, 0.);
        private boolean m_bWholeScreenChosen = false;
        private FunctionList m_nFunctionList = null;


        // □□□□□□ □□□□□ □□□□□□□□□□
        private Image m_image; // □□□□□□□□□□□
□□□□□□□□□□
        private Graphics m_g; // □□□□□□□□□□□□ □ □□□
□□□□□□□□□ □□"□□□
        Dimension m_dimImage; // □□□□□
□□□□□□□□□□□□□ □□□□□□□□□□

      abstract public double f(int i, double x);

       abstract public void init();

       public GraphConstruction()
       {
            ////////////////////////////
            setLayout(new BorderLayout());

            // PAY ATTENTION !!! HERE CONTROL PANEL IS VISIBLE
            add("East",m_p);
            // END !!!!!!!!!!!!!
          // Components to Control Panel can be added
          /*
```

```
                m_p.add(m_button);
                m_p.add(m_buttonFewGraphs);
                m_p.add(m_buttonWholeScreen);
               */
                ///////////////////////////////
            }

        public void paint(Graphics g)
            {

                //     □□□□□□□□□□□□□          □□□□□□□
□□□□□□□□□□□
                if(m_image != null)
                {
                        g.drawImage(m_image, 0, 0, null);
                }
            }


            public void update(Graphics g)
            {
                ///////////////////////////////
                //showStatus("mail to marceniuk@yahoo.com"); this is for applet
embedded in web-page not frame
                ///////////////////////////////

                // □□□□□□□□□□□□□□, □□ □□□□□□□□□□□
□□ □□□ □□ □□□□□□,
                // □□ □ □□□□ □□□□□□
```

```
                ResizeImage();

                //      □□□□□□□□            □□□□□□□□□□□□
□□□□□□□□□□□ (□ □□ □□, □□ □□ □□□□□)
                Color colFG = getForeground();
                Color colBG = getBackground();
                m_g.setColor(colBG);
                m_g.fillRect(0, 0, m_dimImage.width, m_dimImage.height);
                m_g.setColor(colFG);


                if (m_bWholeScreenChosen)
                {
                        //showStatus("Atention! To see previous screen click graph
area...");
                        if (m_nAreaNumber != (m_nGraphCount + 1))
                        {
                                PaintDataFrame(m_g,     m_nAreaNumber,     0,     0,
size().width, size().height, m_x0, m_x1, Color.red);
                        }
                        else
                        {
                                if (!m_graphicalSearchValue.switched_on)
                                PaintFewDataFrame(m_g,     0,     0,     size().width,
size().height, m_x0, m_x1);
                                else                PaintGraphicalSearchValue(m_g,
m_graphicalSearchValue.graphsNumber,
                                        m_graphicalSearchValue.ordinate,     0,     0,
size().width, size().height, m_x0, m_x1);
```

```
                }
            }
            else
            {
                // □□□□□□□□□□ □□ □□□□□□□□□□ □
□□□□□□□ □□□
                PaintFewGraphs(m_g, m_nGraphCount, m_nGraphWidth,
m_nGraphHeight, m_x0, m_x1);
            }

            if (m_bDrag & !(m_bMarkGraphs) & !(m_bWholeScreenMode))
            {
                m_g.setColor(Color.blue);
                //draw Rectangle
                int xPoints[] = {m_dimCursorLocBegin.width,
m_dimDragEvtLoc.width, m_dimDragEvtLoc.width, m_dimCursorLocBegin.width,
m_dimCursorLocBegin.width};
                int yPoints[] = {m_dimCursorLocBegin.height,
m_dimCursorLocBegin.height, m_dimDragEvtLoc.height,
m_dimDragEvtLoc.height, m_dimCursorLocBegin.height};
                m_g.drawPolygon(xPoints, yPoints, 5);
                m_g.drawString(Double.toString(m_x0 +
(m_dimCursorLocBegin.width - (m_dimCursorLocBegin.width / m_nGraphWidth) *
m_nGraphWidth - m_nOffset_x) * m_dStep), m_dimCursorLocBegin.width + 1,
m_dimCursorLocBegin.height + 1);
                m_g.drawString(Double.toString(m_x0 +
(m_dimDragEvtLoc.width - (m_dimCursorLocBegin.width / m_nGraphWidth) *
m_nGraphWidth - m_nOffset_x) * m_dStep), m_dimDragEvtLoc.width + 1,
m_dimDragEvtLoc.height + 1);
```

```
                }

                //          □□□□□      □□□□□□□□□□□□□
□□□□□□□□□□□
                paint(g);

        }

        private void PaintFewGraphs(Graphics g, int n, int nGraphWidth, int
nGraphHeight, double x0, double x1)
        {
                int nCountWidth = size().width / nGraphWidth;
                int nCountHeight = size().height / nGraphHeight;
                int nPossibleToDisplay = nCountWidth * nCountHeight;
                int nFunctionCounter = 0;
                if (n <= nPossibleToDisplay)
                {
    UsualGraphs:
                        for(int j = 0; j <= nCountHeight - 1; j++)
                        {
                                for(int i = 0; i <= nCountWidth - 1; i++)
                                {
                                        if (nFunctionCounter++ <= n - 1)
                                        {
                                                PaintDataFrame(g,      nFunctionCounter,
nGraphWidth * i, nGraphHeight * j, nGraphWidth, nGraphHeight, x0, x1, Color.red);

                                        }
                                        else
```

```
                                {
                                        break UsualGraphs;
                                }
                        }
                }
        }
        else
        {
                for(int j = 0; j <= nCountHeight - 1; j++)
                {
                        for(int i = 0; i <= nCountWidth - 1; i++)
                        {
                                PaintDataFrame(g, nFunctionCounter++,
nGraphWidth * i, nGraphHeight * j, nGraphWidth, nGraphHeight, x0, x1, Color.red);

                        }
                }
                g.drawString("Some graphs couldn't be displayed",
nCountWidth * nGraphWidth, nCountHeight * nGraphHeight);
        }
        // Paint few graphs together
        if (FunctionList.m_head != null)
        {
                PaintFewDataFrame(g, ((n) % (size().width /
m_nGraphWidth)) * m_nGraphWidth , ((n) / (size().width / m_nGraphWidth)) *
m_nGraphHeight, m_nGraphWidth, m_nGraphHeight, x0, x1);
        }
        // Paint graph in order
        //to find value of abscissa at given ordinate
```

```
                if (m_graphicalSearchValue.switched_on)
                {
                        PaintGraphicalSearchValue(g,
m_graphicalSearchValue.graphsNumber,
                                m_graphicalSearchValue.ordinate, ((n) %
(size().width / m_nGraphWidth)) * m_nGraphWidth , ((n) / (size().width /
m_nGraphWidth)) * m_nGraphHeight, m_nGraphWidth, m_nGraphHeight, x0, x1);
                }
        }

        private void PaintGraphicalSearchValue(Graphics g, int graphNumber,
                double ordinate, int nLeftBound,
                int nUpperBound, int nWidth, int nHeight, double x0, double x1)
        {
                // 12.12.2003
                // nLeftBound - x-coordinate of left-upper corner
                // nUpperBound - y-coordinate of left-upper corner
                // nWidth - width of rectangle for graph and legend
                // nHeight - height of rectangle for graph and legend

                // m_dStep - □□□□ □□□□ □□ x
                m_dStep = (x1 - x0) / (nWidth - m_nOffset_x);

                //□□□□□ □□□ □□□□□□ - □□□□
                g.setColor(Color.pink);
                g.drawRect(nLeftBound, nUpperBound, nWidth, nHeight);
                g.setColor(Color.white);
                g.fillRect(nLeftBound + 1, nUpperBound + 1, nWidth - 2, nHeight
- 2);
```

```
//□□□□□□□□□□ □□□□□□ □□□□□□□ y
double ymin = ordinate;
double ymax = ordinate;
double y;

        for (double x = x0; x < x1 ; x = x + m_dStep)
        {
                y = f(graphNumber, x);
                if (y < ymin)
                {
                        ymin = y;
                }
                if (y > ymax)
                {
                        ymax = y;
                }
        }

// ajustment of values for y axis
double c = ymax - ymin;
if (c < 1.e-6) c = 1.;
ymin = ymin - 0.25*c;
ymax = ymax        + 0.25*c;
////////////////////////////////
// h - □□□□ □□□□ □□ y
double h = (nHeight - m_nOffset_y) / (ymax - ymin);
```

```
                ////////////////////////////////////////////////
                // □□□□□□□□□ □□□□ □□□□□□□□□□ □□□□ ////////////////
                ////////////////////////////////////////////////

                // □□□□□□□□□ □□□□
                g.setColor(Color.black);
                g.drawLine(nLeftBound + m_nOffset_x, nUpperBound + nHeight
- m_nOffset_y, nLeftBound + m_nOffset_x, nUpperBound);
                g.drawLine(nLeftBound + m_nOffset_x, nUpperBound + nHeight
- m_nOffset_y, nLeftBound + nWidth, nUpperBound + nHeight - m_nOffset_y);
                int nSymb_Height = m_fm.getHeight(); // □□□□□□
□□□□□□□
                // □□□□□□□□□ □□□□□        y
                for (int nMark = 0; nMark < nHeight - m_nOffset_y; nMark +=
m_nScale)
                {
                        int nL = 3;
                        if (nMark % (2 * m_nScale) == 0)
                        {
                                nL = 6;
                        }
                        g.drawLine(nLeftBound + m_nOffset_x, nUpperBound +
nHeight - m_nOffset_y - nMark, nLeftBound + m_nOffset_x - nL, nUpperBound +
nHeight - m_nOffset_y - nMark);
                        String sAdd_String = Double.toString(ymin +
(double)(nMark / h));//□□□□□□□□ □□□□□□□□
```

```
                int nStrWidth =
m_fm.stringWidth(sAdd_String);//□□□□□□ □□□□□□□□
□□□□□□□□ □ □□□□□□□
                while (nStrWidth > m_nOffset_x - nL - 2)
                {
                    //  □□□□□□□  □□□□□□□□□□
□□□□□□□□ □□ □□y
                    sAdd_String = sAdd_String.substring(0,
sAdd_String.length() - 1);
                    nStrWidth = m_fm.stringWidth(sAdd_String);
                }
                g.drawString(sAdd_String, nLeftBound + m_nOffset_x -
nStrWidth - 8, nUpperBound + nHeight - m_nOffset_y - nMark + nSymb_Height /
2);
            }
            //  □□□□□□□□□ □□□□□x
            for (int nMark = 0; nMark < nWidth - m_nOffset_x; nMark +=
m_nScale)
            {
                int nL = 3;
                if (nMark % (2 * m_nScale) == 0)
                {
                    nL = 6;
                }
                g.drawLine(nLeftBound + m_nOffset_x + nMark,
nUpperBound + nHeight - m_nOffset_y, nLeftBound + m_nOffset_x + nMark,
nUpperBound + nHeight - m_nOffset_y + nL);
                String sAdd_String = Double.toString(x0 + nMark *
m_dStep);//□□□□□□□□ □□□□□□□□
```

```
                    int nStrWidth = m_fm.stringWidth(sAdd_String) /
2;//□□□□□□ □□□□□□□□ □□□□□□□□ □ □□□□□□□ / 2
                    while (nStrWidth > (m_nScale / 2) - 1)
                    {
                        //      □□□□□□□   □□□□□□□□□□
□□□□□□□□ □□ □□
                        sAdd_String      =      sAdd_String.substring(0,
sAdd_String.length() - 1);
                        nStrWidth = m_fm.stringWidth(sAdd_String) / 2;
                    }
                    g.drawString(sAdd_String, nLeftBound + m_nOffset_x +
nMark - nStrWidth, nUpperBound + nHeight - m_nOffset_y + nSymb_Height + 8);
                }

                ////////////////////////////////////////////////
                // □□□□□□□□□□ □□□□□□□ i ////////////////////////
                ////////////////////////////////////////////////

                ////////////////////////////////////////////////
                // □□□□□□□□□ □□□□□□□ ////////////////////////////
                ////////////////////////////////////////////////
                int nCellWidth = 70;
                int nLegendOffset = 5;
                int nGraphWidthForLegend = nWidth - nLegendOffset;
                int nCellHeight = (m_nOffset_y - nSymb_Height - 8) /
((FunctionList.m_nNumberOfFunctions / (nGraphWidthForLegend / nCellWidth)) +
1);

                {
```

```
//PaintLegendCell(g, nLeftBound + nLegendOffset + ((FunctionCounter) % (nGraphWidthForLegend / nCellWidth)) * nCellWidth , nUpperBound + nHeight - m_nOffset_y + nSymb_Height + 10 + ((FunctionCounter) / (nGraphWidthForLegend / nCellWidth)) * nCellHeight, nCellWidth, nCellHeight, graphNumber, Color.blue);
int xStart, yStart, xEnd, yEnd;
double dLengthRemainder = 0;
double y1, y2;
for (int ix = 0 ; ix < nWidth - m_nOffset_x; ix++)
{
    g.setColor(Color.blue);
    xStart = nLeftBound + m_nOffset_x + ix;
    yStart = nUpperBound + (int)(nHeight - m_nOffset_y - (f(graphNumber, x0 + ix * m_dStep) - ymin) * h);
    xEnd = nLeftBound + m_nOffset_x + ix + 1;
    yEnd = nUpperBound + (int)(nHeight - m_nOffset_y - (f(graphNumber, x0 + (ix + 1) * m_dStep) - ymin) * h);
    y1 = yStart;
    y2 = yEnd;
    // !!! mode of PunctureLine is switched off here
    //if(! m_bPunctureLine)
    {
        g.drawLine(xStart, yStart, xEnd, yEnd);
    }
    //else
    //{
    //    dLengthRemainder = drawPunctureLine(g, dLengthRemainder, xStart, yStart, xEnd, yEnd, 5 * (graphNumber - 1), Color.blue);
    //}
```

```
                                // draws strict line
                                xStart = nLeftBound + m_nOffset_x + ix;
                                yStart = nUpperBound + (int)(nHeight - m_nOffset_y
- (ordinate - ymin) * h);
                                xEnd = nLeftBound + m_nOffset_x + ix + 1;
                                yEnd = nUpperBound + (int)(nHeight - m_nOffset_y
- (ordinate - ymin) * h);
                                g.setColor(Color.red);
                                g.drawLine(xStart, yStart, xEnd, yEnd);
                                // check cross section
                                if (((y1 <= yStart)&(y2 > yEnd))|((y1 > yStart)&(y2
<= yEnd)))
                                {
                                        g.setColor(Color.black);
                                        g.drawString(Float.toString((float)(x0 + ix *
m_dStep)), nLeftBound + m_nOffset_x + ix, nUpperBound + (int)(nHeight -
m_nOffset_y - (ordinate - ymin) * h));
                                }

                                //////////////
                        }
                }
        }

            private void PaintFewDataFrame(Graphics g, int nLeftBound,
                    int nUpperBound, int nWidth, int nHeight, double x0, double x1)
            {
                    // m_nFunctionList - numbers of functions f_i(x)
                    // nLeftBound - x-coordinate of left-upper corner
```

```
                // nUpperBound - y-coordinate of left-upper corner
                // nWidth - width of rectangle for graph and legend
                // nHeight - height of rectangle for graph and legend

                // m_dStep - □□□□ □□□□ □□ x
                m_dStep = (x1 - x0) / (nWidth - m_nOffset_x);

                //□□□□□ □□□ □□□□□□ - □□□□
                g.setColor(Color.green);
                g.drawRect(nLeftBound, nUpperBound, nWidth, nHeight);
                g.setColor(Color.white);
                g.fillRect(nLeftBound + 1, nUpperBound + 1, nWidth - 2, nHeight
- 2);


                // y
                double ymin = f(FunctionList.m_head.Data(), x0);
                double ymax = f(FunctionList.m_head.Data(), x0);
                double y;
                for (FunctionList fl = FunctionList.m_head; fl != null; fl =
FunctionList.Next(fl))
                {
                        for (double x = x0; x < x1 ; x = x + m_dStep)
                        {
                                y = f(fl.Data(), x);
                                if (y < ymin)
                                {
                                        ymin = y;
                                }
```

```
                        if (y > ymax)
                        {
                                ymax = y;
                        }
                }
        }
        // ajustment of values for y axis
        double c = ymax - ymin;
        if (c < 1.e-6) c = 1.;
        ymin = ymin - 0.25*c;
        ymax = ymax       + 0.25*c;
        ////////////////////////////////
        // h - □□□□ □□□□ □□ y
        double h = (nHeight - m_nOffset_y) / (ymax - ymin);

        ////////////////////////////////////////////////
        // □□□□□□□□□ □□□□ □□□□□□□□□ □□□□ ////////////////
        ////////////////////////////////////////////////

        // □□□□□□□□□ □□□□
        g.setColor(Color.black);
        g.drawLine(nLeftBound + m_nOffset_x, nUpperBound + nHeight
- m_nOffset_y, nLeftBound + m_nOffset_x, nUpperBound);
        g.drawLine(nLeftBound + m_nOffset_x, nUpperBound + nHeight
- m_nOffset_y, nLeftBound + nWidth, nUpperBound + nHeight - m_nOffset_y);
        int nSymb_Height = m_fm.getHeight(); // □□□□□□
□□□□□□□
        // □□□□□□□□□ □□□□□       y
```

```
                for (int nMark = 0; nMark < nHeight - m_nOffset_y; nMark +=
m_nScale)
                {
                        int nL = 3;
                        if (nMark % (2 * m_nScale) == 0)
                        {
                                nL = 6;
                        }
                        g.drawLine(nLeftBound + m_nOffset_x, nUpperBound +
nHeight - m_nOffset_y - nMark, nLeftBound + m_nOffset_x - nL, nUpperBound +
nHeight - m_nOffset_y - nMark);
                        //String sAdd_String = Double.toString(ymin +
(double)(nMark / h));// value for y axis
                        String sAdd_String = Double.toString(Math.round((ymin +
(double)(nMark / h))*100.)/100.);// 2 digits after point
                    //String sAdd_String = String.valueOf(Math.round((ymin +
(double)(nMark / h))));// integer value
                        int                    nStrWidth                    =
m_fm.stringWidth(sAdd_String);//□□□□□□                □□□□□□□□
□□□□□□□□ □ □□□□□□□
                        while (nStrWidth > m_nOffset_x - nL - 2)
                        {
                                //        □□□□□□□    □□□□□□□□□□
□□□□□□□□ □□ □□y
                                sAdd_String        =        sAdd_String.substring(0,
sAdd_String.length() - 1);
                                nStrWidth = m_fm.stringWidth(sAdd_String);
                        }
```

```
                    g.drawString(sAdd_String, nLeftBound + m_nOffset_x -
nStrWidth - 8, nUpperBound + nHeight - m_nOffset_y - nMark + nSymb_Height /
2);
                }
                //      □□□□□□□□□ □□□□□ x
                for (int nMark = 0; nMark < nWidth - m_nOffset_x; nMark +=
m_nScale)
                {
                    int nL = 3;
                    if (nMark % (2 * m_nScale) == 0)
                    {
                        nL = 6;
                    }
                    g.drawLine(nLeftBound + m_nOffset_x + nMark,
nUpperBound + nHeight - m_nOffset_y, nLeftBound + m_nOffset_x + nMark,
nUpperBound + nHeight - m_nOffset_y + nL);
                    //String sAdd_String = Double.toString(x0 + nMark *
m_dStep);// value for x axis
                    //String sAdd_String = Double.toString(Math.round((x0 +
nMark * m_dStep)*100.)/100.);//2 digits after point
                    String sAdd_String = String.valueOf(Math.round((x0 + nMark
* m_dStep))); // integer values

                    int nStrWidth = m_fm.stringWidth(sAdd_String) /
2;//□□□□□□ □□□□□□□□ □□□□□□□□ □ □□□□□□□ / 2
                    while (nStrWidth > (m_nScale / 2) - 1)
                    {
                        //      □□□□□□□  □□□□□□□□□□
□□□□□□□□ □□ □□
```

```
                    sAdd_String = sAdd_String.substring(0,
sAdd_String.length() - 1);
                    nStrWidth = m_fm.stringWidth(sAdd_String) / 2;
                }
                g.drawString(sAdd_String, nLeftBound + m_nOffset_x +
nMark - nStrWidth, nUpperBound + nHeight - m_nOffset_y + nSymb_Height + 8);
            }


            ////////////////////////////////////////////////
            // □□□□□□□□□□ □□□□□□□ i ////////////////////////
            ////////////////////////////////////////////////

            ////////////////////////////////////////////////
            // □□□□□□□□□□ □□□□□□□ ////////////////////////////
            ////////////////////////////////////////////////
            //int nCellWidth = 70;
            int nCellWidth = 170; // width of legend cell
            int nLegendOffset = 5;
            int nGraphWidthForLegend = nWidth - nLegendOffset;
            int nCellHeight = (m_nOffset_y - nSymb_Height - 8) /
((FunctionList.m_nNumberOfFunctions / (nGraphWidthForLegend / nCellWidth)) +
1);
            ColorList cl = ColorList.m_head;
            int FunctionCounter = 0;
            // Puncture Line is switched off here
            m_bPunctureLine = false;
            ////////////////////////
```

```
for (FunctionList fl = FunctionList.m_head; fl != null; fl = FunctionList.Next(fl))
{
    g.setColor(cl.Data());
    PaintLegendCell(g, nLeftBound + nLegendOffset + ((FunctionCounter) % (nGraphWidthForLegend / nCellWidth)) * nCellWidth , nUpperBound + nHeight - m_nOffset_y + nSymb_Height + 10 + ((FunctionCounter) / (nGraphWidthForLegend / nCellWidth)) * nCellHeight, nCellWidth, nCellHeight, fl.Data(), cl.Data());
    FunctionCounter++;
    int xStart, yStart, xEnd, yEnd;
    double dLengthRemainder = 0;
    for (int ix = 0 ; ix < nWidth - m_nOffset_x; ix++)
    {
        xStart = nLeftBound + m_nOffset_x + ix;
        yStart = nUpperBound + (int)(nHeight - m_nOffset_y - (f(fl.Data(), x0 + ix * m_dStep) - ymin) * h);
        xEnd = nLeftBound + m_nOffset_x + ix + 1;
        yEnd = nUpperBound + (int)(nHeight - m_nOffset_y - (f(fl.Data(), x0 + (ix + 1) * m_dStep) - ymin) * h);
        if(! m_bPunctureLine)
        {
            //g.drawLine(xStart, yStart, xEnd, yEnd); // usual line
            drawBoldLine(g,xStart, yStart, xEnd, yEnd);
        }
        else
        {
```

```
                                    dLengthRemainder = drawPunctureLine(g,
dLengthRemainder, xStart, yStart, xEnd, yEnd, 5 * (fl.Data() - 1), cl.Data());
                                }
                            }
                            cl = ColorList.Next(cl);
                        }
                    }

                private void PaintLegendCell(Graphics g, int nLeftBound, int
nUpperBound, int nWidth, int nHeight, int nFunctionNumber, Color colLineColor)
                {
                    int nSymb_Height = m_fm.getHeight(); // □□□□□□
□□□□□□□
                    //String sCaption = "x" + Integer.toString(nFunctionNumber);
                    String sCaption = GraphCaptions(nFunctionNumber);
                    ////////////// □□□□□□□ □□□□□□□ □□□□□□□.
□□□□ □□□□ □□□□□□ □□□□□□□□ - □□□ □□□□□□
                    if (m_fm.stringWidth(sCaption) > nWidth) {
                    sCaption = sCaption.substring(0,
Math.round(nWidth/m_fm.getWidths()[0]));
                    }
                    //////////////
                    g.setColor(Color.black);
                    g.drawString(sCaption, (int)(nLeftBound + (3./4.) * nWidth -
m_fm.stringWidth(sCaption)), (int)(nUpperBound + (nHeight + nSymb_Height) /
2));
                    if (m_bPunctureLine)
                    {
```

```
                    drawPunctureLine(g, 0, nLeftBound, nUpperBound + (nHeight / 2), (int)(nLeftBound + (3./4.) * nWidth - m_fm.stringWidth(sCaption) - 5),  nUpperBound + (nHeight / 2), 5 * (nFunctionNumber - 1), colLineColor);
                }
                else
                {
                    g.setColor(colLineColor);
                    g.drawLine(nLeftBound, nUpperBound + (nHeight / 2),(int)(nLeftBound + (3./4.) * nWidth - m_fm.stringWidth(sCaption) - 5), nUpperBound + (nHeight / 2));
                }

            }


            protected void PaintDataFrame(Graphics g, int nFunctionNumber, int nLeftBound, int nUpperBound, int nWidth, int nHeight, double x0, double x1, Color GraphColor)
            {
                // nFunctionNumber - number of function f_i(x)
                // nLeftBound - x-coordinate of left-upper corner
                // nUpperBound - y-coordinate of left-upper corner
                // nWidth - width of rectangle for graph and legend
                // nHeight - height of rectangle for graph and legend

                // m_dStep - □□□□ □□□□ □□ x
                m_dStep = (x1 - x0) / (nWidth - m_nOffset_x);

                //□□□□□ □□□ □□□□□□ - □□□□
```

```
g.setColor(Color.yellow);
g.drawRect(nLeftBound, nUpperBound, nWidth, nHeight);
g.setColor(Color.white);
g.fillRect(nLeftBound + 1, nUpperBound + 1, nWidth - 2, nHeight
- 2);


//□□□□□□□□□□ □□□□□□ □□□□□□□ y
double ymin = f(nFunctionNumber, x0);
double ymax = f(nFunctionNumber, x0);
for (double x = x0; x < x1 ; x = x + m_dStep)
{
        double y = f(nFunctionNumber, x);
        if (y < ymin)
        {
           ymin = y;
        }
        if (y > ymax)
        {
                ymax = y;
        }
}
// ajustment of values for y axis
double c = ymax - ymin;
if (c < 1.e-6) c = 1.;
ymin = ymin - 0.25*c;
ymax = ymax        + 0.25*c;
////////////////////////////////
// h - □□□□ □□□□ □□ y
```

```
                double h = (nHeight - m_nOffset_y) / (ymax - ymin);
                ////////////////////////////////////////////////
                // □□□□□□□□□ □□□□ □□□□□□□□□ □□□□ ////////////////
                ////////////////////////////////////////////////

                // □□□□□□□□□ □□□□
                g.setColor(Color.black);
                g.drawLine(nLeftBound + m_nOffset_x, nUpperBound + nHeight - m_nOffset_y, nLeftBound + m_nOffset_x, nUpperBound);
                g.drawLine(nLeftBound + m_nOffset_x, nUpperBound + nHeight - m_nOffset_y, nLeftBound + nWidth, nUpperBound + nHeight - m_nOffset_y);
                int nSymb_Height = m_fm.getHeight(); // □□□□□□ □□□□□□□
                // Drawing axis y
                for (int nMark = 0; nMark < nHeight - m_nOffset_y; nMark += m_nScale)
                {
                        int nL = 3;
                        if (nMark % (2 * m_nScale) == 0)
                        {
                                nL = 6;
                        }
                        g.drawLine(nLeftBound + m_nOffset_x, nUpperBound + nHeight - m_nOffset_y - nMark, nLeftBound + m_nOffset_x - nL, nUpperBound + nHeight - m_nOffset_y - nMark);
                                double value = ymin + (double)(nMark / h);
                           // 2 digits after point
                             value = Math.round(value * 100.)/100.;
```

```
                    ////////////////////////////////
                        String sAdd_String = Double.toString(value);//double value
to print
                    //String                sAdd_String                =
String.valueOf(Math.round(value));//integer value to print
                        //String    sAdd_String    =    Double.toString(ymin    +
(double)(nMark / h));//□□□□□□□□ □□□□□□□□
                        int                    nStrWidth                =
m_fm.stringWidth(sAdd_String);//□□□□□□          □□□□□□□□
□□□□□□□□ □ □□□□□□□□
                        while (nStrWidth > m_nOffset_x - nL - 2)
                        {
                            //        □□□□□□□    □□□□□□□□□□
□□□□□□□□ □□ □□y
                            sAdd_String    =    sAdd_String.substring(0,
sAdd_String.length() - 1);
                            nStrWidth = m_fm.stringWidth(sAdd_String);
                        }
                        g.drawString(sAdd_String, nLeftBound + m_nOffset_x -
nStrWidth - 8, nUpperBound + nHeight - m_nOffset_y - nMark + nSymb_Height /
2);
                    }
                    //    Drawing axis x
                    for (int nMark = 0; nMark < nWidth - m_nOffset_x; nMark +=
m_nScale)
                    {
                        int nL = 3;
                        if (nMark % (2 * m_nScale) == 0)
                        {
```

```
                    nL = 6;
                }
                g.drawLine(nLeftBound + m_nOffset_x + nMark,
nUpperBound + nHeight - m_nOffset_y, nLeftBound + m_nOffset_x + nMark,
nUpperBound + nHeight - m_nOffset_y + nL);
                //String sAdd_String = Double.toString(x0 + nMark *
m_dStep);//□□□□□□□□ □□□□□□□□
                String sAdd_String = Double.toString(Math.round((x0 +
nMark * m_dStep)*100.)/100.);// 2 digits after point
                //String sAdd_String = String.valueOf(Math.round((x0 + nMark
* m_dStep)));// 0 digits after point
                int nStrWidth = m_fm.stringWidth(sAdd_String) /
2;//□□□□□□ □□□□□□□□ □□□□□□□□ □ □□□□□□□ / 2
                while (nStrWidth > (m_nScale / 2) - 1)
                {
                    //        □□□□□□□    □□□□□□□□□□
□□□□□□□□ □□ □□
                    sAdd_String        =        sAdd_String.substring(0,
sAdd_String.length() - 1);
                    nStrWidth = m_fm.stringWidth(sAdd_String) / 2;
                }
                g.drawString(sAdd_String, nLeftBound + m_nOffset_x +
nMark - nStrWidth, nUpperBound + nHeight - m_nOffset_y + nSymb_Height + 8);
            }


            ////////////////////////////////////////////////
            // □□□□□□□□□□ □□□□□□□ ////////////////////////
            ////////////////////////////////////////////////
```

```
                if (m_bLegend && !m_bCaptions)
                {
                        // minimal  value
                        String sAdd_String = "Minimal value = " +
Double.toString(ymin);//
                        int nStrWidth =
m_fm.stringWidth(sAdd_String);//□□□□□□
                        while (nStrWidth > (nWidth / 2) - 10)
                        {
                                // □□□□□□□
                                sAdd_String = sAdd_String.substring(0,
sAdd_String.length() - 1);
                                nStrWidth = m_fm.stringWidth(sAdd_String);
                        }

                        g.drawString(sAdd_String, nLeftBound + (int)((nWidth / 4.)
- (nStrWidth / 2.)), nUpperBound + (int)((nHeight / 3.) + (2. * (nHeight -
m_nOffset_y + nSymb_Height + 8.) / 3.) + (nSymb_Height / 2.)));


                        // left bound
                        sAdd_String = "Left bound = " + Double.toString(x0);
                        nStrWidth =
m_fm.stringWidth(sAdd_String);//□□□□□□
                        while (nStrWidth > (nWidth / 2) - 10)
                        {
                                // □□□□□□□
                                sAdd_String = sAdd_String.substring(0,
sAdd_String.length() - 1);
```

```
                nStrWidth = m_fm.stringWidth(sAdd_String);
            }

            g.drawString(sAdd_String, nLeftBound + (int)(((3. *
nWidth) / 4.) - (nStrWidth / 2.)),nUpperBound + (int)((nHeight / 3.) + (2. * (nHeight
- m_nOffset_y + nSymb_Height + 8.) / 3.) + (nSymb_Height / 2.)));

            // maximal value
            sAdd_String = "Maximal value = " +
Double.toString(ymax);
            nStrWidth =
m_fm.stringWidth(sAdd_String);//□□□□□□
            while (nStrWidth > (nWidth / 2) - 10)
            {
                // □□□□□□□
                sAdd_String = sAdd_String.substring(0,
sAdd_String.length() - 1);
                nStrWidth = m_fm.stringWidth(sAdd_String);
            }

            g.drawString(sAdd_String, nLeftBound + (int)((nWidth / 4.)
- (nStrWidth / 2.)),nUpperBound + (int)(((2. * nHeight) / 3.) + (nHeight -
m_nOffset_y + nSymb_Height + 8.) / 3. + (nSymb_Height / 2.)));

            // right bound
            sAdd_String = "Right bound = " + Double.toString(x1);
            nStrWidth =
m_fm.stringWidth(sAdd_String);//□□□□□□
            while (nStrWidth > (nWidth / 2) - 10)
```

```
                    {
                        // □□□□□□□
                        sAdd_String = sAdd_String.substring(0,
sAdd_String.length() - 1);
                        nStrWidth = m_fm.stringWidth(sAdd_String);
                    }

                    g.drawString(sAdd_String, nLeftBound + (int)((3. * nWidth
/ 4.) - (nStrWidth / 2.)),nUpperBound + (int)(((2. * nHeight) / 3.) + (nHeight -
m_nOffset_y + nSymb_Height + 8.) / 3. + (nSymb_Height / 2.)));

                }
                if (m_bLegend && m_bCaptions)
                {
                    String sAdd_String = GraphCaptions(nFunctionNumber);
                    int nStrWidth =
m_fm.stringWidth(sAdd_String);//□□□□□□
                    //while (nStrWidth > (nWidth / 2) - 10)
                    while (nStrWidth > nWidth - 2)
                    {
                        // □□□□□□□
                        sAdd_String = sAdd_String.substring(0,
sAdd_String.length() - 1);
                        nStrWidth = m_fm.stringWidth(sAdd_String);
                    }

                    g.drawString(sAdd_String, nLeftBound + (int)((nWidth -
nStrWidth) / 2.),nUpperBound + (int)(((2. * nHeight) / 3.) + (nHeight - m_nOffset_y
+ nSymb_Height + 8.) / 3. + (nSymb_Height / 2.)));
```

```
                }

                ////////////////////////////////////////////////
                // □□□□□□□□□ □□□□□□□ ////////////////////////////
                ////////////////////////////////////////////////
                g.setColor(GraphColor);
                int xStart, yStart, xEnd, yEnd;
            for (int ix = 0 ; ix < nWidth - m_nOffset_x; ix++)
                {
                        xStart = nLeftBound + m_nOffset_x + ix;
                        yStart = nUpperBound + (int)(nHeight - m_nOffset_y -
(f(nFunctionNumber, x0 + ix * m_dStep) - ymin) * h);
                        xEnd = nLeftBound + m_nOffset_x + ix + 1;
                        yEnd = nUpperBound + (int)(nHeight - m_nOffset_y -
(f(nFunctionNumber, x0 + (ix + 1) * m_dStep) - ymin) * h);

                        //g.drawLine(xStart, yStart, xEnd, yEnd);
                        drawBoldLine(g, xStart, yStart, xEnd, yEnd);
                }
        }

            public void drawBoldLine(Graphics g, int xStart, int yStart, int xEnd,
                int yEnd)
            {
                g.drawLine(xStart, yStart, xEnd, yEnd);
                g.drawLine(xStart - 1, yStart - 1, xEnd - 1, yEnd - 1);
            }
```

```
        ////////////////////////////////////////////////////
        // □□□□□□□□□ □□□□□□□□□ □? ////////////////////////
        ////////////////////////////////////////////////////
        private double drawPunctureLine(Graphics g, double dLengthRemainder, int x0, int y0, int x1, int y1, double dPieceLength, Color colLineColor)
        {
            dPieceLength = Math.abs(dPieceLength);
            g.setColor(colLineColor);
            if (dPieceLength == 0)
            {
                g.drawLine(x0, y0, x1, y1);
                return 0;
            }
            else
            {
                int nDirection_x = x1 - x0;
                int nDirection_y = y1 - y0;
                double dLineLength = lineLength(x0, y0, x1, y1);
                if (Math.abs(dLengthRemainder) > dLineLength)
                {
                    if(dLengthRemainder > 0)
                    {
                        g.drawLine(x0, y0, x1, y1);
                        return dLengthRemainder - dLineLength;
                    }
                    else
```

```
                {
                    return -(Math.abs(dLengthRemainder) -
dLineLength);
                }
            }
            else
            {
                boolean bDraw = true;
                int x1_new = x0 +
(int)((Math.abs(dLengthRemainder) / dLineLength) * nDirection_x);
                int y1_new = y0 +
(int)((Math.abs(dLengthRemainder) / dLineLength) * nDirection_y);
                if(dLengthRemainder > 0)
                {
                    g.drawLine(x0, y0, x1_new, y1_new);
                    bDraw = false;
                }
                int x0_new = x1_new;
                int y0_new = y1_new;
                double dCurrentLengthRemainder = dLineLength -
Math.abs(dLengthRemainder);
                while(dCurrentLengthRemainder >= dPieceLength)
                {
                    x1_new = x0_new + (int)((dPieceLength /
dLineLength) * nDirection_x);
                    y1_new = y0_new + (int)((dPieceLength /
dLineLength) * nDirection_y);
                    if(bDraw)
                    {
```

```
                                        g.drawLine(x0_new, y0_new, x1_new,
y1_new);
                                    }
                                    x0_new = x1_new;
                                    y0_new = y1_new;
                                    bDraw = ! bDraw;
                                    dCurrentLengthRemainder =
dCurrentLengthRemainder - dPieceLength;
                                }
                                // Finally, for dCurrentLengthRemainder >= 0
                                if(bDraw)
                                {
                                    g.drawLine(x0_new, y0_new, x1, y1);
                                    return (dPieceLength -
dCurrentLengthRemainder);
                                }
                                else
                                {
                                    return (dCurrentLengthRemainder -
dPieceLength);
                                }
                            }
                        }
                    }



                    private double lineLength(int x0, int y0, int x1, int y1)
                    {
```

```
        return Math.sqrt(Math.pow(x1 - x0, 2) + Math.pow(y1 - y0, 2));
}

public void start()
{
        if (m_Graphics == null)
        {
                m_Graphics = new Thread(this);
                m_Graphics.start();
        }
}

public void stop()
{
        if (m_Graphics != null)
        {
                m_Graphics.stop();
                m_Graphics = null;
        }
}

public void run()
{
        int     nSleepTime = (int)(1000 / m_fps);

        while (true)
        {
                try
                {
```

```
                repaint();

                //      □□□□□□□□□□□□□□□□      □□
□□□□□□□□ fps
                Thread.sleep(nSleepTime);
            }
            catch (InterruptedException e)
            {
                stop();
            }
        }
    }

    public boolean mouseDrag(Event evt, int x, int y)
    {
        m_bDrag = true;
        if ((m_bBeginDrag == true) & !m_bMarkGraphs &
!m_bWholeScreenMode)
        {
            m_dimCursorLocBegin = new Dimension(x, y);
            if(x - (x / m_nGraphWidth) * m_nGraphWidth <
m_nOffset_x)
            {
                m_dimCursorLocBegin.width = (x /
m_nGraphWidth) * m_nGraphWidth + m_nOffset_x;                ;
            }
            if(y - (y / m_nGraphHeight) * m_nGraphHeight >
m_nGraphHeight - m_nOffset_y)
            {
```

```
                    m_dimCursorLocBegin.height = (y / m_nGraphHeight) * m_nGraphHeight + m_nGraphHeight - m_nOffset_y;
            ;
                }
                m_bBeginDrag = false;
            }
            ////////////////////////////////////////////////////////////////////////
            int k_width = m_dimCursorLocBegin.width / m_nGraphWidth;
            int k_height = m_dimCursorLocBegin.height / m_nGraphHeight;
            m_dimDragEvtLoc = new Dimension(x, y);
            if(x / m_nGraphWidth > k_width)
            {
                m_dimDragEvtLoc.width = m_nGraphWidth * (k_width + 1);
            }
            if(x - k_width * m_nGraphWidth < m_nOffset_x)
            {
                m_dimDragEvtLoc.width = k_width * m_nGraphWidth + m_nOffset_x;
            }
            if(y / m_nGraphHeight < k_height)
            {
                m_dimDragEvtLoc.height = m_nGraphHeight * k_height;

            }
            if(y - k_height * m_nGraphHeight > m_nGraphHeight - m_nOffset_y)
            {
```

```
                    m_dimDragEvtLoc.height = k_height * m_nGraphHeight +
m_nGraphHeight - m_nOffset_y;
                }
                ////////////////////////////////////////////////////////////////////////
                repaint();
                return true;  // □□□□□ □□□□□□□ □□□

        }

        public boolean mouseUp(Event evt, int x, int y)
        {
                m_nAreaNumber = ((x / m_nGraphWidth) + 1) + (y /
m_nGraphHeight)*(size().width / m_nGraphWidth);
                if (m_bWholeScreenChosen)
                {
                        m_bWholeScreenMode = false;
                        m_bWholeScreenChosen = false;
                        repaint();
                        m_button.show();
                        m_buttonFewGraphs.show();
                        m_buttonWholeScreen.setLabel("Whole screen");
                        m_buttonWholeScreen.show();

                }

                if (m_bWholeScreenMode & (m_nAreaNumber >= 1) &
(m_nAreaNumber <= m_nGraphCount + 1))
                {
                        m_bWholeScreenChosen = true;
                        m_button.hide();
```

```
                m_buttonFewGraphs.hide();
                m_buttonWholeScreen.hide();
                repaint();
            }

            if (m_bMarkGraphs & (m_nAreaNumber >= 1) &
(m_nAreaNumber <= m_nGraphCount))
            {
                m_nFunctionList.Add(((x / m_nGraphWidth) + 1) + (y /
m_nGraphHeight)*(size().width / m_nGraphWidth));
                repaint();
            }

            m_bDrag = false;
            if ((m_bBeginDrag == false) & !m_bMarkGraphs &
!m_bWholeScreenMode)
            {
                int k_width = m_dimCursorLocBegin.width /
m_nGraphWidth;
                int k_height = m_dimCursorLocBegin.height /
m_nGraphHeight;

                m_dimCursorLocEnd = new Dimension(x, y);
                if(x / m_nGraphWidth > k_width)
                {
                    m_dimDragEvtLoc.width = m_nGraphWidth *
(k_width + 1);
                }
                if(x - k_width * m_nGraphWidth < m_nOffset_x)
```

```
                {
                    m_dimDragEvtLoc.width = k_width *
m_nGraphWidth + m_nOffset_x;
                }
                if(y / m_nGraphHeight > k_height)
                {
                    m_dimDragEvtLoc.height = m_nGraphHeight *
(k_height + 1);
                }
                if(y - k_height * m_nGraphHeight > m_nGraphHeight -
m_nOffset_y)
                {
                    m_dimDragEvtLoc.height = k_height *
m_nGraphHeight + m_nGraphHeight - m_nOffset_y;
                }

                m_bBeginDrag = true;

                //  □□□  □□□  □□□□□□□□□□
□□□□□□□
                m_x1 = m_x0 + (m_dimCursorLocEnd.width -
(m_dimCursorLocEnd.width / m_nGraphWidth) * m_nGraphWidth - m_nOffset_x)
* m_dStep;
                m_x0 = m_x0 + (m_dimCursorLocBegin.width -
(m_dimCursorLocBegin.width / m_nGraphWidth) * m_nGraphWidth -
m_nOffset_x) * m_dStep;
                if (m_x0 != m_x1)
                {
                    if (m_x0 > m_x1)
```

```
                    {
                        m_x0 = m_x0 + m_x1;
                        m_x1 = m_x0 - m_x1;
                        m_x0 = m_x0 - m_x1;
                    }

                    // □□□□□□□□ □□□□□ □□□□□□□
□ □□□□□□ □□"□□□□□□□ □□□□□□
                    BoundsLocation bl = new BoundsLocation();
                    bl.m_dLeftBound = m_x0;
                    bl.m_dRightBound = m_x1;
                    m_llCursorLocations.Add(bl);
                    ////////////////////////////////////////////////

                    repaint();    //    □□□□□□□□□□□□□
□□□□□□□ □ □□□□□□ □□□□□□□□
                }
            }
            return true; // □□□□ □□□□□□□ □□□

        }


        public boolean action(Event event, Object obj)
        {
            Object oTarget = event.target;
            if (oTarget instanceof Button)
            {
```

```
                    Button buttonTarget = (Button)oTarget;
                    String sButtonString = buttonTarget.getLabel();
                    if (sButtonString.compareTo("Back") == 0)
                    {
                        //  □□□  □□□  □□□□□□□□□□
□□□□□□□
                        m_llCursorLocations =
LinkedList.Next(m_llCursorLocations);
                        m_x1 =
m_llCursorLocations.Data().m_dRightBound;
                        m_x0 = m_llCursorLocations.Data().m_dLeftBound;

                        repaint();  //  □□□□□□□□□□□□
□□□□□□□ □ □□□□□□ □□□□□□□
                        return true;
                    }
                    if (sButtonString.compareTo("Mark few graphs") == 0)
                    {
                        m_buttonFewGraphs.setLabel("Marking...");
                        FunctionList.m_head = null;
                        FunctionList.m_nNumberOfFunctions = 0;
                        m_bMarkGraphs = true;

                        repaint();
                    }
                    if (sButtonString.compareTo("Marking...") == 0)
                    {
                        m_buttonFewGraphs.setLabel("Mark few graphs");
                        m_bMarkGraphs = false;
```

```
                }
                if (sButtonString.compareTo("Whole screen") == 0)
                {
                        m_bWholeScreenMode = true;
                        m_buttonWholeScreen.setLabel("Point graph");
                }

        }
        return false;
}


private void ResizeImage()
{
        // □□□□□□□□ □□□□□ □□□□ □□□□□□
        Dimension dim = size();
        int nWidth = dim.width;
        int nHeight = dim.height;


        // □□□□□□□□□ □□□□ □ □□□□□□□
□□□□□□ □□□□□□□□□□□;
        // □□□□ □□ □□ □□□ □□□□□□□ ...
        if(m_dimImage != null &&
                m_dimImage.width == nWidth &&
                m_dimImage.height == nHeight)
        {
                //      ... □□ □□□□□□ □□□□□
                return;
```

```
            }
            //          □□□□□□□□          □□□□□□□□
□□□□□□□□□□ □□□ □□□□□□
            m_dimImage = new Dimension(nWidth, nHeight);
            m_image = createImage(nWidth, nHeight);
            m_g = m_image.getGraphics();

        }

    }
```

```
    package medbioinvestigations.fde;
    //**********************************************************
**********
    // DelaySystemSolution.java:    Applet
    //
    //**********************************************************
**********
    import java.applet.*;
    import java.awt.*;

    //==========================================================
======================
    // Main Class for applet DelaySystemSolution
    //
    //==========================================================
======================
    public class DelaySystemSolution
```

```
{

    // SOLUTION (initial value and solution at xend) ///////////
    public double[] y;
    ////////////////////////////////////////////////////////

    //  POSITS ////////////////////////////////////////////
    // Meaning of these variables:
    // ifirst       lowest step number still in memory coef;
    // last         address of last data written by store on common block coef
    //                      Must be set to 0 in the calling program
    //                      before the first call.
    // x0           initial point, must be set  in the calling program
    //                      before the first call.
    // xlast        = x + h of last written step;
    // ipos         position of last successful search in function ylag;
    // disc         logical variable, necessary for the distinction
    //                      of k7 and k1 of the following step in the case
    //                      when y(x0) is different from phi(x0).

    public int ifirst;
    public int last;
    public double x0;
    double xlast;
    int ipos;
    boolean disc;

    ////////////////////////////////////////////
```

```
  // STAT contains statistical information:
  ////////////////////////////////////////////
  //       nfcn   number of function evaluations
  // nstep        number of computed steps
  // naccpt       number of accepted steps
  // nrejct       number of rejected steps

  int nfcn;
  int nstep;
  int naccpt;
  int nrejct;

  ////////////////////////////////////////////
  // UROUND   smallest number satisfying 1. + uround > 1.
  //                      (to be adapted by user)
  ////////////////////////////////////////////


  ////////////////////////////////////////////
  // COEF contains coefficients for global solution
  ////////////////////////////////////////////


  public final int nn = 4;
  //public final int mxst = 800;
 public final int mxst = 1800;
  public double xstor[];
  public double ystor[][];
  public double c1[][];
```

```
public double c2[][];
public double c3[][];
public double c4[][];

final double uround = 5.e-8;

////////////////////////////////////////////////
// SIZE OF SYSTEM  ////////////////////////////
////////////////////////////////////////////////
public int n;
////////////////////////////////////////////////




//////////////////////////////////////////////////////////////////
// This method return value of system right-side f(x,y(x),y(x-tau))
// for given (x,y)    ////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
synchronized public double[] fcn(double x, double y[])
{
        double[] dRight_side = new double[Math.max(n, nn) + 1];
        // □□□□□ □□□□□□□ □□ □□□□□□□□□□□
        dRight_side[1] = - y[1];
        dRight_side[2] = - y[2];
        dRight_side[3] = - y[3];
        dRight_side[4] = - y[4];
        dRight_side[5] = - ylag(2,x-0.5);
        dRight_side[6] =  - y[5];
        return dRight_side;
```

```
}

public double phi(int i, double x)
{
        switch (i)
        {
                case 1:
                        return Math.max(0,1.0e-6);
                case 2:
                        return 0.561;
                case 3:
                        return 0.712;
                case 4:
                        return 2.021;
                case 5:
                        return 49.850;
                case 6:
                        return 297.3;
                case 7:
                        return 148.6;
                case 8:
                        return 74.2;
                case 9:
                        return 33.618;
                case 10:
                        return 0.;
                default:
                        return 1.;
        }
```

```
}


synchronized public void retard(int n, double x, double y[], double xend,
double eps, double hmax, double h)
{


    // Numerical solution of a system of first order
    // retarded differential equations y'=f(x,y(x),y(x-tau),...).
    // This is based on an embedded Runge-Kutta method of order
(4)5
    // due to Dormand & Prince (with stepsize control).
    // C. F. sections II.5 and II.15
    //
    //
    //
    // INPUT PARAMETERS
    ////////////////////////////////////////
    // n                dimension of the system (n <= 51)
    // fcn              name of void function computing the
    //                          first derivative f(x,y)
    // x                initial x-value
    // xend                     final x-value  (xend > x)
    // double[] y initial values for y
    // eps              local tolerance
    // hmax                     maximal step size
    // h                initial step size guess
```

```
//
//
// OUTPUT PARAMETERS
//////////////////////////////////////////
// double y                    solution at xend
//
//////////////////////////////////////////

double[] k1 = new double[Math.max(n, nn) + 1];
double[] k2 = new double[Math.max(n, nn) + 1];
double[] k3 = new double[Math.max(n, nn) + 1];
double[] k4 = new double[Math.max(n, nn) + 1];
double[] k5 = new double[Math.max(n, nn) + 1];
double[] k6 = new double[Math.max(n, nn) + 1];
double[] k7 = new double[Math.max(n, nn) + 1];
double[] y1 = new double[Math.max(n, nn) + 1];
boolean reject;
double xph;
double err;
double demon;
double fac;
double hnew;
int iadr;

int nmax = 3000; // maximal number of steps

// initial preparations
//////////////////////////////////////////
```

```
hmax = Math.abs(hmax);
h = Math.min(Math.max(1.e-4, Math.abs(h)), hmax);
h = h / Math.abs(h); //signum of h
eps = Math.max(eps, 7. * uround);
reject = false;
naccpt = 0;
nrejct = 0;
nfcn = 1;
nstep = 0;
disc = true;
k1 = fcn(x, y);

if (! disc)
{
        k1 = fcn(x, y);
}

// basic integration step
////////////////////////////////////////////
// NewStepLabel:
disc = true;
while (!((nstep > nmax) | (x + .1 * h == x)))
{
        if ((x - xend) + uround > 0.)
        {
                return;
        }
        if ((x + h - xend) > 0.)
        {
```

```
            h = xend - x;
        }
        nstep = nstep + 1;

        // the 7 Runge - Kutta stages
        for (int i = 1; i <= n; i++)
        {
            y1[i] = y[i] + h * .2 * k1[i];
        }
        k2 = fcn(x + .2 * h, y1);
        for (int i = 1; i <= n; i++)
        {
            y1[i] = y[i] + h * ((3./40.) * k1[i] + (9./40.) * k2[i]);
        }
        k3 = fcn(x + .3 * h, y1);
        for (int i = 1; i <= n; i++)
        {
            y1[i] = y[i] + h * ((44./45.) * k1[i] - (56./15.) * k2[i] +
(32./9.) * k3[i]);
        }
        k4 = fcn(x + .8 * h, y1);
        for (int i = 1; i <= n; i++)
        {
            y1[i] = y[i] + h * ((19372./6561.) * k1[i] -
(25360./2187.) * k2[i] + (64448./6561.) * k3[i] - (212./729.) * k4[i]);
        }
        k5 = fcn(x + (8./9.) * h, y1);
        for (int i = 1; i <= n; i++)
        {
```

```
                    y1[i] = y[i] + h * ((9017./3168.) * k1[i] - (355./33.) *
k2[i] + (46732./5247.)* k3[i] + (49./176.) * k4[i] - (5103./18656.) * k5[i]);
                }
                xph = x + h;
                k6 = fcn(xph, y1);
                for (int i = 1; i <= n; i++)
                {
                    y1[i] = y[i] + h * ((35./384.) * k1[i] + (500./1113.) *
k3[i] + (125./192.) * k4[i] - (2187./6784.) * k5[i] + (11./84.) * k6[i]);
                }
                disc = true;
                k7 = fcn(xph, y1);
                for (int i = 1; i <= n; i++)
                {
                    k2[i] = ((71./57600.) * k1[i] - (71./16695.) * k3[i] +
(71./1920.) * k4[i] - (17253./339200.) * k5[i] + (22./525.) * k6[i] - (1./40.) * k7[i]) *
h;
                }
                nfcn = nfcn + 6;

                // error estimation

                err = 0.;
                for (int i = 1; i <= n; i++)
                {
                    demon = Math.max(Math.max(1.e-5,
Math.abs(y1[i])), Math.max(Math.abs(y[i]), 2. * uround / eps));
                    err = err + Math.pow(k2[i] / demon, 2);
                }
```

```
                    err = Math.sqrt(err / (double)n);

                    // computation of hnew
                    // We require  .2 <= hnew / h <= 10.

                    fac = Math.max(.1, Math.min(5., Math.pow(err / eps, 1./5.) /
.9));
                    hnew = h / fac;
                    if (err <= eps)
                    {
                        // step is accepted
                        naccpt = naccpt + 1;
                        //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                        store(x, xph, k1, k3, k4, k5, k6);//////////////////
                        //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                        //g.setColor(Color.white);
                        //g.fillRect(90, 20, 30, 20);
                        //g.setColor(Color.black);
                        //g.drawString(Double.toString(fac), 90, 40);
                        for (int i = 1; i <= n; i++)
                        {
                            k1[i] = k7[i];
                            y[i] = y1[i];
                        }
                        // recompute k1 in the case of discontinuous initial
phase
                        if (! disc)
                        {
                            k1 = fcn(xph, y);
```

```
                    }
                    x = xph;
                    if (Math.abs(hnew) > hmax)
                    {
                        hnew = nmax;
                    }
                    if (reject)
                    {
                        hnew = Math.min(Math.abs(hnew),
Math.abs(h));
                    }
                    reject = false;
                }
                else
                {
                    // step is rejected
                    reject = true;
                    if (naccpt >= 1)
                    {
                        nrejct = nrejct + 1;
                    }
                } // end if

                h = hnew;


            //      break NewStepLabel;
            disc = true;
            }
```

```
                // fail exit
                //g.drawString("exit of retard at x = " + Double.toString(x), 10,
20);
                // return;
        }


        synchronized private void store(double x, double xph, double fg1[],
double fg3[], double fg4[], double fg5[], double fg6[])
        {
                int iadr;



                last = last + 1;
                ifirst = Math.max(1, last - mxst + 1);
                iadr = ((last - 1) % mxst) + 1;
                xlast = xph;

                xstor[iadr] = x;

                for (int i = 1; i <= Math.max(n, nn); i++)
                {
                        ystor[i][iadr] = y[i];
                        c1[i][iadr] = fg1[i];
                        c2[i][iadr] = -(1337./480.) * fg1[i] + (105400./27825.) *
fg3[i] - (135./80.) * fg4[i] - (54675./212000.) * fg5[i] + (66./70.) * fg6[i];
                        c3[i][iadr] = (1039./360.) * fg1[i] - (468200./83475.) *
fg3[i] + (9./2.) * fg4[i] + (400950./318000.) * fg5[i] - (638./210.) * fg6[i];
```

```
                    c4[i][iadr] = -(1163./1152.) * fg1[i] + (37900./16695.) *
fg3[i] - (415./192.) * fg4[i] - (674325./508800.) * fg5[i] + (374./168.) * fg6[i];

            }

            return;
        }

        public synchronized double ylag(int i, double x)
        {
            int iadr;
            double h;
            double s;

            // initial phase
            if (disc)
            {
                if (Math.abs(x - x0) <= (3. * uround * Math.abs(x)))
                {
                    disc = false;
                }
                if (x <= x0)
                {

                    return phi(i, x);
                }
            }

            // compute the position of x
```

```
                if (x < xstor[ifirst])
                {
                        //g.drawString("Memory full, mxst = " +
Integer.parseInt(mxst), 10, 30);
                        //stop;
                        //return 0; // □□
                        ////////////////
                }

                ipos = Math.max(ifirst, Math.min(last, ipos));
                iadr = ((ipos - 1) % mxst) + 1;
                while ((x < xstor[iadr]) & (ipos > ifirst))
                {
                        ipos = ipos - 1;
                        iadr = ((ipos - 1) % mxst) + 1;
                }
                iadr = (ipos % mxst) + 1;
        Cycle:
                while(ipos < last)
                {
                        if (x > xstor[iadr])
                        {
                                ipos = ipos + 1;
                                iadr = (ipos % mxst) + 1;
                                continue;
                        }
                        else
                        {
                                break Cycle;
```

```
            }
        }

        // Compute the desired approximation

        iadr = ((ipos - 1) % mxst) + 1;
        if (ipos == last)
        {
            h = xlast - xstor[iadr];
        }
        else
        {
            h = xstor[(ipos % mxst) + 1] - xstor[iadr];
        }

        s = (x - xstor[iadr]) / h;


        return ystor[i][ipos] + h * s * (c1[i][ipos] + s * (c2[i][ipos] + s *
(c3[i][ipos] + s * c4[i][ipos])));

    }
    // DelaySystemSolution Class Constructor
    //-------------------------------------------------------------------------

    public DelaySystemSolution(int arg_n, double dDelay, double hmax)
    {
        // GENERAL INITALIZATION ////////////////////
```

```
xstor = new double[mxst + 1];
ystor = new double[nn + 1][mxst + 1];
c1 = new double[nn + 1][mxst + 1];
c2 = new double[nn + 1][mxst + 1];
c3 = new double[nn + 1][mxst + 1];
c4 = new double[nn + 1][mxst + 1];

n = arg_n;
x0 = 0;
double x1 = 1;
last = 0;
double x;
double  h = 0.5;
y = new double[Math.max(n, nn) + 1];
for(int i = 1; i <= Math.max(n, nn); i++)
{
        if(i <= n)
        {
                y[i] = phi(i, x0);
                //y[i] = .1;
        }
        else
        {
                y[i] = 0;
        }
}
for (double i = (int)x0; i <= (int)x1; i += dDelay)
{
        x = i - dDelay;
```

```
                double xend = (double)(i);
                double eps = 1.e-6;
                retard(n, x, y, xend, eps, hmax, h);

        }
    }

    public DelaySystemSolution()
    {
        ////////   Sample constructor for child classes
    }

    public double correlationDimension(int limit_number_of_points,
        double limit_distance_between_points, double initial_time,
        double last_time)
    {
        double [][] arrayOfPoints = new double [limit_number_of_points
+ 1]
                [n+1];
        double t;
        for(int i=1; i<=limit_number_of_points; i++)
        {
            t = initial_time + Math.random()*(last_time-initial_time);
            for(int j=1; j<=n; j++)
            arrayOfPoints[i][j] = ylag(j,t);
        }
        double h=0;
        for(int i=1; i<=limit_number_of_points; i++)
            for(int j=i+1; j<=limit_number_of_points; j++)
```

```
                    {
                        double distance = 0;
                        for(int k=1; k<=n; k++)
                            distance = distance +
                                Math.pow(arrayOfPoints[i][k]-
arrayOfPoints[j][k],2);
                        distance = Math.sqrt(distance);
                        if(distance <= limit_distance_between_points)
                            h = h+1;
                    }
                double C=h/(limit_number_of_points*limit_number_of_points);
                if(C<=0) C=1.;
                if(limit_distance_between_points<1.e6)
limit_distance_between_points = 1.e6;
                return Math.log(C)/Math.log(limit_distance_between_points);
            }
```