Західноукраїнський національний університет
Фаультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

НИКОЛИН Іванна Павлівна

АЛГОРИТМИ ФОРМУВАННЯ БАЗИ ДАНИХ ІМУНОГІСТОХІМІЧНИХ ЗОБРАЖЕНЬ / ALGORITHMS OF IMMUNOHISTOCHEMICAL IMAGES DATABASE FORMATION

спеціальність; 123 – Комп'ютерна інженерія
освітньо-професійна програма - Комп'ютерна інженерія

Кваліфікаційна робота

Виконав студент групи КІзм-21
НИКОЛИН Іванна Павлівна
_____

Науковий керівник:
д.т.н., професор, О.М. Березький
_____

Кваліфікаційну роботу
допущено до захисту
«___» _____2021 р.

Завідувач кафедри КІ
О.М.Березький
_____

Тернопіль – 2021

ЗМІСТ

ВСТУП

Актуальність теми. Імуногістохімічні зображення використовуються при уточненій постановці діагнозу онкологічних захворювань. Цитологічні зображення окремих клітин дають змогу визначати патологічні зміни окремих клітин. Гістологічні зображення дозволять слідкувати за змінами тканин органів людини або тварин. В роботі сфомовано базу даних імуногістохімічних зображень раку молочної залози. Від цього виду раку спостерігається найбільша смертність жінок на планеті. Отже, формування бази даних цих зображень з метою подальшого опрацювання є актуальною задачею.

Питання формування баз даних біомедичних зображень розглядалися в наукових працях викладачів кафедри комп'ютерної інженерії Західноукраїнського національного університету [1-7]. Зокрема, в роботах [5, 7] досліджені нечіткі бази даних та знань, в роботах [2, 6] – інформаційно-аналітичні. База цифрових гістологічних та цитологічних зображень передракових та ракових станів молочної залози «BPCI2100» описана в роботі [1], «CIFDB» – в роботі [2].

Мета і завдання дослідження. Метою роботи є розробка алгоритму алгоритму обчислення числових характеристик статистичних вибірок морфометричних даних.

Об'єкт дослідження – процес формування бази даних зображень.

Предмет дослідження – алгоритми формування бази даних в ImageJ.

Для досягнення поставленої мети необхідно розв'язати такі задачі:

- проаналізувати прорамну систему ImageJ;
- проаналізувати імуногістохімічні зображення;
- розробити алгоритм обчислення числових характеристик статистичних вибірок морфометричних даних;
- розробити базу даних зображень в середовищі ImageJ.

Методи дослідження базуються на використанні теорії баз даних, технології Java.

Наукова новизна одержаних результатів. Розроблено алгоритм обчислення числових характеристик статистичних вибірок морфометричних даних.

Практичне значення отриманих результатів. Розроблено базу даних імуногістохімічних зображень.

Публікації та апробація КР. Результати досліджень кваліфікаційної роботи опубліковано в двох тезах доповідей на V-ій науково-практичній конференції молодих вчених і студентів «Інтелектуальні комп'ютерні системи та мережі» (2 грудня 2021 р., м. Тернопіль, Західноукраїнський національний університет) [8, 9]:

1. Галан В. Ю., Мачуляк М. В., Іпіроті В. О., Николин І. П. Моделювання знань в системах медичної діагностики. Інтелектуальні комп'ютерні системи та мережі : тези доп. V Наук.-практ. конф. молодих вчених і студентів (2 груд. 2021 р.). Тернопіль : ЗУНУ, 2021. С. 13.

2. Мачуляк М. В., Галан В. Ю., Іпіроті В. О., Николин І. П. Системи підтримки прийняття рішень в медичній діагностиці. Інтелектуальні комп'ютерні системи та мережі : тези доп. V Наук.-практ. конф. молодих вчених і студентів (2 груд. 2021 р.). Тернопіль : ЗУНУ, 2021. С. 12.

Кваліфікаційна робота складається з вступу, трьох розділів, вписку використаних джерел, додатків [10].

1 АНАЛІЗ ІМУНОГІСТОХІМІЧНИХ ЗОБРАЖЕНЬ ТА СИСТЕМ ЇХ ЗБЕРІГАННЯ

1.1 Загальна характеристика програмної системи ImageJ

ImageJ – це програмна система з відкритим вихідним кодом для опрацювання зображень. Програмна система написана на мові Java співробітниками National Institutes of Health і поширюється без ліцензійних обмежень як суспільне надбання. Відкритий API дозволяє гнучко нарощувати функціонал за рахунок плагінів, що підключаються, а вбудована макромова - автоматизує складні дії, що повторюються [11-12]. ImageJ широко застосовується в біомедичних дослідженнях, астрономії, географії та інших дисциплінах, пов'язаних з аналізом зображень, в якості альтернативи пропрієтарного ПЗ.

Плагіни сторонніх розробників охоплюють широке коло завдань аналізу і обробки зображень: дозволяють проводити тривимірну візуалізацію в діапазоні від клітин до рентгенологічних зображень [13], автоматичні порівняння [14] аж до створення автоматизованих систем вивчення, наприклад, в гематології [15]. Архітектура плагінів ImageJ і вбудована в програму система розробки робить цю платформу достатньо популярною для роботи і викладання аналізу та обробки зображень [16-22].

ImageJ може працювати як онлайн аплет. Використання програми можливе у всіх операційних системах для яких існує Java Virtual Machine версії 1.4 і вище: Microsoft Windows, Mac OS, Mac OS X, Linux і Sharp Zaurus PDA. Вихідний код ImageJ також знаходиться у вільному доступі [12-13].

ImageJ дозволяє відображати, редагувати, аналізувати, обробляти , зберігати і друкувати 8 -бітові, 16 - бітові та 32 -бітові зображення. Програма може читати багато форматів зображень, такі як TIFF, PNG, GIF, JPEG, BMP, DICOM, FITS , а також raw формати. ImageJ підтримує стеки - серії зображень ,

які об'єднані в одному вікні, а багатопотокові трудомісткі операції можуть виконуватися на багатопроцесорних системах в паралельному режимі. У ImageJ можна обчислювати площі, статистичні показники піксельних значень різних виділених областей інтересу на зображеннях, які виділені вручну або за допомогою порогових функцій. Програма може вимірювати відстані і кути. Вона може створювати гістограми щільності і малювати профілі ліній. ImageJ підтримує стандартні функції обробки зображень , такі як логічні і арифметичні операції між зображеннями, маніпуляції з контрастністю, згортки, Фур'є - аналіз, підвищення різкості, згладжування, виявлення меж і медіанний фільтр. Програма дозволяє проводити різні геометричні перетворення, такі як масштабування, поворот або дзеркальне відображення. Програма підтримує будь-яку кількість одночасно використовуваних зображень, обмеження пов'язане тільки з обсягом доступної пам'яті.

ImageJ дозволяє відображати, редагувати, аналізувати, обробляти, зберігати і друкувати 8-бітові , 16-бітові та 32-бітові зображення. Програма може читати багато форматів зображень, таких як TIFF, PNG, GIF, JPEG, BMP, DICOM, FITS, а також raw формати. ImageJ підтримує стеки - серії зображень, які об'єднані в одному вікні, а багатопотокові трудомісткі операції можуть виконуватися на багатопроцесорних системах в паралельному режимі. У ImageJ можна обчислювати площі, статистичні показники піксельних значень різних виділених областей інтересу на зображеннях, які виділені вручну або за допомогою порогових функцій. Програма може вимірювати відстані і кути. Вона може створювати гістограми щільності і малювати профілі ліній. ImageJ підтримує стандартні функції обробки зображень, такі як логічні і арифметичні операції між зображеннями, маніпуляції з контрастністю, згортки, Фур'є - аналіз, підвищення різкості, згладжування, виявлення меж і медіанний фільтр. Програма дозволяє проводити різні геометричні перетворення, такі як масштабування, поворот або віддзеркалення. Програма підтримує будь-яку кількість одночасно використовуваних зображень, обмеження пов'язане тільки з об'ємом доступної

пам'яті.

Ідеолог і розробник проекту - Wayne Rasband (Research Services Branch of the National Institute of Mental Health).

1.2 Структура класів ImageJ

В ImageJ є такі класи пакету ij:

– ImageJApplet – ImageJ може виконуватись як аплет або як програма. Це клас аплета ImageJ. Перевага у запуску ImageJ як аплета полягає в тому, що програму можна запускати у браузері, найбільші недоліки це обмежений доступ до файлів на диску через політику безпеки аплету;

– ImageJ - головний клас програми ImageJ. Цей клас містить метод з якого програма починає своє виконання, тобто є головною точкою входу програми і головним вікном ImageJ;

– Executer - клас для виконання меню команд у окремих потоках (не блокуючи інші частини програми);

– IJ - клас, що містить багато сервісних методів;

– ImagePlus - представлення зображення в ImageJ, яке заснований на ImageProcessor;

– ImageStack - масив зображень.

– Нижче наведено класи пакету ij.gui:

– ProgressBar — показник прогресу виконання операції в головному вікні ImageJ;

– GenericDialog - модальний діалог, який можна налаштувати і застосувати на льоту, наприклад, для отримання введених користувачем даних;

– NewImage - клас для створення нового зображення певного типу з нуля;

– Roi - клас, що представляє область інтересу на зображенні. Якщо ця

область підтримується плагіном то буде оброблена лише ця область, а не все зображення.

Пакет ij.io містить класи для читання/декодування і запису/кодування файлів зображень.

Нижче наведено класи пакету ij.plugin:

– PlugIn – це інтерфейс, що повинен реалізовуватись плагінами, що не потребують зображення в якості вхідного параметру;

– PlugInFilter – це інтерфейс, що повинен реалізовуватись плагінами, що потребують зображення в якості вхідного параметру;

– PlugInFrame – це клас вікна, що може бути підкласом плагіна.

– Нижче наведено класи пакету ij.process:

– ImageConverter – це клас, що містить методи для конвертування зображень з одного типу у інший;

– ImageProcessor – це абстрактний суперклас для різних типів зображень. Він надає методи для роботи із зображенням;

– StackConverter – це клас для конвертування стеків з одного типу зображення в інший;

– StackProcessor – це клас для опрацювання стеку зображень.

1.3 Структура плагінів ImageJ

Функції, що надаються командами меню ImageJ можуть бути розширені користувацькими плагінами. Ці плагіни – це java класи, що реалізують необхідні інтерфейси, які поміщені в певну директорію. Вони можуть бути зручно зібрані в ImageJ або виконуватись з командного файлу в навколишньому середовищі Windows. Плагіни, які знайдені програмою ImageJ поміщаються в меню Plugins [13].

Існує в основному два типи плагінів: ті, які не вимагають вхідного зображення (реалізують інтерфейс PlugIn) і ті, які вимагають вхідного зображення (реалізують інтерфейс PlugIn Filter).

Інтерфейси. Інтерфейс PlugIn має лише один метод:

void run(java.lang.String arg)

Цей метод запускає плагін, те що в ньому реалізовано і буде роботою плагіна. Arg – це стрічка, що передана як аргумент, також може бути пустою стрічкою. Існує можливість установити плагіни більше одного разу, так що кожен з них буде викликати цей же плагін, але з іншим аргументом.

Інтерфейс PlugInFilter має метод: void run(ImageProcessor ip).

Перший метод запускає плагін, те що в ньому реалізовано і буде роботою плагіна. Він отримує зображення в якості вхідного аргументу яке може бути модифікувано прямо або може бути створена його копія для того, щоб оригінальне зображення залишилось без змін. Оригінальне зображення буде заблоковано поки плагін не завершить роботу. На відміну від методу run інтерфейсу PlugIn даний метод не отримує стрічкового аргументу, проте стрічковий аргумент може бути переданий у метод: int setup(java.lang.String arg, ImagePlus imp)

Даний метод налаштовує плагін для використання. Стрічка arg виконує цю саму функцію, що і метод run інтерфейсу PlugIn. Аргумент img обробляється програмою ImageJ і в даний момент передає поточне активне зображення. Метод повертає прапорці, які представляють можливості плагіна (наприклад, який тип зображення може бути обробленим). Значення прапорців містяться у класі PlugInFilter і вони наступні:

– static int DOES_16 — плагін може обробити 16-бітне зображення градації сірого;

– static int DOES_32 — плагін може обробити 32-бітне зображення градації сірого з плаваючою точкою;

– static int DOES_8C — плагін може обробити 8-бітне кольорове

зображення;

– static int DOES_8G — плагін може обробити 8-бітне зображення градації сірого;

– static int DOES_ALL — плагін може обробити усі типи зображень;

– static int DOES_RGB — плагін може обробити RGB зображення;

– static int DOES_STACKS — плагін може обробити стек зображень. ImageJ буде викликати метод для всіх зображень у стеку;

– static int DONE — якщо метод setup поверне це значення, то метод run не буде виконаним;

– static int NO_CHANGES — плагін не вносить змін у пікселі зображення;

– static int NO_IMAGE_REQUIRED — плагін не вимагає зображення;

– static int NO_UNDO — плагін не потребує операції відновлення (undo);

– static int ROI_REQUIRED — плагін вимагає область інтересу (ROI);

– static int STACK_REQUIRED — плагін вимагає стек зображень;

– static int SUPPORTS_MASKING — плагін завжди працює над обмежувальним прямокутником (ROI). Якщо цей прапорець буде встановлений і немає прямокутного ROI, то ImageJ відновить пікселі, які є в обмежувальному прямокутнику, але поза ROI.

Директорія для плагінів. Користувацькі плагіни для програми ImageJ повинні бути поміщені у директорію plugins, яка є піддерикторією програми ImageJ. Файли класів з'являться у меню лише, якщо назва класу містить хоча б один символ підкреслення.

Для компіляції і запуску плагінів можна використати команду “Compile and run” в меню плагінів. Для компіляції плагінів з командного рядка у середовищі Windows потрібно відредагувати файл “compile.bat”, який розташований у дерикторії plugins. Файл виглядає наступним чином:

```
rem Компілює усі плагіни у цій директорії і після запускає ImageJ
set PATH=c:\jdk1.3\bin;%PATH%
set CLASSPATH=../ij.jar;.
```

javac *.java

java ij.ImageJ

У директорії plugins можна знайти зразки плагінів, яку будуть розглянуті нижче.

Inverter_ це плагін, що інвертує 8-бітне зображення градації сірого. Нижче проводиться імпорт потрібних пакетів, ij.* для базових класів ImageJ, ij.process.* для оброблювачів зображень та інтерфейс ij.plugin.filter.PlugInFilter для реалізації його плагіном:

```
import ij.*;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;
import java.awt.*.
```

Потрібно зазначити, що не можна використовувати оператор пакету всередині класів, вони повинні бути в пакеті за замовчуванням. Плагіну додано необхідне підкреслення. Плагін потребує вхідного зображення, тому він повинен реалізувати інтерфейс PlugInFilter:

```
public class Inverter_ implements PlugInFilter {
```

Дальше іде метод налаштування класу. В даному випадку при значенні аргументу arg “about” буде викликаний метод showAbout, який виводить на екран діалогове вікно. Після чого повертається DONE, щоб метод run не був викликаний. У будь-якому іншому випадку повертаються прапорці, що означають наступне: плагін може опрацьовувати 8-бітні зображення градації сірого, також може опрацьовувати стек зображень і у випадку, якщо визначена область інтересу (ROI) то то буде оброблятися лише ця область:

```
public int setup(String arg, ImagePlus imp) {
        if (arg.equals("about"))
        {
                showAbout();
                return DONE;
```

```
        }
        return DOES_8G+DOES_STACKS+SUPPORTS_MASKING;
    }
```

Метод run реалізує основну функцію цього плагіна. Він отримує оброблювач зображення з оригінального зображення. Після чого отримується масив пікселів, так ніби це 8-бітне зображення градації сірого (містить 256 можливих значень). Цей масив є одновимірним, що містить один рядок пікселів за другим. Дальше отримується ширина цього зображення, а також прямокутник області інтересу (ROI):

```
    public void run(ImageProcessor ip) {
        byte[] pixels = (byte[])ip.getPixels();
        int width = ip.getWidth();
        Rectangle r = ip.getRoi();
```

Тепер оголошуються дві змінні для уникання обчислення позиції в одновимірному масиві для кожної ітерації. У зовнішньому циклі іде прохід від першого рядка ROI до його останнього рядка. Далі обчислюється зміщення (позиція першого пікселя поточного рядка) і іде перехід у внутрішній цикл від самого лівого пікселю ROI до самого правого пікселю. Після чого іде присвоєння поточному пікселю його інвертованого значення шляхом віднімання його поточного значення від 255:

```
    int offset, i;
    for (int y=r.y; y<(r.y+r.height); y++) {
        offset = y*width;
        for (int x=r.x; x<(r.x+r.width); x++) {
            i = offset + x;
            pixels[i] = (byte)(255-pixels[i]);
        }
    }
```

showAbout використовує статичний метод showMessage з класу IJ для відображення тексту у вікні із повідомленням. Перший параметр визначає його заголовок, а другий його текстове повідомлення:

```
void showAbout() {
IJ.showMessage("About Inverter_...",
        "This sample plugin filter inverts 8-bit images. Look\n" +
        "at the 'Inverter_.java' source file to see how easy it is\n" +
        "in ImageJ to process non-rectangular ROIs, to process\n" +
        "all the slices in a stack, and to display an About box."
);
}
```

Компіляція і запуск плагінів. Після написання демонстраційного плагіна його потрібно скомпілювати і запустити. У середовищі Windows можливо викликати пакетний файл compile.bat в директорії плагінів. Це скомпілює всі вихідні файли Java в цій директорії і запустить ImageJ. Скомпільовані плагіни (ці, що містять хоча б один символ підкреслення) стануть доступними в меню Plugins.

Також є можливість скомпілювати файл та запустити плагін за допомогою команди “Compile and run...”, яка міститься у меню плагінів.

Так само як і команди, плагіни можуть бути доступні через гарячі клавіші. Існує можливість для створення нових гарячих клавіш за допомогою вибору команди “Create hot-key” з меню "Plugins/ Hot-Keys". Також є можливість задати значення аргумента із істрічкою.

Для видалення гарячої клавіші плагіна потрібно натиснути “Plugins/Hot-Keys/Remove” з меню плагінів.

Представлення зображення в програмі ImageJ. Вище, коли розглядався демонстраційний плагін, можна було побачити, що зображення представлені об'єктами ImagePlus і ImageProcessor в ImageJ. Тут будуть більш детальніше

розглянуті операції над зображеннями, що обробляються програмою ImageJ.

Типи зображень.Зображення — це великі масиви піксельних значень. Але важливо знати, як ці піксельні значення будуть інтерпретовані. Це визначається типом зображення. ImageJ має п'ять типів зображень:

– 8-бітне зображення градації сірого: можуть відобразити 256 градацій, піксель представляється типом byte;

– 8-бітне кольорове зображення: може відобразити 256 кольорів, які визначені у таблиці LUT, піксель представляється типом byte;

– 16-бітне зображення градації сірого: може відобразити 65536 градацій, піксель представляється типом short;

– RGB кольорове зображення: може відобразити 256 значення на канал, піксель представляється типом int;

– 32-бітне зображення: зображення градації сірого із плаваючою точкою, піксель представляється типом float.

ImagePlus - об'єкт, який представляє зображення. Він базується на класі ImageProcessor, який містить піксельний масив і виконує фактично всю роботу над зображенням. Тип ImageProcessor'a, що використовується залежить від типу зображення. Типи зображення представлені константами в ImagePlus:

– COLOR_256 — 8-бітне зображення з LUT;

– COLOR_RGB — RGB кольорове зображення;

– GRAY16 — 16-бітне зображення градації сірого;

– GRAY32 — 32-бітне зображення градації сірого з плаваючою точкою;

– GRAY8 — 8-бітне зображення градації сірого;

ImageJ відображає зображення за допомогою класу ImageWindow. Цей клас виконує перефарбування, масштабування, зміну маски і тд. Для створення об’єкту ImagePlus необхідно використати один з наступних конструкторів:

– ImagePlus() - це конструктор по замовчуванням, створює новий пустий неініціалізований об’єкт;

– ImagePlus(java.lang.String urlString) - створює об’єкт і завантажує

зображення по вказаному URL;

– ImagePlus(java.lang.String title, java.awt.Image img) - створює новий об’єкт на базі об’єкту java.awt.Image. Перший аргумент є заголовком вікна;

– ImagePlus(java.lang.String title, ImageProcessor ip) - створює новий об’єкт з використанням об’єкту ImageProcessor. Перший аргумент є заголовком вікна;

– ImagePlus(java.lang.String title, ImageStack stack) - створює новий об’єкт з об’єкту ImageStack. Перший аргумент є заголовком вікна.

– Тип зображення може бути отриманим за допомогою наступного методу:

– int getType()

Подібні методи існують для розміру зображення, заголовку та URL адреси зображення:

int getHeight()

int getWidth()

java.lang.String getTitle()

java.net.URL getURL()

Об’єкт ImagePlus базується на java.at.Image, тому зображення може бути встановленим подібно до заголовку:

void setImage(java.awt.Image img)

void setTitle(java.lang.String title)

ImageProcessor. Кожне зображення в ImageJ представляється абстрактним класом ImageProcessor. Його тип визначається класом ImageType. Встановити та отримати об’єкт ImageProcessor можна за допомогою двох методів класу ImagePlus:

ImageProcessor getProcessor()

void setProcessor(java.lang.String title, ImageProcessor ip)

При роботі із плагіном PlugInFilter не потрібно отримувати ImageProcessor з ImagePlus, оскільки він передається у метод run як вхідний аргумент.

Оскільки ImageProcessor — це абстрактний клас, то конкретний тип зображення визначається його підкласами. Існує п’ять таких підкласів:

– ByteProcessor - використовується для 8-бітних зображень градації сірого та кольорових зображень. Він має підклас, який називається BinaryProcessor для зображень градації сірого, які містять пікселі із значенням від 0 до 255;

– ShortProcessor - використовується для 16-бітних зображень градації сірого;

– ColorProcessor - використовується для 32-бітних RGB зображень, де надається 8 біт на канал;

– FloatProcessor - використовується для 32-бітних зображень із плаваючою точкою.

Доступ до значень пік селів. Для роботи із зображенням необхідно отримати доступ до його пікселів. Вище було наведено як отримати доступ до ImageProcessor, після чого можна отримати доступ до масиву його пікселів за допомогою наступного методу:

java.lang.Object getPixels() . Вищенаведений метод повертає посилання на масив пікселів об’єкту ImageProcessor. Тип цього масиву залежить від типу зображення, тому необхідно привести тип отриманого масиву до відповідного типу зображення:

```
ColorProcessor myProcessor = myImage.getProcessor();
int[] pixels = (int[]) myProcessor.getPixels();
```

Вищенаведений приклад буде працювати для RGB зображень. Як уже було сказано, масив є одновимірним і містить рядок пікселів за рядком. Для перетворення доступу до масиву, що була можливість отримувати значення пікселів по координатам (x, y) потрібно знати хоча б ширину зображення. Ширина і висота об’єкту ImageProcessor може бути отримана за допомогою наступних методів:

```
int getHeight()
```

int getWidth()

Це все, що потрібно знати для проходу по масиву пікселів. Як це уже було показано в демонстраційному плагіні, де використовувались два вкладені цикли. Нижче буде розглянути 2 випадки для отримання значення пікселів для ByteProcessor та ColorProcessor.

У Java тип даних byte є знаковим і може мати значення від -128 до 127, проте очікувався діапазон значення від 0 до 255. Тому при приведені типу byte до іншого типу, потрібно переконатися, що біт знаку не враховується. Це може бути зроблено наступним чином з використанням бінарної операції AND:

```
int pix = 0xff & pixels[i];
...
pixels[i] = (byte) (pix & 0xff);
```

ColorProcessor повертає масив пікселів типу int (int[]). Значення кольору трьох компонентів запаковано у тип int. Ці значення можуть бути отримані наступним чином:

```
int red = (int)(pixels[i] & 0xff0000)>>16;
int green = (int)(pixels[i] & 0x00ff00)>>8;
int blue = (int)(pixels[i] & 0x0000ff);
...
pixels[i] = ((red & 0xff) << 16) + ((green & 0xff) << 8) + (blue & 0xff);
```

Оскільки масив пікселів над яким вище проводились операції є посиланням на масив пікселів об’єкту ImageProcessor, тому усі внесені зміни у цей масив будуть відразу задіяні для зображення. Проте, якщо потрібно, щоб ImageProcessor використовував інший масив (можливо новостворений), то масив пікселів можна встановити наступним чином:

void setPixels(java.lang.Object pixels)

Також не потрібно завжди отримувати доступ до всього масиву пікселів. ImageProcessor надає деякі інші методи для отримання та встановлення значеня

пікселів:

– int getPixel(int x, int y) - повертає значення вказаного пікселя;

– void putPixel(int x, int y, int value) - встановлює значення для вказаного пікселя;

– float getPixelValue(int x, int y) - повертає значення вказаного пікселя;

– void getColumn(int x, int y, int[] data, int length) — повертає значення пікселів вниз по колонці починаючи з (x, y);

– void putColumn(int x, int y, int[] data, int length) - встановлює значення пікселів у колонку починаючи з (x, y);

– void getRow(int x, int y, int[] data, int length) - повертає значення пікселів по горизонталі починаючи з (x, y);

– void putRow(int x, int y, int[] data, int length) - встановлює значення пікселів по горизонталі починаючи з (x, y);

– double[] getLine(int x1, int y1, int x2, int y2) - повертає значення пікселів по горизонталі (x1,y1)/(x2,y2);

– int[] getPixel(int x, int y) — повертає значення пікселя з (x, y), як 4-вимірний масив;

Усі ці методи повинні бути використані, якщо існує намір для модифікування лише декількох пікселів. Якщо потрібно модифікувати великі частини зображення, то для збільшення продуктивності необхідно працювати напряму з піксельним масивом.

Області інтересу. PlugInFilter не обов’язково повинен завжди опрацьовувати повністю все зображення. ImageJ підтримує наступні області інтересу (ROI):

– прямокутні;

– круглі;

– полігональні;

– довільної форми.

Поточний обмежувальний прямокутник область інтересу може бути

отриманим з об’єкту ImageProcessor за допомогою наступного метода:

java.awt.Rectangle getRoi()

Метод дає можливість обробити лише пікселі, що входять у обмежувальний прямокутник. Встановити обмежувальний прямокутник дає можливість наступний метод:

void setRoi(int x, int y, int rwidth, int rheight)

Метод встановлює обмежувальний прямокутник починаючи з (x, y) із заданою шириною та висотою.

Більше методів для роботи з ROI можна знайти у класі ImagePlus. PlugInFilter у метод run передає лише ImageProcessor, але є можливість отримати доступ до ImagePlus у методі setup. Нижче наведені ці методи:

void setRoi(int x, int y, int width, int height)

void setRoi(java.awt.Rectangle r)

void setRoi(Roi roi)

Roi getRoi()

Створення нових зображень. У багатьох випадках є сенс, щоб плагін не модифікував оригінальне зображення, а створював нове зображення та усі зміни вносив у нього.

Нижче наведений метод об’єкта ImagePlus:

ImagePlus createImagePlus()

Цей метод повертає новий об’єкт ImagePlus, із збереженням атрибутів, але без зображення. Подібний метод існує у об’єкта ImageProcessor:

ImageProcessor createProcessor(int width, int height)

Метод повертає новий пустий ImageProcessor із збереженням ширини та висоти, який може бути використаний для створення нового ImagePlus за допомогою наступного конструктору:

ImagePlus(java.lang.String title, ImageProcessor ip)

Клас NewImage надає деякі корисні статичні методи для створення нового ImagePlus певного типу. Ці статичні методи наведені нижче:

static ImagePlus createByteImage(java.lang.String title,

int width, int height,

int slices, int fill).

Метод створює нове 8-бітне зображення градації сірого або кольорове зображення з вказаним заголовком, шириною та висотою, а також із вказаною кількість частин. Аргумент fill містить значення константи, яка визначає як зображення буде ініціалізовано. Можливі значення будуть наведені нижче.

static ImagePlus createFloatImage(java.lang.String title,

int width, int height,

int slices, int fill)

Метод створює нове 32-бітне зображення з плаваючою точкою з вказаним заголовком, шириною та висотою, а також із вказаною кількість частин. Аргумент fill містить значення константи, яка визначає як зображення буде ініціалізовано.

static ImagePlus createRGBImage(java.lang.String title,

int width, int height,

int slices, int fill)

Метод створює нове RGN зображення з вказаним заголовком, шириною та висотою, а також із вказаною кількість частин. Аргумент fill містить значення константи, яка визначає як зображення буде ініціалізовано.

static ImagePlus createShortImage(java.lang.String title,

int width, int height,

int slices, int fill)

Метод створює нове 16-бітне зображення градації сірого з вказаним заголовком, шириною та висотою, а також із вказаною кількість частин. Аргумент fill містить значення константи, яка визначає як зображення буде ініціалізовано.

Нижче наведені константи аргументу fill, ці константи визначені у класі NewImage:

– FILL_BLACK - заповнює зображення чорним кольором;

– FILL_WHITE - заповнює зображення білим кольором;

– FILL_RAMP - заповнює зображення з градацією сірого по горизонталі.

Існує два методи для копіювання значення пікселів між двома різними ImageProcessor:

void copyBits(ImageProcessor ip, int xloc, int yloc, int mode)

Метод копіює зображення, яке представлене як ImageProcessor по xcol, ycol і використовує вказаний bliting mode. Список цих констант визначено у інтерфейсі Blitter:

– AND - destination = destination AND source ;

– AVERAGE - destination = (destination+source)/2 ;

– COPY - destination = source ;

– COPY_INVERTED - destination = 255-source ;

– COPY_TRANSPARENT — білі пікселі вважаються прозорими;

– DIFFERENCE - destination = |destination-source| ;

– DIVIDE - destination = destination/source ;

– MAX - destination = maximum(destination,source) ;

– MIN - destination = minimum(destination,source) ;

– MULTIPLY - destination = destination*source ;

– OR - destination = destination OR source ;

– SUBTRACT - destination = destination-source ;

– XOR - destination = destination XOR source.

Наступний метод вставляє зображення ImageProcessor у (xloc, yloc):

void insert(ImageProcessor ip, int xloc, int yloc)

Якщо немає необхідності у створені об’єкту ImagePlus для використання у ImagePlus, а потрібне зображення типу java.awt.Image то існує можливість отримати його за допомогою об’єкту ImageProcessor або ImagePlus:

java.awt.Image createImage()

java.awt.Image getImage()

Відображення зображень. Після модифікування зображень необхідно знати як зміни можуть бути відображені. ImageJ використовує клас, що називається ImageWindow для відображення зображень, що представляються класом ImagePlus. ImagePlus містить все, що є необхідним для обновлення або відображення новостворених зображень.

Ці методи наведені нижче:

– void draw() - відображає це зображення;

– void draw(int x, int y, int width, int height) - перерисовує пікселі зображення у заданій області;

– void updateAndDraw() - обновляє зображення з масиву пікселів, що знаходиться у ImageProcessor, після чого відображає його;

– void updateAndRepaintWindow() - викликає метод updateAndDraw і після нього також перерисовує вікно, щоб оновити інформацію про зображення (розміри, тип);

– void show() - відкриває вікно для відображення цього зображення і очищає область статусу;

– void show(java.lang.String statusMessage) - відкриває вікно для відображення цього зображення і показує повідомлення із станом у області статусу;

– void hide() - закриває вікно, що відображає це зображення.

Стеки зображень. ImageJ підтримує групу зображення, що є масивом зображені і називається стеком зображень, усі ці зображення однакового розміру. В PlugInFilter отримати доступ до поточного відкритого стеку зображень можна за допомогою поточного ImagePlus використовуючи метод:

ImageStack getStack()

ImagePlus також містить метод для створення нового стеку. Цей метод повертає пустий стек, що має такий же розмір і колір як і зображення на якому був викликаний цей метод:

ImageStack createEmptyStack()

Також є можливість створити ImageStack зображення використовуючи один з цих конструкторів:

– ImageStack(int width, int height) - створює новий пустий стек зображень із вказаною шириною та висотою;

– ImageStack(int width, int height, java.awt.image.ColorModel cm) — створює новий пустий стек зображень із вказаною шириною, висотою та колірною моделлю;

Щоб встановити новостворений стек зображень у поточне зображення можна використати наступний метод:

void setStack(java.lang.String title, ImageStack stack)

Кількість зображень у стеку може бути отримано за допомогою методу getSize над класом ImageStack або за допомогою методу getStackSize над класом ImagePlus:

int getSize()

int getStackSize()

Поточне зображення що відображається у ImagePlus може бути отримано та встановлено за допомогою наступних методів:

int getCurrentSlice()

void setSlice(int index)

Стек зображень містить наступні методи для отримання та встановлення своїх властивостей:

– int getHeight() - повертає висоту стеку;

– int getWidth() - повертає ширину стеку;

– java.lang.Object getPixels(int n) — повертає масив пікселів для вказаного зображення у стеку, де n це число від 1 до кількості зображень у стеку;

– void setPixels(java.lang.Object pixels, int n) — встановлює масив пікселів для вказаного зображення у стеку, де n це число від 1 до кількості зображень у стеку;

– ImageProcessor getProcessor(int n) — повертає ImageProcessor для вказаного зображення у стеку, де n це число від 1 до кількості зображень у стеку;

– java.lang.String getSliceLabel(int n) — повертає мітку для вказаного зображення у стеку, де n це число від 1 до кількості зображень у стеку;

– void setSliceLabel(java.lang.String label, int n) — встановлює мітку для вказаного зображення, де n це число від 1 до кількості зображень у стеку;

– java.awt.Rectangle getRoi() - повертає прямокутне ROI для стеку зображень;

– void setRoi(java.awt.Rectangle roi) — встановлює прямокутне ROI для стеку зображень;

Зображення можуть бути додані і видалені зі стеку за допомогою наступних методів:

– void addSlice(java.lang.String sliceLabel, ImageProcessor ip) — додає зображення, яке представляється типом ImageProcessor у кінець стеку зображень;

– void addSlice(java.lang.String sliceLabel, ImageProcessor ip, int n) - додає зображення, яке представляється типом ImageProcessor у вказану позицію стеку зображень;

– void addSlice(java.lang.String sliceLabel, java.lang.Object pixels) — додає зображення, яке представлене масивом пікселів у кінець стеку зображень;

– void deleteLastSlice() - видаляє останнє зображення із стеку зображень;

– void deleteSlice(int n) — видаляє зображення із стеку зображень у вказаній позиції.

ImageProcessor.Нижче наведены методи для геометричного перетворення:

– void flipHorizontal() - дзеркальне перетворення зображення по горизонталі;

– void flipVertical() - дзеркальне перетворення зображення по вертикалі;

– void rotate(double angle) — повертає кут зображення за годинниковою

стрілкою;

– void scale(double xScale, double yScale) — масштабування зображення за вказаними факторами;

– ImageProcessor crop() - обрізає зображення до прямокутного ROI. Після чого повертає новий ImageProcessor, який представляє обрізане зображення;

– ImageProcessor resize(int dstWidth, int dstHeight) — змінює розмір зображення до заданого розміру. Після чого повертає новий ImageProcessor, який представляє зображення із новим розміром;

– ImageProcessor rotateLeft() - повертає зображення на кут 90 градусів проти годинникової стрілки. Після чого повертає новий ImageProcessor, який представляє змінене зображення;

– ImageProcessor rotateRight() 0 повертаэ зображення на кут 90 градусів за годиниковою стрылкою. Після чого повертає новий ImageProcessor, який представляє змінене зображення;

– void setInterpolate(boolean interpolate) — встановлює білінійну інтерполяцію для масштабування, зміни розміру та обертання.

Нижче наведено деякі методи для фільтрації:

– void convolve3×3(int[] kernel) - виконує згортку зображення з вказаним 3×3 масивом;

– void sharpen() - зрізає зображення з використанням ядра згортки 3×3;

– void smooth() - замінює кожен піксель середнім значенням сусідніх пікселів 3×3;

– void filter(int type) — виконує операцію фільтрування 3×3. Тип фільтру визначається вхідним аргументом;

– void dilate() - розширює зображення за допомогою 3×3 мінімального фільтру;

– void erode() - звужує зображення за допомогою 3×3 максимального фільтру;

– void findEdges() - знаходить краї з використанням оператору Собеля;

– void medianFilter() - виконує 3×3 медіанний фільтр;

– void gamma(double value) — виконує гамма корекцію;

– void invert() - виконує інвертування зображення;

– void add(int value) — додає вхідне значення до усіх пікселів зображення;

– void multiply(double value) — помножує усі пікселі зображення на вхідне значення;

– void and(int value) — виконує бінарну операцію AND над вхідним аргументом та кожним пікселем зображення;

– void or(int value) — виконує бінарну операцію OR над вхідним аргументом та кожним пікселем зображення;

– void xor(int value) – виконує бінарну операцію XOR над вхідним аргументом та кожним пікселем зображення;

– void log() – виконує обчислення значення пікселів в логарифмічному масштабі;

– void noise(double range) — додає випадковий шум (випадкове число вказаного діапазону) до зображення;

Нижче наведено методи для рисування:

– void setColor(java.awt.Color color) — встановлює колір переднього плану. Також цей колір буде використовуватись як колір за замовчуванням для операцій рисування та заливання;

– void setValue(double value) — встановлює колір за замовчуванням для операцій рисування та заливання;

– void setLineWidth(int width) — встановлює товщину лінії;

– void moveTo(int x, int y) — встановлює поточну точку рисування (x, y);

– void lineTo(int x2, int y2) — рисує лінію з поточної точки до точки (x2, y2);

– void drawPixel(int x, int y) — рисує піксель (x, y) поточним кольором для рисування;

– void drawDot(int xcenter, int ycenter) — рисує точку з використанням поточної товщини лінії та поточного кольору рисування;

– void drawDot2(int x, int y) — рисує точку розміром 2×2 з поточним кольором рисування;

– void fill()- заливає поточне квадратне ROI поточним кольором рисування;

– void fill(int[] mask) — заливає пікселі в межах поточного ROI і частину маски (пікселі, що мають значення 0, тобто чорні в масиві маски);

– void drawString(java.lang.String s) — рисує стрічку в поточні позиції з поточним кольором рисування;

– int getStringWidth(java.lang.String s) — повертає ширину вказаної стрічки в пікселях;

– int getBestIndex(java.awt.Color c) — повертає індекс LUT, що найбільш точно відповідає вказаному кольору;

– java.awt.image.ColorModel getColorModel() - повертає колірну модель;

– void invertLut() - інвертує кольори в таблиці LUT.

Нижче наведені методи для роботи із гістограмою:

– int[] getHistogram() - повертає гістограму зображення. Цей метод буде повертати гістограми яркості для RGB зображень і null для зображені із плавучою точкою;

– int getHistogramSize() - повертає розмір гістограми, який буде 256 для 8-бітних і RGB зображень і max-min+1 для 16-бітних зображень;

– Нижче наведені методи для роботи із знімками (Undo):

– void snapshot() - зберігає поточний стан ImageProcessor як знімок;

– java.lang.Object getPixelsCopy() - повертає посилання на знімок масиву пікселів цього зображення. Це масив пікселів перед останніми змінами;

– void reset() - відновлює ImageProcessor до стану, який був збережений у знімку;

API програми ImageJ містять клас, що називається IJ, який містить деякі

дуже корисні статичні методи.

Повідомлення. Дуже часто плагіну необхідно показувати повідомлення, які можуть бути як повідомленнями про помилки так і повідомленнями про будь-яку іншу інформацію. Для першого випадку можна використати наступні методи:

– static void error(java.lang.String msg) — відображає повідомлення в діалоговому вікні з назвою “Erorr”;

– static void showMessage(java.lang.String msg) — відображає повідомлення в діалоговому вікні з назвою “Message”;

– static void showMessage(java.lang.String title, java.lang.String msg) — відображає повідомлення в діалоговому вікні з вказаною назвою;

– static boolean showMessageWithCancel(java.lang.String title, java.lang.String msg) — На відміну від вищенаведених методів, які відображають повідомлення, яке користувач повинен прийняти, даний метод дозволяє користувачеві прийняти повідомлення або ні. Даний метод повертає “false”, якщо користувач натиснув кнопку “Cancel” та повертає “true”, якщо користувач натиснув “OK”.

Клас містить також декілька уже визначених методів із повідомленнями:

– static void noImage() – показує діалог з повідомленням “no images are open”;

– static void outOfMemory(java.lang.String name) — показує діалог з повідомленням “out of memory”;

– static boolean versionLessThan(java.lang.String version) — показує діалог з повідомленням про помилку, якщо задана версія вища за поточну і повертає результат.

Головне вікно ImageJ складається з наступних компонентів:

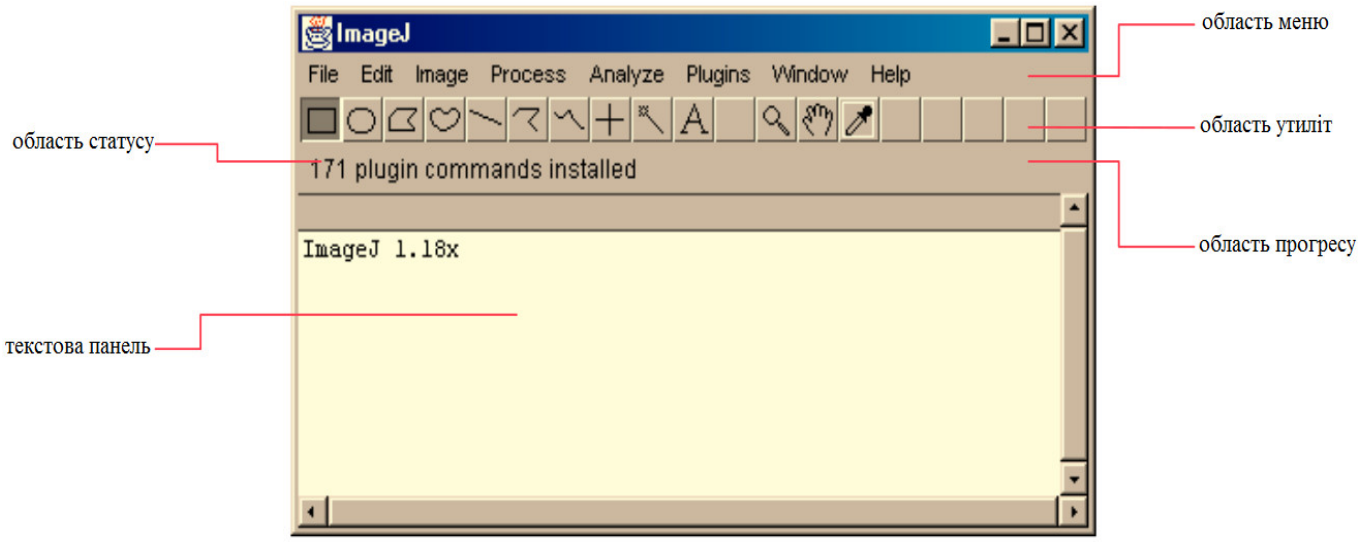


Рисунок 1.1 – Головне вікно ImageJ

Текстова панель. Для відображення стрічки тексту у цій панелі використовується наступний метод:

static void write(java.lang.String s)

Також існує можливість використовувати панель як таблицю (для відображення статистики, вимірювань і т. д.). В даному випадку ImageJ надає метод для встановлення назви стовпців, при виклиці цього методу весь текст буде очищено. Метод виглядає наступним чином:

static void setColumnHeadings(java.lang.String headings)

Наступний метод дозволяє записувати числа. Метод конвертує число у відформатовану стрічку, яка відображає 2 числа після коми:

static java.lang.String d2s(double n)

Наступний метод конвертує число у стрічку до заданої точності:

static java.lang.String d2s(double n, int precision)

Область статусу. Текст також може бути відображеним у меню статусу, яке знаходиться вище текстової панелі за допомогою наступного методу:

static void showStatus(java.lang.String s)

Також може бути корисно відобразити час, що був необхідний операції за допомогою наступного методу:

static void showTime(ImagePlus imp, long start, java.lang.String str)

Область прогресу. Прогрес поточної операції може бути відображеним за допомогою області прогресу вікна ImageJ. Наступний метод обновлює позицію області прогресу до вказаного значення (значення може бути в межах від 0.0 жо 1.0):

static void showProgress(double progress)

1.4 Аналіз імуногістохімічних зображень

Для проведення уточненого аналізу раку використовують біомаркери. Біомаркери взаємодіють із гістологічним зрізом тканини і зафарбовують певну кількість клітин. Кожен тип біомаркеру, взаємодіючи із зрізом тканини, зафарбовує певну кількість клітин. При цьому основними інформативними показниками реакції біомаркеру на клітини тканини є площа зафарбованих клітин та їх яскравість.

Обчислення підтипу раку молочної залози проводять на основі Нотінгемської методики (протоколу) [23-28].

В таблиці 1.1 приведені критерії та оцінка в балах кількості зафарбованих клітин і їх інтенсивність.

Таблиця 1.1 – Оцінки та критерії в балах кількості зафарбованих клітин і їх інтенсивність

Частка позитивних клітин (PS - proportion score)	
Оцінка	Критерії
0 балів	Позитивних клітин немає
1 бал	Частка позитивних клітин менше 1%
2 бали	Частка позитивних клітин від 1 % до 10%
3 бали	Частка позитивних клітин від 11% до 33%

4 бали	Частка позитивних клітин від 34% до 66%
5 бали	Частка позитивних клітин від 67% до 100%
Інтенсивність забарвлення клітин (IS - intensity score)	
1 бал	Слабке забарвлення
2 бал	Помірне забарвлення
3 бал	Сильне забарвлення, велика,значна
Інтерпретація результатів (TS - total score) = PS + IS	
0-2	Статус негативний
3-8	Статус позитивний

В таблиці 1.2 приведені підтипи раку молочної залози

Таблиця 1.2 – Підтипи раку молочної залози з ІГХ критеріями та гістологічною градацією

Молекулярно-генетичний підтип	ІГХ критерії	Гістологічна градація (G)
Люмінальна А	ER+ (>66% 3+); PR+ (>20%), HER2-, Кі-67<20%	G1, G2
Люмінальна В	ER+ (<66% 3+) або (1-2+), PR- (або <20%), HER2- або +, Кі-б7>20%	Частіше G2, G3
НЕР-2/ new -позитивна	HER2- +, ER/PR-	G3
Базальноподібна	ER/PR-, HER2-; більшість позитивні на базальні маркери, Кі-67 високий	G3

В таблиці 1.3 приведена інтерпретація результатів ІКХ дослідження біомаркера HER2/ neu.

Таблиця 1.3 – Інтерпретація результатів ІКХ дослідження біомаркера HER2/ neu

Керівництва та рекомендації ASCO/CAP 2013 щодо інтерпретації результатів ІГХ дослідження HER2/ neu	
Показник	Результат
0 (негативна)	Негативне або неповне слабке, ледь помітне мембранне забарвлення ≤10% клітин інвазивної пухлини
1+ (негативна)	Слабке, ледь помітне неповне мембранне забарвлення >10% клітин інвазивної пухлини
2+ (сумнівна)	Неповне та/або слабке/помірне мембранне забарвлення >10% клітин інвазивної пухлини або інтенсивне повне мембранне забарвлення ≤10% клітин інвазивної пухлини
3+ (позитивна)	Повне інтенсивне мембранне забарвлення >10% клітин інвазивної пухлини

Приклад реакції естрогену приведений на рисунку 1.2.

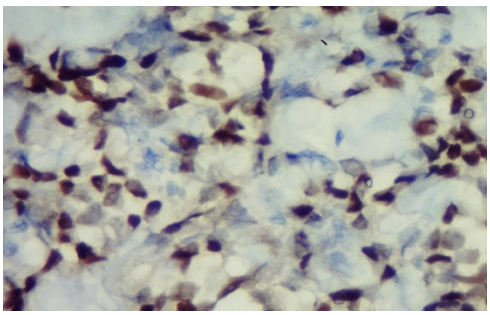


Рисунок 1.2 – Реакція естрогену

Естроген IS=2 PS=3 (11-33%) TS=5

Приклад реакції прогестерону приведений на рисунку 1.3.

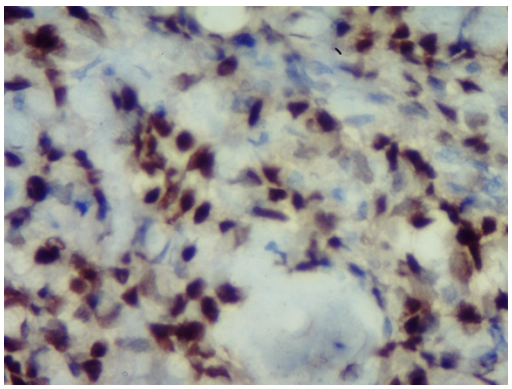


Рисунок 1.3 – Реакція прогестерону

Прогестерон IS=3 PS=4 (34-66%) TS=7

Приклад реакції Her2/neu negative приведений на рисунку 1.4.

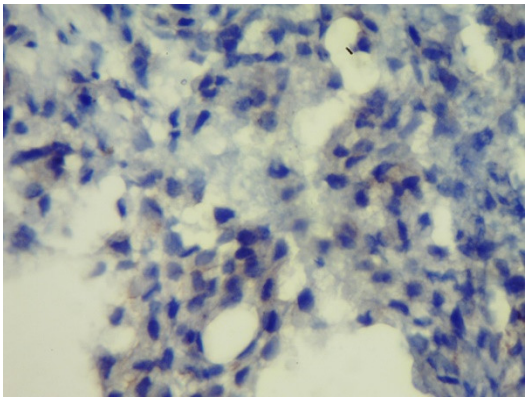


Рисунок 1.4 – Реакція Her2/neu negative

Приклад реакції Ki 67 приведений на рисунку 1.5.

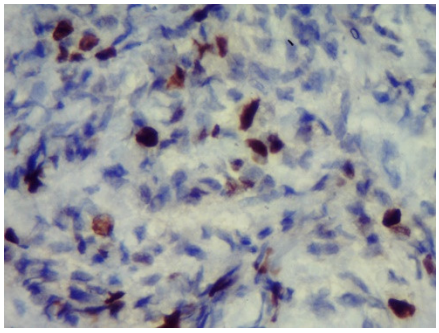


Рисунок 1.5 – Реакція Ki 67

1+

Діагноз: підтип люмінальний А

1.5 Висновки до розділу 1

Отже, в даному розділі проаналізовано програмну систему ImageJ, розглянуто, структура класів ImageJ, структуру плагінів ImageJ і здійснено аналіз імуногістохімічних зображень і біомаркера.

2 СТАТИСТИЧНЕ ОПРАЦЮВАННЯ ЗОБРАЖЕНЬ

2.1 Морфолого-статистичний аналіз

Планування кількісних морфологічних спостережень і експериментів складається з наступних етапів [29]:

1) визначається мета експерименту (спостережень);
2) виділяється перелік основних морфологічних ознак, що підлягають кількісній оцінці;
3) визначається обсяг вибірки, достатній для отримання достовірних даних;
4) встановлюється перелік необхідних морфометричних методик і ступінь точності дослідження. Остання визначається по величині помилки показника або на основі визначення достовірності різниць порівнюваних величин.

Підхід до розв’язання задач кількісної морфології достатньо складний, так як вимагає перетворення в числа всіх морфологічних ознак, що вивчаються. В медичній діагностиці використовується системно-класифікаційний підхід, який полягає в виділенні підсистем і елементів, визначенні тісноти зв’язку між ними, виявленні набору діагностичних ознак і відношень між ними.

Ознаки, що досліджуються, представляються описовими і квантованими шкалами, структурно-функціональними зв’язками, факторною структурою. При розв’язуванні діагностичних задач використовуються різні математичні підходи, в тому числі статистичні. Статистичний підхід до діагностичних задач включає знаходження законів розподілу, виду залежності, оцінку тісноти зв’язку між ознаками, встановлення «подібності – розбіжності» між ознаками і процесами.

Морфолого-статистичний аналіз включає в себе три великих і складних етапи. По-перше, слід створити математично обґрунтовану представницьку вибірку з великої кількості біологічних явищ, що проходять в організмі. По-

друге, отримати інформацію про вибрані для вивчення ознаки рахункового і мірного характеру і, по-третє, провести кваліфіковану статистичну обробку даних, отриманих в відповідності з метою дослідження представницьких вибірок.

Кількісне вираження ознаки називають величиною. Особливу увагу звертають на об'єднання матеріалу з урахуванням найважливіших ознак, що виражають суть явища, що вивчається.

Зведення абсолютних величин рахунку або міри в таблицях дозволяє отримати після відповідних обчислень: відносні показники, варіаційні ряди і узагальнені статистичні характеристики [30-31].

1. Відносні показники. Є відношенням двох чисел, що представляють порівняльні сукупності або їх частки. В якості основи беруть стандартні числа: 1, 100, 1000 і т. д. Ці показники можуть бути двох видів.

Відносні показники екстенсивності демонструють відношення частки до цілого (останнє зазвичай прирівнюють до 100%). Відношення часток дає уявлення про структуру явища, питому вагу складових його часток.

Відносні показники інтенсивності є показниками частоти появи ознаки. Ці показники отримують шляхом ділення числа появи ознаки на число всієї сукупності, що вивчається, де ця ознака може спостерігатися. Інтенсивну ознаку виражають в долях одиниці або, частіше, у відсотках або в проміле.

2. Варіаційні ряди. Рахунок і міра дають сукупності показників, які називаються варіантами, числовими значеннями ознаки, що змінюється. Варіанти характеризують відмінності в частотах появи події або ж зміни величини мірних ознак. Розкид вказаних значень ознак називається їх варіюванням.

Отримана сукупність показників, що варіюють називається варіаційним рядом (рядом розподілів). Цей ряд характеризує не лише кількісні особливості, але і якісні ознаки сукупності, що вивчається. Властивості ряду демонструються

або ранжуванням ознак по наростанню їх значень, або представленням накопичених частот за допомогою інтервальних рядів (з рівним числом варіант).

Варіаційні ряди представляють графічно у вигляді полігонів (для дискретних величин) або у вигляді гістограм (для неперервних величин). Графічне зображення варіаційного ряду – обов'язковий етап морфолого-статистичного аналізу, який візуалізує особливості процесу, що вивчається і дає можливість оцінити його якісні особливості.

При побудові варіаційного ряду, за даними морфометричного дослідження, межі груп встановлюють залежно від завдань роботи. Основу для цього етапу аналізу складають числові дані про кількісні (рахунок, міра) або якісні (атрибутивні) ознаки, які також перетворені в числа. Перший етап аналізу – визначення оптимального числа груп для вивчення явища. Для цієї мети зазвичай використовують спеціальну формулу

$$N = 1 + 3{,}32 \lg n,$$

за якою з урахуванням числа спостережень (n), що є у розпорядженні дослідника, встановлюють необхідну кількість груп, достатніх для проведення аналізу варіаційного ряду, що вивчається.

Число спостережень n також не може будь-яким. Приступаючи до морфологічного дослідження, необхідно мати в своєму розпорядженні репрезентативну (представницьку) вибірку. Точність результатів вибіркового спостереження залежить від способу відбору об’єктів, ступеня коливання досліджуваної ознаки в генеральній сукупності та від обсягу вибірки n.

Суцільне обстеження в медичній морфометрії, тобто обстеження генеральної сукупності, проводиться дуже рідко [32-33]. Воно є дуже трудомістким або, в більшості випадків, неможливим. В основному для досліджень вичерпну інформацію отримують з допомогою вибірок.

В медичній практиці вже встановилися приблизні обсяги вибірок: в антропометрії – 50-100 людей, в цитокаріометрії – 50-200 клітин і т.д. Але якщо досліджувані ознаки варіюють, тоді кінцеві результати статистичної обробки можуть бути зміненими, і необхідно зменшити вибірку або значно її збільшити. З точки зору наукових досліджень, величина вибірки повинна забезпечити довірчу ймовірність ознаки, яка вивчається, не менше 0,95 і граничну помилку не більше 0,05. В таблиці 2.1 приведено достатньо велике число одиниць спостережень в вибірках в залежності від довірчої ймовірності та точності оцінки.

Таблиця 2.1 – Достатній обсяг вибірки при заданій ймовірності і допустимій граничній помилці

Допустима гранична помилка Δ вибірки	Ймовірність γ			
	0,95	0,99	0,995	0,999
0,05	384	663	787	1082
0,04	600	1036	1231	1691
0,03	1067	1843	2188	3007
0,02	2400	4146	4924	6767
0,01	9603	16587	19699	27069

Ці дані важливі для ознак, що підраховуються (дискретних випадкових величин). Адже при морфометричному аналізі досить часто необхідно знати частку (долю) об’єктів генеральної сукупності (смертність, кількість клітин, ультраструктур певного типу і т. д.), що володіють деякою ознакою до загального числа елементів генеральної сукупності.

При заданій ймовірності 0,99, відомому коефіцієнту варіації та величині похибки обсяг вибірки мірних ознак можна визначити по номограмі або використати таблицю 2.2.

Важливо також при дослідженнях знати кількість основних об'єктів (хворих, тварин, органів) і кількість об'єктів, що вимірюються, для кожного з них (клітин, ядер, мітозів, судин і т.д.). При такому системному морфологічному дослідженні використовується наступний підхід до числа елементів вибірки. Відбираються випадково n_1 одиниць першого рівня (наприклад, органи). З кожної одиниці цього рівня відбирається одна і таж кількість n_2 одиниць другого рівня (наприклад, судини), з них в свою чергу одна і таж кількість n_3 одиниць третього рівня (наприклад, кількість зрізів, фотографій і т.д.). На кожній одиниці третього рівня, наприклад, робиться кількість n_4 однотипних вимірювань. Таким чином, при такому дослідженні використовуються n_1 органів, $n_1 \cdot n_2$ судин, $n_1 \cdot n_2 \cdot n_3$ зрізів і $n_1 \cdot n_2 \cdot n_3 \cdot n_4$ вимірів. Одиниці кожного рівня вибираються випадковим чином, що забезпечує репрезентативність вибірки і запобігає помилкам вибірки.

Таблиця 2.2 – Кількість одиниць спостережень в вибірках при вивченні мірних ознак

Значення коефіцієнта точності (k)	Ймовірність		
	0,95	0,98	0,99
	Критерій Ст'юдента t		
	2,0	2,5	3,0
Для робіт орієнтовного характеру			
0,5	16	25	36
Для робіт середньої точності			
0,4	25	39	56
0,3	45	70	100
0,2	100	156	225

Для робіт високої точності 0,1	400	625	900

Другий етап аналізу пов'язаний з групуванням отриманих результатів вимірів. Особливе місце в патоморфологічних дослідженнях займає типологічне групування, призначення якого полягає в розподілі сукупностей ознак на однорідні групи відповідно до їх основних типів.

Результати групувань по окремих морфологічних ознаках, що підлягають вивченню, зводять в статистичну таблицю, в якій головне групування, що відображає її зміст, отримало назву статистичного підмета.

Як правило, ці головні ознаки розміщують в горизонтальних рядках таблиці, а у вертикальних стовпцях (графах) записують дані, що характеризують і уточнюють головні ознаки. Ці уточнення отримали назву статистичного присудка.

При будь-якій іншій формі таблиці головна ознака завжди залишається статистичним підметом і має бути чітко позначена в заголовку таблиці, в якому обов'язково приводяться найменування одиниць виміру. Нульові значення ознаки позначають тире, за відсутності даних пишуть "немає відомостей".

Залежно від характеру дослідження і якості даних складають прості або складні таблиці (групові і комбінаційні). На початку дослідження таблиці можуть бути розробленими, а потім, по завершенню дослідження — остаточними (аналітичними), в яких вже приводяться результати математичного аналізу досліджуваного явища.

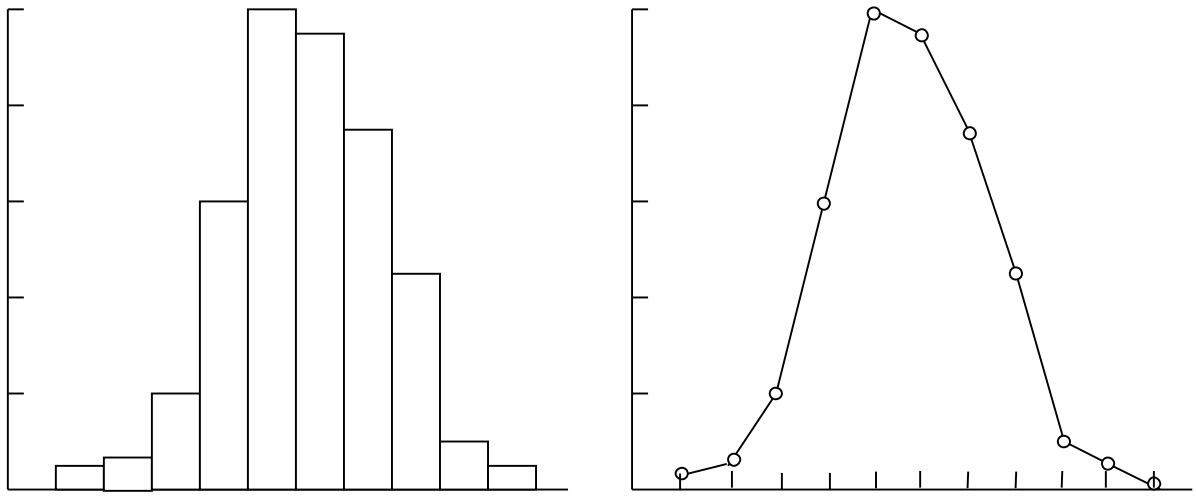
На завершальному етапі документації результатів спостережень складають статистичне зведення даних спостережень, в яке включають рахунковий і логічний контроль зібраного матеріалу, його шифровку (заміну знаками, числами, шифрами окремих значень ознак), розкладку документів обліку і їх підрахунок, заповнення макетів статистичних таблиць. При великому аналітичному матеріалі використовують комп'ютерні програми.

Таблиці є основним документом для всього подальшого аналізу, що має у ряді випадків і юридичне значення.

Далі, отримані в роботі дані, представляють графічно у вигляді гістограми, в якій висота стовпчиків характеризує розподіл варіант по частоті їх виявлення (рисунок 2.1, а).

Більш наочно явище, що вивчається, демонструється з допомогою варіаційної кривої, ординати якої пропорційні частотам варіаційного ряду (рисунок 2.1, б). При наростаючому накопиченні частот або збільшенні значень величин мірних ознак використовують кумуляту (рисунок 2.1, в).

В більшості випадків при проведенні морфометричного аналізу виникає так званий нормальний розподіл результатів (Гауса) [17], коли варіанти в основному розміщуються біля модального класу (рисунок 2.2). При дуже тісному розташуванні варіант біля середнього значення криву називають біноміальною, а при дуже згладженому – кривою розподілу рідкісних явищ (Пуасона). З урахуванням характеру отриманих кривих, що характеризують варіаційні ряди, проводять вибір адекватного статистичного методу їх аналізу для вирішення поставлених завдань.



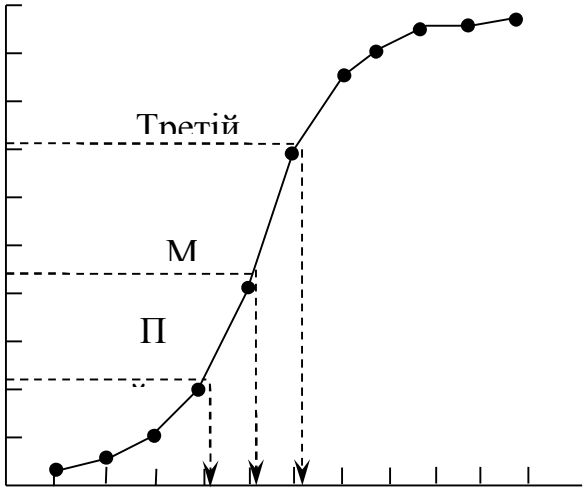


Рисунок 2.1 – Графічне зображення розподілу частоти морфологічної ознаки, що вивчається: а – гістограма; б – варіаційна крива; в – кумулята

Слід мати на увазі, що при нормальному розподілі морфометричних ознак, що вивчаються, інколи спостерігається асиметрія кривої розподілу варіант або поява ексцесів (параметрів, що вискакують). У цих випадках використовують спеціальні статистичні прийоми.

Найбільшого практичного поширення в морфометричних дослідженнях набули дві групи узагальнених характеристик варіацій ознак.

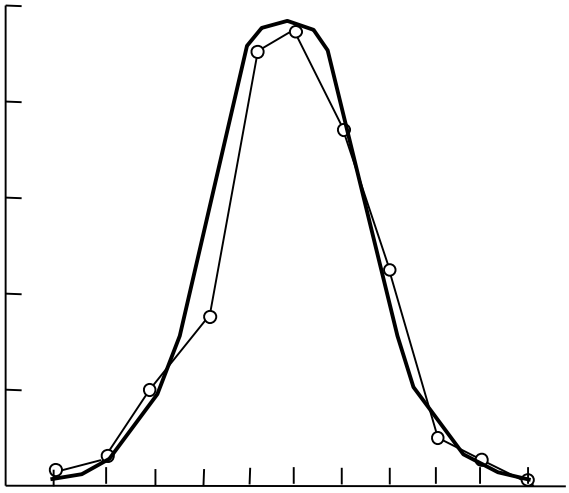


Рисунок 2.2 – Емпірична (а) і теоретична (б) варіаційні криві розподілу частоти морфологічної ознаки

2.2 Статистичні показники морфометричних ознак

До числа узагальнених показників входять: середня арифметична, мода, медіана і квартилі.

1. Проста середня арифметична величина (для генеральної сукупності М, для вибірки – $\bar{x}_в$) представляє найбільш загальну характеристику варіаційного ряду. Середню арифметичну просту ($\bar{x}_в$) отримують шляхом ділення суми значень всіх варіант на їх число (1.2).

Варіаційні ряди, що мають різну частоту появи окремих варіант, характеризуються зваженою середньою арифметичною, яку обчислюють за формулою (1.1). У цій середній величині враховується частота повтору (n_i) кожної варіанти, завдяки чому на середньо зваженій арифметичній позначається вплив частот появ окремих варіантів. Зважена середня арифметична обчислюється як для мірних, так і для рахункових ознак, що особливо необхідно мати на увазі при розрахунку середніх з відносних показників (наприклад, індекс накопичення ДНК — ІНДНК — в ядрах кліток). У інтервальних варіаційних рядах для визначення середньої арифметичної враховують середини інтервалів (x_i – напівсума крайніх їх значень).

В медицині використовуються також не тільки проста і зважена середні арифметичні, а й інші різновиди. При вивченні середніх приростів за певний період (вік, зростання патологічного осередку пухлини і ін.) необхідно обчислювати не просту середню, а середню геометричну (*G*), таку, що є коренем *n*-го степеня з добутків всіх варіант, що враховуються

$$G = \sqrt[n]{x_1 x_2 \ldots x_n} .$$

При вивченні, наприклад, просвітів судин слідує застосовувати середню квадратичну (S) – корінь квадратний з суми квадратів варіант радіусів усереднюваних кіл

$$S = \sqrt{\sum R_i^2}\ .$$

Середня гармонійна $H = \dfrac{n}{1/x_1 + 1/x_2 + ... + 1/x_n}$ використовується при усередненні змінних швидкостей розвитку патологічних процесів (наприклад, атеросклерозу, пухлини і ін.

2. Мода (*Mo*). Модою називають значення варіанти, що найбільш часто зустрічається. При нормальному розподілі значень ознак є тільки одна мода. При появі у ряді декількох вершин слід проаналізувати склад вибірки і виключити спостереження, що вискакують. В разі двовершинної (бімодальної) кривої розподілу, як це, наприклад, спостерігається при аналізі розподілу ядер клітин по плоїдності, необхідно шляхом виключення варіант, що спотворюють криву, зробити групу однорідною.

Мода для інтервального статистичного розподілу обчислюється таким чином. Спочатку визначається модальний інтервал $[x_m, x_{m+1})$, тобто такий інтервал, для якого $n_m / h_m = \max\limits_{1 \le i \le k} \{n_i / h_i\}$, де h_i – довжина частинного інтервалу $[x_m, x_{m+1})$, n_i – число варіант з цього інтервалу. Значення моди міститься всередині модального інтервалу і обчислюється за інтерполяційною формулою

$$M_o = x_m + \frac{n_m - n_{m-1}}{2n_m - n_{m-1} - n_{m+1}} h_m . \qquad (2.1)$$

3. Медіана (*Me*) – це значення варіанти, що ділить варіаційний ряд навпіл.

Медіаною для інтервального статистичного розподілу називається таке число M_e, для якого виконується рівність:

$$F^*(M_e) = 0{,}5\ ,$$

де $F^*(x)$ – емпірична функція цього розподілу.

Формула для обчислення медіани має такий вид:

$$M_e = x_m + \frac{0{,}5 - F^*(x_m)}{F^*(x_{m+1}) - F^*(x_m)}(x_{m+1} - x_m), \qquad (2.2)$$

де $[x_m, x_{m+1})$ – медіанний частинний інтервал $(1 \le m \le k)$, для якого виконуються нерівності $F^*(x_m) < 0{,}5$, $F^*(x_{m+1}) > 0{,}5$.

4. Квартилі (верхній $Q_в$ і нижній $Q_н$ – значення варіант, що ділять варіаційний ряд разом з медіаною на 4 частини. При нормальному розподілі між верхнім і нижнім квартилями розташовується половина всіх варіант.

У симетричному варіаційному ряді при нормальному розподілі значень варіант вказані загальні характеристики: мода, медіана і середнє арифметичне – по своєму розташуванню співпадають.

Після визначення узагальнених статистичних характеристик варіаційного ряду встановлюють другу його головну ознаку, що визначає якість сукупності, що вивчається, – різноманітність ознак, складових варіаційного ряду.

Різноманітність (варіабельність, коливність) ознак. Варіабельність ознак демонструється межами їх варіацій, середнім квадратичним (стандартним) відхиленням, дисперсією і коефіцієнтом варіації.

У випадках, коли більшість варіант ряду розташовується біля середньої арифметичної, графік такого ряду виглядає гостровершинним, і свідчить про малу мінливість ознаки, що вивчається. Значне розсіювання величин ознаки біля середньої арифметичної вже демонструє велику варіабельність структур, що вивчаються.

1. Варіабельність ознаки, що вивчається, приблизно демонструється межами (лімітами – lim), тобто розмахом (амплітудою) варіаційного ряду, (R)– різницею значень найбільшої ($\lim_{\max}$) і найменшої варіант ($\lim_{\min}$).

2. Середнє квадратичне відхилення (для генеральної сукупності – σ, для вибірки – $\sigma_в$) точніше характеризує коливання ряду, тобто ступінь відхилення варіант від їх середньої величини.

Обчислюють середнє квадратичне відхилення за формулою

$$\sigma_в = \sqrt{\frac{\sum_{i=1}^{k}(x_i - \bar{x}_в)^2 n_i}{n-1}}. \qquad (2.3)$$

Простіший спосіб визначення сигми, що носить орієнтовне значення, ґрунтується на визначенні лімітів і коефіцієнта Єрмолаєва – *K* (таблиця 2.3). Обчислення значень сигми проводять за формулою

$$\sigma_в = \frac{R}{K}. \qquad (2.4)$$

Мале значення сигми свідчить про однорідність досліджуваної групи, велике – про неоднорідність або про значну варіабельність ознаки.

Таблиця 2.3 – Величина *K* при різних обсягах вибірки

n	0	1	2	3	4	5	6	7	8	9
0	–	–	1,13	1,69	2,06	2,33	2,53	2,70	2,95	2,97
10	3,08	3,17	3,26	3,34	3,41	3,47	3,53	3,59	3,64	3,69
20	3,73	3,78	3,82	3,86	3,90	3,93	3,96	4,00	4,03	4,06
30	4,09	4,11	4,14	4,16	4,19	4,21	4,24	4,26	4,28	4,30
40	4,32	4,34	4,36	4,38	4,40	4,42	4,43	4,45	4,47	4,48
50	4,50									
100	5,02									

При нормальному розподілі 68,3% всіх значень варіюючої ознаки

знаходяться в межах розмаху одного квадратичного відхилення ($\pm\sigma$); при 95,5% – в межах двох сигм ($\pm 2\sigma$) і 99,7 % – в межах трьох сигм ($\pm 3\sigma$).

Правило трьох сигм дозволяє проводити також ряд розрахунків:

– Визначати кордони середніх (нормальних) значень ознаки. Найчастіше за норму набувають значень ознаки в межах $x \pm \sigma$. Інколи ці межі звужують до $x \pm 0{,}5\sigma$, залежно від особливостей патології, що вивчається. Інтервали норми і патологічного стану повинні помітно відрізнятися. Відзначимо, що при рівній імовірності знаходження ознаки в сусідніх групах (ймовірність, рівна 0,5) сигма має рівень 0,67 (стан "біфуркації" вибіркової групи показників).

– Визначати нормоване відхилення варіант – критерій Стьюдента (t)– від середньої арифметичної, а також можливу частоту її появи (у відсотках): $t = (x - \bar{x})/\sigma$.

– Вирішувати питання про приналежність даного спостереження до сукупності, що цікавить нас (при $t < 3$).

– Оцінювати варіанти, що вискакують ($x_в$). Право на виключення варіант з ряду з'являється за умови, якщо $t = (x - \bar{x})/\sigma$ буде більше 3 (значення x і σ отримують заздалегідь, виключивши варіанти, що вискакують).

– Будувати теоретичний ряд, що відповідає нормальному розподілу варіант.

3. Дисперсію (D) – суму квадратів центральних відхилень від середньої – використовують у ряді випадків як первинну міру різноманітності. Враховують значення варіанси – квадрату сигми (σ^2), яка є дисперсією, що приходить на один елемент різноманітності.

4. Коефіцієнт варіації (CV) дає можливість судити про ступінь розсіяння варіант навколо середньої. Його отримують (у відсотках) шляхом ділення величини середньоквадратичного відхилення на величину середнього арифметичного $CV = \frac{\sigma}{\bar{x}} \cdot 100$.

Значення коефіцієнта CV нижчі 10% демонструють малу варіацію, до 20 %

– середню і більше 20 % – сильну.

Вибірковий метод дослідження. У практиці патологоанатомічних досліджень зустрічаються випадки, коли число спостережень не перевищує 30. Для цих вибірок доводиться враховувати дещо збільшені значення середньоквадратичних відхилень шляхом зменшення у формулі кількості спостережень *n* на одиницю (2.3). У ряді випадків використовують також таблицю Єрмолаєва (див. табл. 2.3).

Встановлено, що при великому числі спостережень для охоплення 95% всіх очікуваних спостережень слід скористатися значенням довірчого коефіцієнта *t*, рівне 1,96, а при обхваті 99 % випадків – *t* повинно бути рівне 2,58 або 3.

При вивченні малих вибірок значення *t* отримують з таблиці стандартних значень критерію Стьюдента, в якій по числу степенів свободи (*n*) визначаються рівні безпомилкових суджень (β) і ймовірності випадковості відмінностей (*p*) порівнюваних сукупностей (рисунок 2.3).

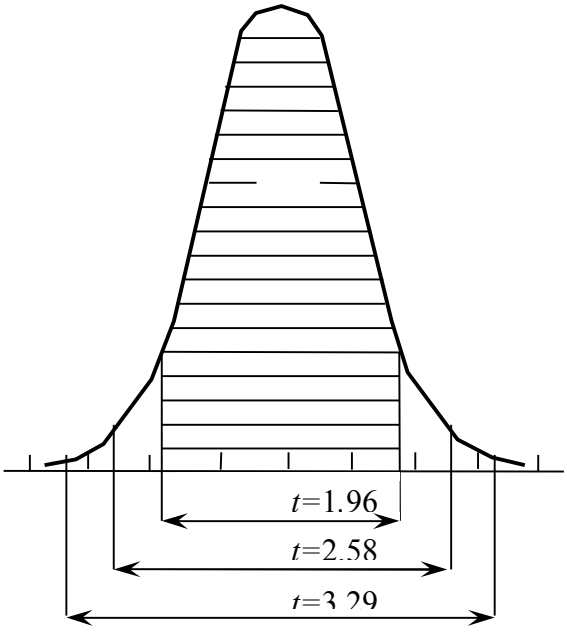


Рисунок 2.3 – Нормальний розподіл. *t* – число сигм, на яке повинні відхилятися показники від середньої межі (M–*t*σ і M+*t*σ), щоб задана ймовірність появи ознаки β дорівнювала 0,95; 0,98; 0,999

При морфометричних дослідженнях зустрічаються, як це було сказано вище, три види розподілу варіант вимірів: нормальний, біноміальний і для рідких подій. Кожен з цих видів варіаційних розподілів вимагає спеціальних методів подальшої статистичної обробки. Особливої уваги заслуговує робота з вибірками, які складають основну масу морфометричних досліджень.

Працюючи з вибірками, важливо постійно мати уявлення про достовірність отриманих вибіркових показників.

1. Помилки репрезентативності (представництва). Так, для генеральної сукупності, що прямує до нескінченності, помилка визначається за формулами: при нормальному розподілі мірних показників помилка вибірки (m) прямо пропорційна середньому квадратичному відхиленню і обернено пропорційна до кореня квадратного з числа спостережень

$$m = \pm\frac{\sigma}{\sqrt{n}}, \text{ при } n \geq 30; \qquad (2.5)$$

$$m = \pm\frac{\sigma}{\sqrt{n-1}}, \text{ при } n < 30. \qquad (2.6)$$

При малих вибірках *n*, що знаходиться в знаменнику, зменшують на одиницю). Для визначення помилок вибірки рахункових ознак ($m_{\text{р}}$) використовують іншу формулу

$$m_{\text{р}} = \pm\sqrt{\frac{pq}{n}}.$$

По розмірах середньої помилки можна судити, наскільки знайдена вибіркова середня величина відрізняється від середньої генеральної сукупності. Знаючи середню арифметичну з її помилкою, можна визначити

довірчі границі, в яких з певною вірогідністю може знаходитися середня генеральна сукупність ($M_{ген}$).

Мірою коливань вибіркових середніх показників є середнє квадратичне відхилення ($\sigma_{ген}$). Виходячи з правила трьох сигм, середня генеральна сукупність з імовірністю 68,3% знаходиться в межах $M \pm m$, а для більшої надійності і в ширших довірчих межах – $M \pm 2m$, що підвищує ймовірність безпомилкового судження до 95,5%. Цей рівень визнається достатнім в біологічних і морфологічних дослідженнях. Середня з інтервалом в 3 помилки ($M \pm 3m$) збільшує надійність її значення до 99,7%.

2. Оцінка достовірності вибіркових показників. Групові властивості вибірок характеризуються всіма описаними вище показниками, які називаються вибірковими. Представницькість може мати різний ступінь вираженості, яка характеризується відповідними помилками репрезентативності.

Довірчий коефіцієнт позначають через t (критерій Стьюдента), а ступінь надійності (довірчу ймовірність) виражають у відсотках або долях одиниці ($1 - p$). Віднімаючи з 1 довірчу ймовірність, отримують рівень значущості (ризик помилки) – p, який можна виражати в відсотках. У таблиці 2.4 приводяться декілька найбільш вживаних значень цих параметрів.

Слід мати на увазі, що для досягнення мінімальної надійності виводу при великому числі спостережень зазвичай приймаються значення t=1,96 і рівня значущості p=0,05. Таким чином, при роботі з малими вибірками можна завжди встановити межі, в яких може знаходитися генеральна середня.

Таблиця 2.4 – Критерії безпомилкових суджень

Критерії	Значення				
Довірчий коефіцієнт t	1,96	2,0	2,6	3,0	3,3
Довірча ймовірність $1 - p$	0,95	0,955	0,99	0,997	0,999
Рівень значущості p	0,05	0,045	0,01	0,003	0,001

Середня арифметична генеральна дорівнює вибірковій середній з межами

коливань величин добутку вибіркової помилки (m) на довірчий коефіцієнт t:

$$M_{ген} = \bar{x} \pm tm . \qquad (2.7)$$

Значення критерію надійності t (Стьюдента) отримують з таблиці 2.4. При проведенні морфометричних досліджень перший поріг вірогідності безпомилкової думки допустимий при звичайних вимогах до точності досліджень (t = 1,96 або 2, об'єм вибірки n рівний або більше 30 спостережень), другий поріг використовують при перевірочних спостереженнях (t = 2,58, об'єм вибірки – 100 і більше спостережень). Третій поріг – при дуже високих вимогах до точності досліджень (t = 3,3, вибірка складає не менше 200–400 спостережень).

При визначенні обсягу вибірки (n) зручно користуватися таблицями достатньо великого числа одиниць спостережень для рахункових ознак (табл. 2.1) і для випадкових мірних величин (табл. 2.2). У цих таблицях приведені також міри ризику (вірогідність безпомилкових думок – p).

У практиці морфометричних досліджень досить використовувати три пороги вірогідності безпомилкових прогнозів: β_1 = 0,95; β_2 = 0,99; β_3 = 0,999 і відповідними їм ймовірностями p (0,05; 0,01; 0,001) випадковості відмінностей між тими сукупностями, що вивчаються.

2.3 Алгоритми обчислення числових характеристик статистичних вибірок морфометричних даних

Ми розглянули аналіз статистичного дослідження, націлений на біологічні явища, що проходять в організмі. В даний час, як сказано вище, для проведення статистичного аналізу використовують різні комп’ютерні програми. Проте

патологоанатом повинен знати всі елементи і етапи морфолого-статистичного аналізу для правильного розуміння патологічного процесу, що вивчається. Результати аналізу повинні представлятися не тільки комп'ютерною роздруківкою результатів обробки даних, а й науковим аналізом.

В нашому розпорядженні є біомедичні зображення клітин людини, з яких формуємо статистичні дані для подальших досліджень, а саме маємо показники: Площа клітини, Площа ядра, Площа цитоплазми. Окрім цих мірних ознак, обчислимо ядерно-цитоплазматичне відношення (ЯЦВ), яке являє собою відношення площі ядра до площі цитоплазми. Це є, на перший погляд, нескладний показник, але він є досить інформативним. За допомогою ЯЦВ надалі будемо робити основний науковий аналіз.

Розглянемо алгоритм розрахунку числових характеристик і їх аналізу:

1. Формуємо дані вимірювання. При визначенні обсягу вибірки використовуємо таблиці 2.1 або 2.2.
2. Групуємо їх правильно по: нормі, дисплазії III-А, III-Б, III-В ступенів.
2. Визначаємо оптимальну кількість груп для вивчення біомедичних зображень.
3. Обчислюємо ЯЦВ для кожної групи статистичних даних.
4. Будуємо аналітично варіаційний ряд біомедичного дослідження.
5. Представляємо варіаційний ряд з допомогою гістограми.
6. Представляємо варіаційний ряд з допомогою полігону.
7. При нарощуванні частот або збільшенні величини ознаки будуємо кумуляту.
8. В результаті кроків 6-8 отримаємо візуалізацію досліджуваного процесу.
9. Обчислюємо середню арифметичну.
10. Знаходимо моду. При появі декількох мод аналізуємо склад вибірки і викидаємо спостереження, що «вискакують».
11. Знаходимо медіану.

12. Знаходимо квартилі.

13. За результатами кроків 10-13 робимо припущення про вид розподілу вибірки.

14. Обчислюємо розмах варіації, який дорівнює різниці між найбільшою та найменшою варіантами розподілу.

15. Знаходимо різниці між середньою і кожною варіантою, піднімаємо їх до квадрату.

16. Множимо відхилення знайдені на кроці 16 на відповідні частоти і сумуємо їх.

17. Ділимо результат кроку 17 на число спостережень n (або на $n-1$ при малій вибірці). В результаті отримаємо дисперсію.

18. Обчислюємо середнє квадратичне відхилення за формулою (2.3).

19. Обчислюємо коефіцієнт варіації.

20. За результатами кроків 15-20 робимо висновки про ступінь розсіювання (мала, середня, сильна) варіант навколо середньої.

21. Обчислюємо помилки репрезентативності вибірки і робимо висновки по розміру помилки, наскільки середня величина відрізняється від середньої генеральної сукупності.

22. З таблиці 2.4 вибираємо довірчий коефіцієнт t для вибраного ступеня надійності p.

23. Знаходимо межі середньої арифметичної генеральної за формулою (2.7).

24. Робимо перевірку критерію достовірності різниці середніх показників, що характеризують порівнювані групи.

25. Робимо перевірку приналежності вибіркових даних нормальному (біноміальному, показниковому, Пуасона і т.д.) розподілу із визначеними параметрами за критерієм узгодженості Пірсона (χ^2):

$$\chi^2 = \sum_{i=1}^{m} \frac{(n_i - n_i^0)^2}{n_i^0} \qquad (2.8)$$

де n_i – частота варіанти x_i, $n_i^0 = np_i$ – теоретична частота варіанти x_i, n – обсяг вибірки, p_i – імовірність варіанти x_i, яка обчислюється за формулою:

$$p_i = \Phi\left(\frac{x_{i+1} - \bar{x}}{\sigma}\right) - \Phi\left(\frac{x_i - \bar{x}}{\sigma}\right), \qquad (2.9)$$

де $\Phi(x)$ – функція Лапласа.

26. Здійснюємо узагальнені висновки стосовно узагальнених показників значень варіантів морфометричних ознак, їх варіабельності, розподілу, меж середньої арифметичної генеральної.

2.4 Висновки до розділу 2

Отже, в даному розділі розроблені алгоритми визначення числових характеристик статистичних даних біомедичних зображень, алгоритми кореляційного, регресійного і дисперсійного аналізів, які лягли в основу побудови системи аналізу біомедичних зображень.

3 БАЗА ДАНИХ ІМУНОГІСТОХІМІЧНИХ ЗОБРАЖЕНЬ

3.1 Інтерфейс ImageJ

Дана програма реалізована у вигляді плагіну до ImageJ. Оскільки програмі не потрібне зображення в якості вхідного аргументу то плагін реалізує інтерфейс PlugIn. Після того як плагін викликається із меню плагінів “Plugins/Diagnosis/AMS Diagnosis” (рисунок 3.1) запускається метод run.

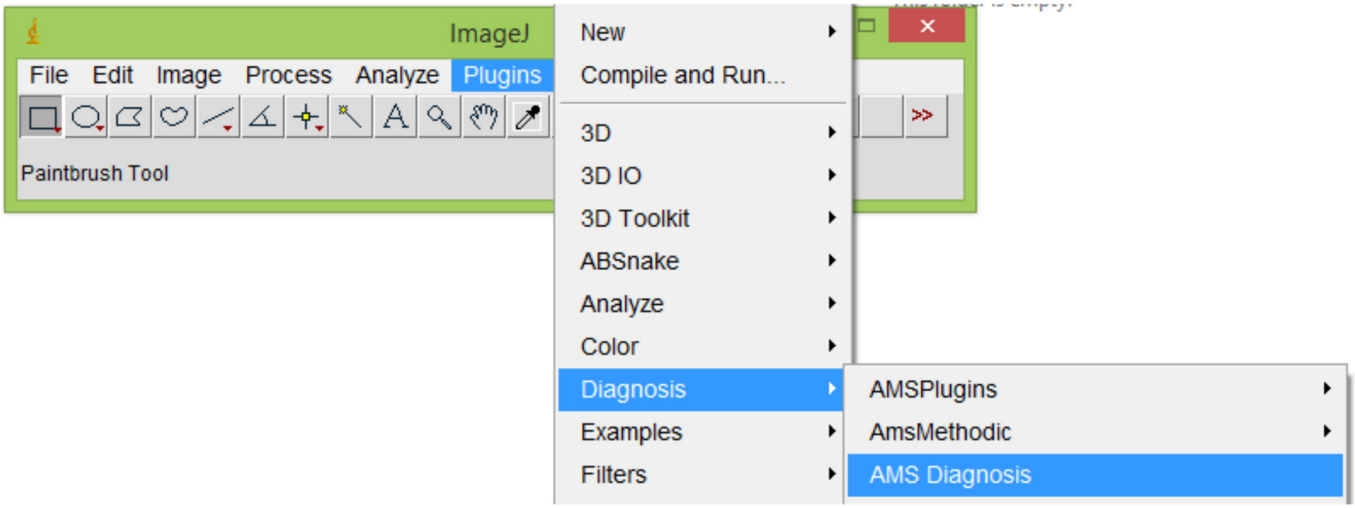


Рисунок 3.1 – Запуск плагіну AMS Diagnosis

При першому звернені до класу AMS_Diagnosis виконується його статична ініціалізація, де виконується необхідні налаштування та ініціалізація.

Після запуску у методі run показується головне вікно програми, яке наведено на рисунку 3.2.

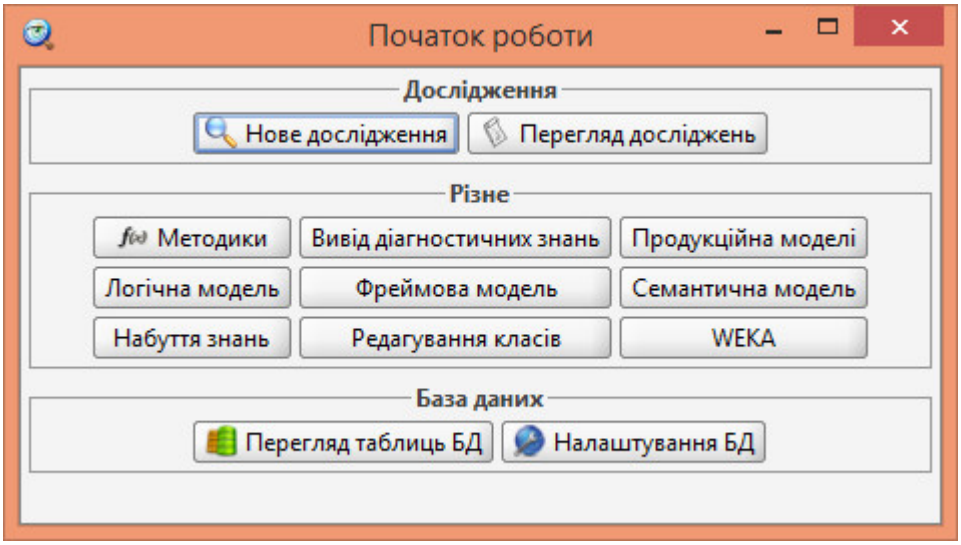


Рисунок 3.2 – Головне вікно плагіну AMS Diagnosis

3.2 Налаштування БД та її перегляд

Перед початком роботи із програмою необхідно запустити базу даних і вказати налаштування для підключення (рисунок 3.3).

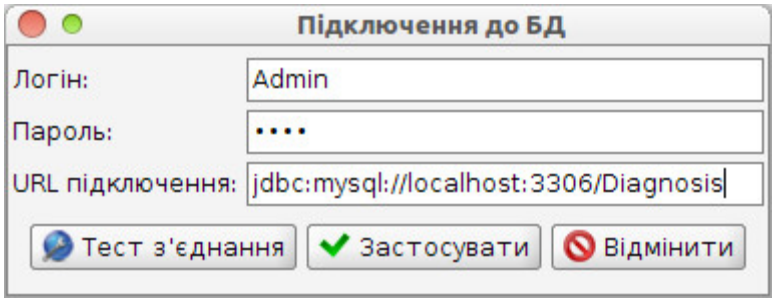


Рисунок 3.3 – Налаштування підключення до бази даних

Нижче наведений детальний опис про URL підключення до БД:

– mysql – драйвер бази даних MySQL;
– localhost – адрес сервера бази даних;
– 3306 – порт сервера бази даних;
– Diagnosis – назва бази даних.

Після введення параметрів підключення до БД можна протестувати чи вдається встановити з'єднання (рисунок 3.4).

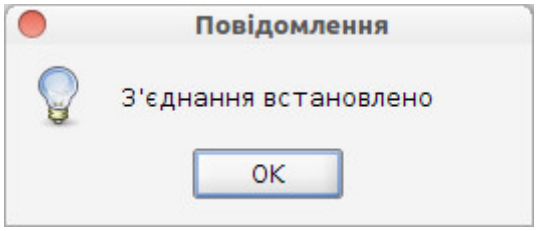


Рисунок 3.4 – Повідомлення про успішне встановлення з'єднання із БД

Також існує можливість переглянути структуру БД, а також даних які вона зберігає (рисунок 3.5).

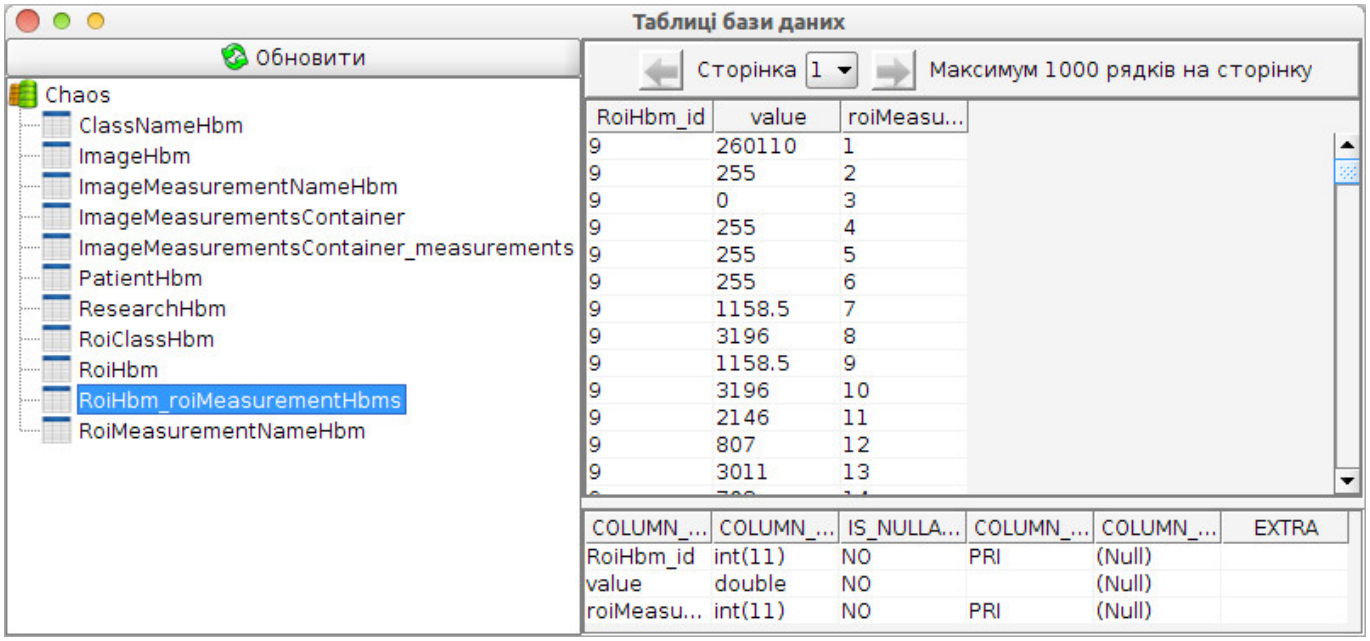


Рисунок 3.5 – Перегляд структури та даних таблиць БД

3.3 Функції графічного інтерфейсу користувача

Вікно для побудови методик (рисунок 3.6) являє собою, щось подібне на графічне програмування і використовується для створення методик та логіки їх

виконання.

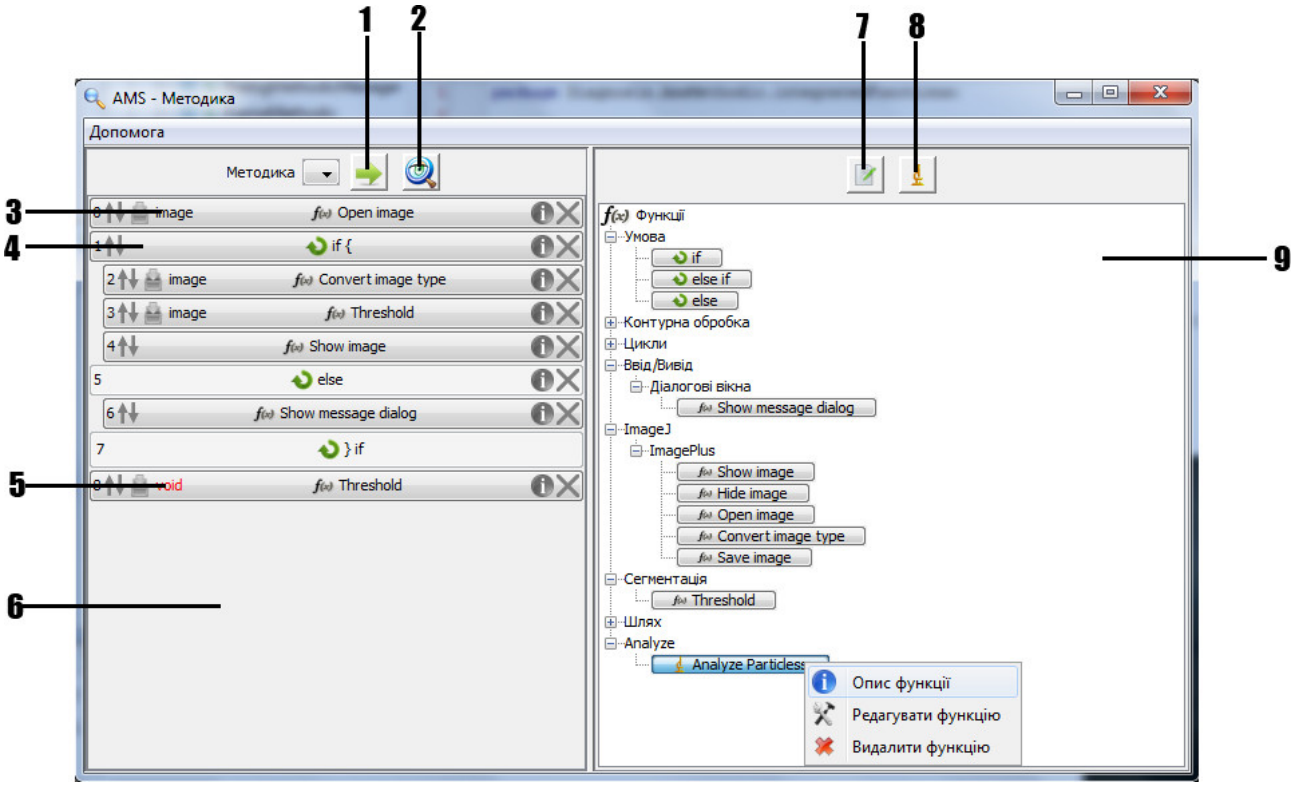


Рисунок 3.6 – Вікно для побудови методик та функцій

З вищенаведеного рисунку можна побачити наступні елементи меню:

– запуск методики або функції на виконання для тестування роботи (доступно лише для методик та функцій без вхідних параметрів);

– виклик діалогового вікна для управління методиками;

– назва змінної у яку присвоюється результат функції;

– функція, що нічого не повертає;

– результат функції ігнорується;

– панель, що містить послідовність функцій для виконання;

– виклик діалогового вікна для створення функції з уже існуючих функцій;

– виклик діалогового вікна для додання ImageJ плагіну у список функцій;

– дерево доступних функцій.

Для побудови методики потрібно перемістити необхідні функції з правої частини вікна у ліву. Для цього необхідно натиснути ліву кнопку мишки на

потрібній функції та перемістити її у ліву панель.

Задання вхідних параметрів функції здійснюється за допомогою діалогового вікна (рисунок 3.7), для його виклику потрібно натиснути лівою кнопкою мишки по функції. Значення вхідного параметра “Не вибрано” не дозволяє запустити методику чи функцію на виконання. Константи – це наперед задані можливі варіанти для вхідного параметра. Змінні містять значення, що отримуються як результат виконання функції. Якщо тип параметра з наступних: String, boolean, char, byte, short, int, long, float, double або їх клас “обгортка”, то їх значення можна ввести вручну.

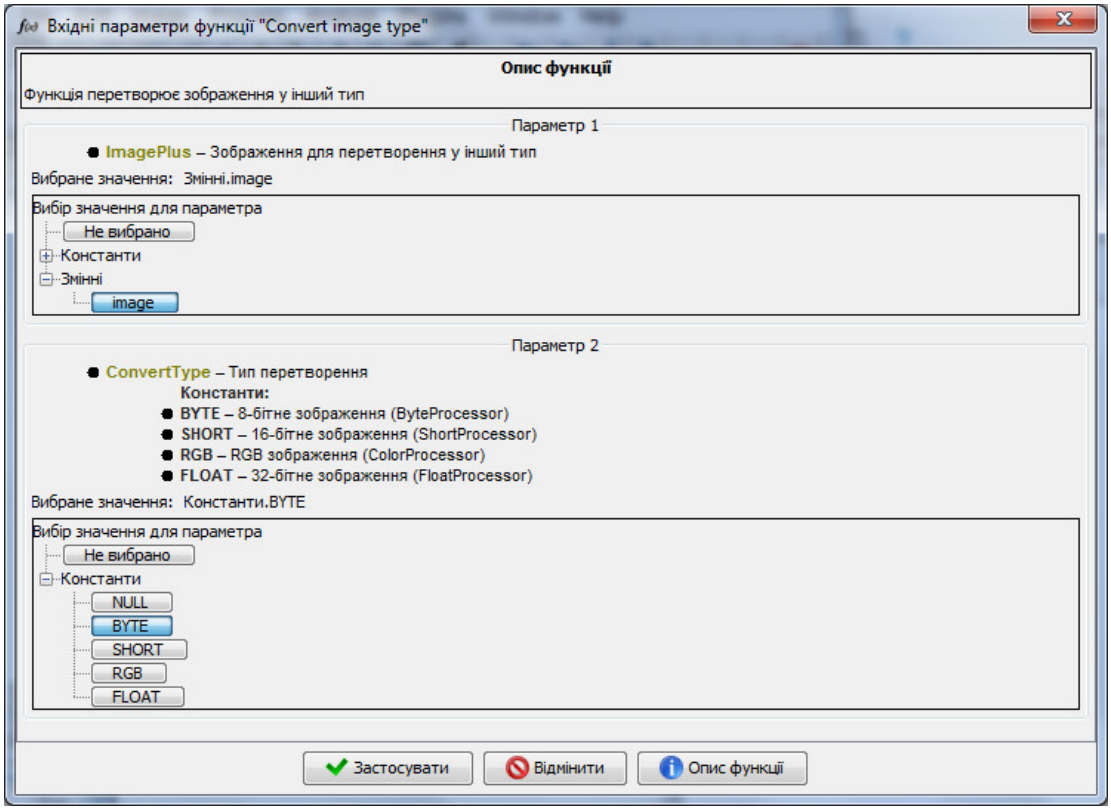


Рисунок 3.7 – Діалог задання вхідних параметрів для функції

Щоб задати назву змінної для присвоєння, потрібно натиснути лівою кнопкою мишки на відповідну іконку (пункт 3 на рисунку 3.6). Після чого відкриється діалогове вікно для задання назви змінної (рисунок 3.8).

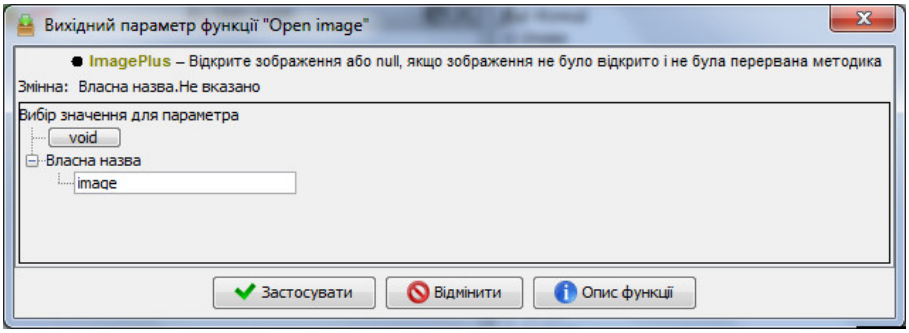


Рисунок 3.8 – Діалог задання імені змінної для вихідного параметра функції

Для повного опису функції потрібно натиснути на відповідну кнопку після чого відкриється вікно з повним описом функції (рисунок 3.9).

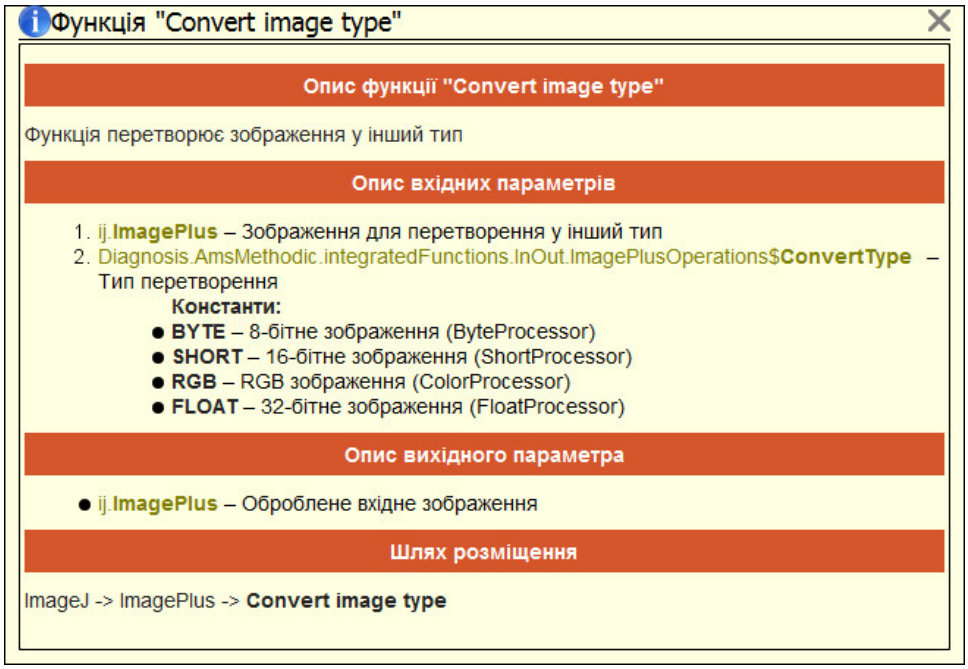


Рисунок 3.9 – Вікно з повним описом функції

Щоб додати ImageJ плагін існує спеціальне діалогове вікно (рисунок 3.10) для генерування функції “обгортки” (пункт 8 на рисунку 3.6). Для цього необхідно вказати назву плагіну, яка використовується для його виклику, цю назву можна дізнатися за допомогою вбудованого в ImageJ записувача макросів (рисунок 3.11). Для цього потрібно запустити його та викликати ImageJ плагін. Якщо ImageJ плагін містить діалогове вікно для задання вхідних параметрів, то

необхідно поставити галку “Містить параметри”. Якщо ImageJ плагін потребує вхідного зображення, то це теж потрібно відмітити. Також, якщо плагін повертає зображення, то це теж потрібно відмітити. Якщо ImageJ плагін містить діалогове вікно з параметрами, то необхідно зробити так, щоб воно відкрилося при натисненні на кнопку “Згенерувати функцію”. Наприклад, для відкриття діалогового вікна плагіна “Analyze Particles…” потрібно, щоб над зображенням була виконана сегментація. Діалогове вікно для редагування згенерованої функції “обгортки” для ImageJ плагіну наведено на рисунку 3.12.

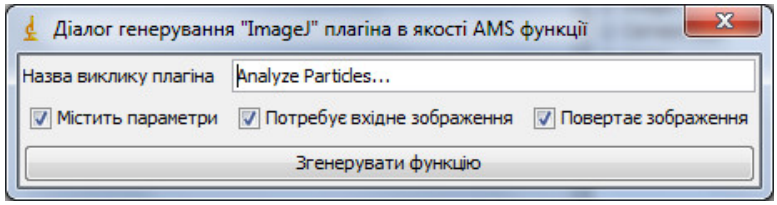


Рисунок 3.10 – Діалог генерування функції “обгортки” для ImageJ плагіна

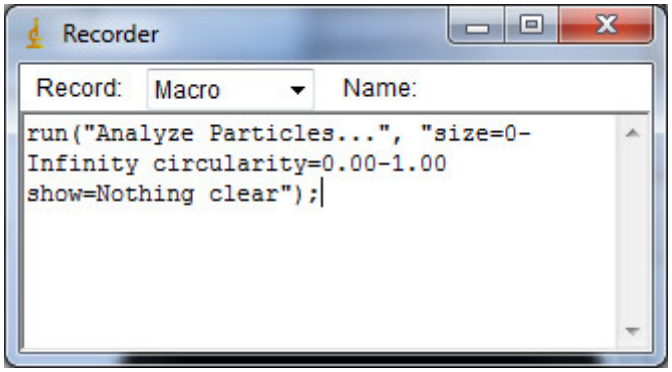


Рисунок 3.11 – Записувач макросів

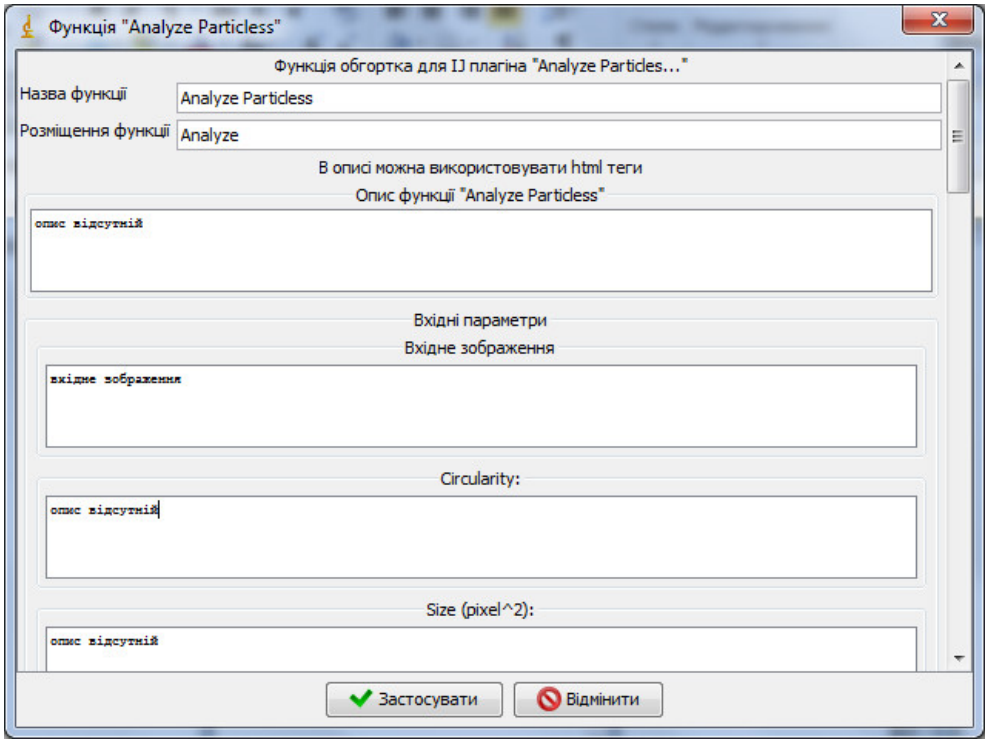


Рисунок 3.12 – Діалогове вікно для редагування згенерованої функції “обгортки” для ImageJ плагіна

Функція для задання умови має особливе діалогове вікно для редагування умов (рисунок 3.13). Якщо поставити галку на “Інвертувати результат”, то результат цього вираження буде інвертований (якщо результат істина, то стане не істиною і навпаки). Кнопки “+” та “–” додають та видаляють дужки, які можна використовувати для задання пріоритету. Кнопка “Добавити вираження” додає нижче цієї кнопки ще одне вираження, а кнопка “закрити” відповідно його видаляє. Діалогове вікно контролює правильність задання дужок і в разі їх неправильного задання буде виведена відповідна помилка (рисунок 3.14).

Рисунок 3.13 – Діалогове вікно для задання умови

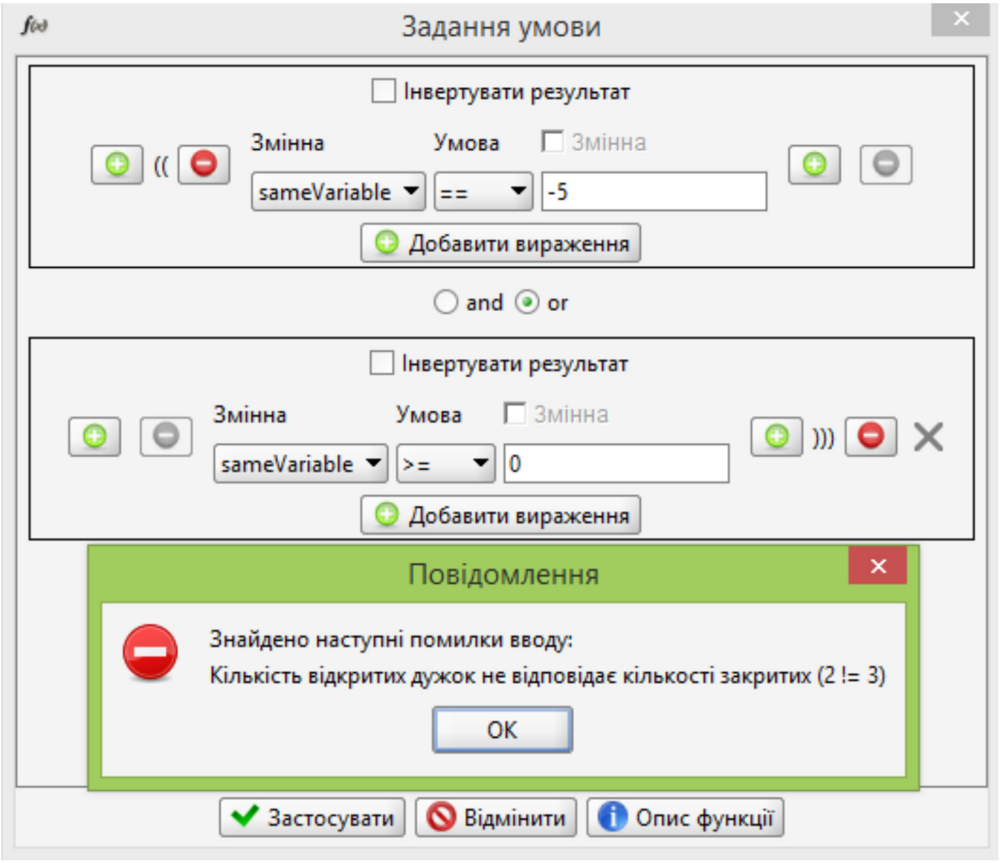


Рисунок 3.14 – Повідомлення про нерівність відкритих дужок закритим

3.4 Класи інтерфейсу користувача

Діаграма основних класів, що відносяться до інтерфейсу користувача наведена на рисунку 3.15.

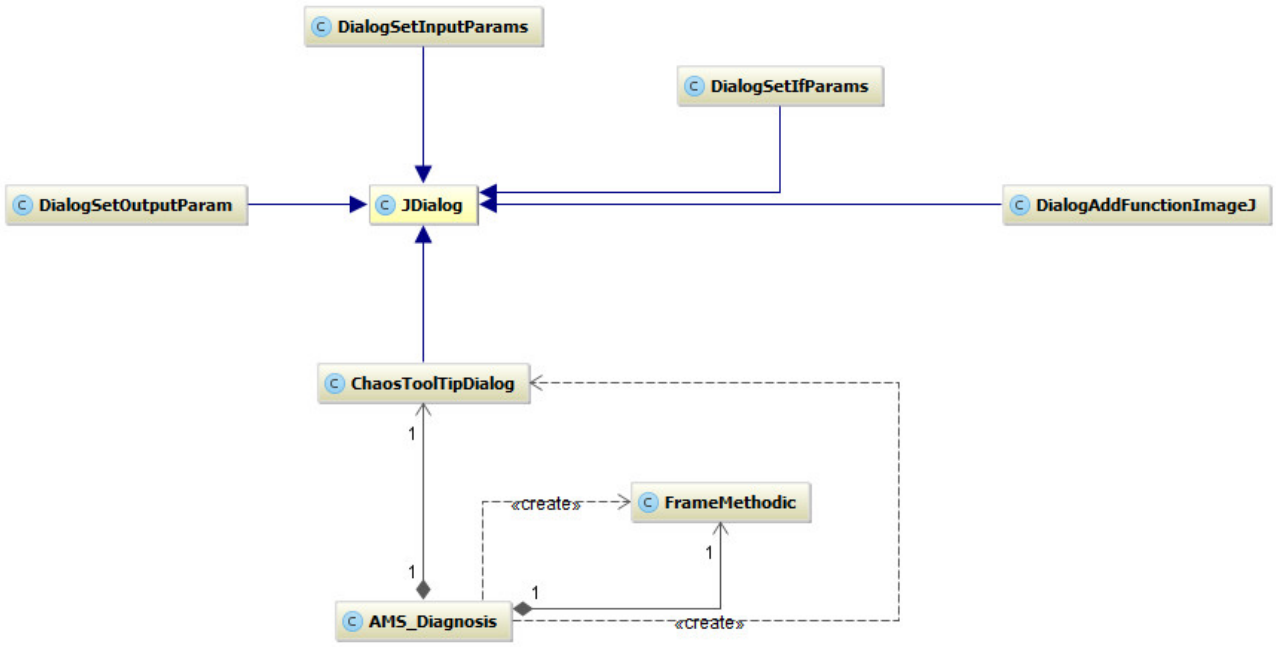


Рисунок 3.15 – Діаграма основних класів, що відносяться до інтерфейсу користувача

З рисунку можна побачити наступні класи:

– “AMS_Diagnosis” – початкове вікно програми (рисунок 3.2);

– “FrameMethodic” – вікно для побудови методик та функцій (рисунок 3.6);

– “DialogSetInputParams” – діалогове вікно для задання вхідних параметрів функції (рисунок 3.7);

– “DialogSetOutputParam” – діалогове вікно для задання назви змінної, що повертається функцією (рисунок 3.8);

– “ChaosToolTipDialog” – діалогове вікно з підказками або описом

функцій (рисунок 3.9);

– “DialogAddFunctionImageJ” – діалогове вікно для генерування функції обгортки для ImageJ плагіна (рисунок 3.10);

– “DialogSetIfParams” – діалогове вікно для задання умови (рисунок 3.13).

На рисунку 3.16 наведена діаграма класів, які пов’язані з функціями методики.

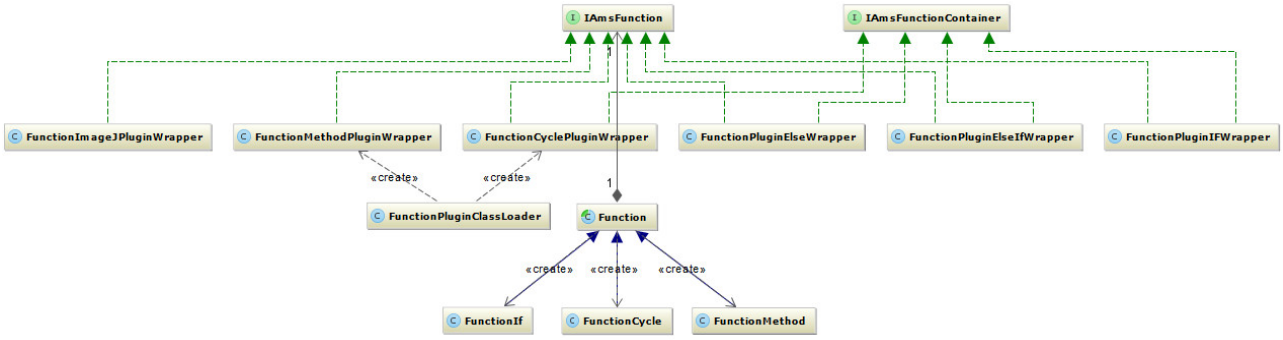


Рисунок 3.16 – Діаграма класів, які відносяться до функцій методики

З рисунку можна побачити, що наступні класи реалізують інтерфейс “IAmsFunction”:

– “FunctionImageJPluginWrapper”;

– “FunctionMethodPluginWrapper”;

– “FunctionCyclePluginWrapper”;

– “FunctionPluginElseWrapper”;

– “FunctionPluginElseIfWrapper”;

– “FunctionPluginIfWrapper”.

Інтерфейс “IAmsFunction” описує базові методи, які повинні бути реалізовані для того, щоб функція появилася у списку доступних (права частина методики із списком доступних функцій, яку можна побачити з рисунку 3.6). Кожен з вищенаведених класів реалізує методи цього інтерфейсу згідно із своїми

уподобаннями. Повний код інтерфейсу з коментарями наведений нижче:

```
package Diagnosis.AmsMethodic.functions.interfaces;

import Diagnosis.AmsMethodic.functions.parameters.FunctionParameter;
import javax.swing.*;

/**
 * Клас повинен реалізовувати даний інтерфейс, щоб добавитись у список доступних
 * функцій методики.
 */
public interface IAmsFunction {
    /**
     * @return повний опис функції (включає опис функції, опис вхідних
     * параметрів, опис вихідного параметру...)
     */
    public String getFullFunctionDescription();

    /**
     * @return загальний опис функції
     */
    public String getFunctionDescription();

    /**
     * @return шлях, де буде розміщена функція. В якості розділення
     * використовується символ крапки ".".<br>
     *    Для прикладу, якщо getTreePath повертає "Відкриття файлів.Зображення", то
     * функція буде розміщена
     * у "Відкриття файлів->Зображення->Назва функції"
     */
    public String getTreePath();

    /**
     * @return назва функції, що буде відображатися у програмі
     */
    public String getFunctionName();

    /**
     * @return клас у якому знаходиться функція(метод)
     */
    public Class<?> getFunctionClass();

    /**
     * @return вхідні параметри функції
     */
    public FunctionParameter[] getFunctionInputParams();

    /**
     * @return параметер, що функція повертає
     */
    public FunctionParameter getFunctionReturnParam();

    /**
     * @return іконка функції
     */
    public ImageIcon getImageIcon();

    /**
     * Метод, що буде викликатися інтерпретатором на виконання
     * @param inputParams вхідні параметри, що будуть передаватися
     * інтерпретатором
```

```
     * @return Результат, що повертає функція
     */
    public Object invoke(Object... inputParams) throws Exception;
}
```

Також з рисунку 3.16 можна побачити, що наступні функції реалізують інтерфейс “IAmsFunctionContainer”:

– “FunctionCyclePluginWrapper”;
– “FunctionPluginElseWrapper” ;
– “FunctionPluginElseIfWrapper”;
– “FunctionPluginIfWrapper”.

Даний інтерфейс не містить жодних методів і лише є знаком того, що функція є контейнером для інших функцій, тобто в її тілі можуть бути розміщені інші функції.

Клас “FunctionPluginClassLoader” призначений для завантаження плагінів із носія інформації, їх розпізнавання та додання у список доступних функцій. Як можна побачити з рисунку 3.16 в даний момент він розпізнає лише 2 плагіни, “FunctionMethodPluginWrapper” та “FunctionCyclePluginWrapper”.

Також з рисунку 3.16 можна побачити, що інтерфейс “IAmsFunction”, а отже і усі класи, що його реалізують пов’язані з абстрактним класом “Function”. Клас “Function” і всі його похідні класи будують уже саму методику (рисунок 3.6 ліва частина методики). Тобто користувачем задається порядок їх виконання, їх вхідні та вихідні параметри. Сам клас реалізує багато функцій, наприклад будує кнопки, їх представлення та дії, що відбуваються по їх натисненню, а класи, що походять від нього задають ці дії, коректують представлення функції у лівому вікні або роблять усе, що їм потрібно.

На рисунку 3.17 – зображена діаграма класів пов’язаних з функцією “обгорткою” для існуючих ImageJ плагінів.

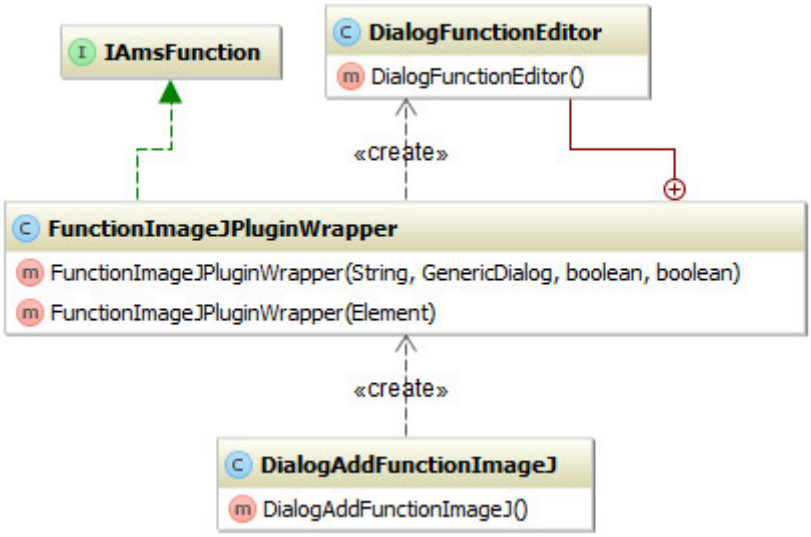


Рисунок 3.17 – Діаграма класів функції “обгортки” для існуючих ImageJ плагінів

Після того як користувач ввів відповідні дані і не відбулося помилок у діалоговому вікні “DialogAddFunctionImageJ” (рисунок 3.10), то генерується функція “обгортка” “FunctionImageJPluginWrapper” за допомогою введених користувачем параметрів та діалогового вікна самого ImageJ плагіну. Для цього використовується конструктор класу “обгортки” для генерування параметрів плагіну із діалогового вікна, який виглядає наступним чином:

```
 /**
 * Даний конструктор викликається для генерування параметрів функції із
 * діалогового вікна ImageJ плагіна
 * @param gd Вікно з параметрами плагіна для аналізу. Якщо == null, то
 * плагін без параметрів
 * @param pluginNameToRun Назва плагіна для виклику
 * @throws NoSuchFieldException
 * @throws IllegalAccessException
 */
public FunctionImageJPluginWrapper(String pluginNameToRun,
GenericDialog gd,
        boolean isImageRequired, boolean isImageReturn) throws
                NoSuchFieldException, IllegalAccessException
{...}
```

Для розпізнавання параметрів використовується Java Reflection API, що дозволяє аналізувати клас під час виконання програми, а також отримувати доступ навіть до “private” полів та методів [34-37].

Згенерована функція “обгортка” зберігається у носії інформації в XML форматі для відновлення згенерованих функцій при наступному запуску програми. Конструктор для відновлення функції із XML формату виглядає наступним чином:

```
            /**
             * Даний конструктор використовується для загрузки функцій із файлу
             * @param elemFunc Кореневий XML-тег (Function) функції, для відновлення
із
             * XML
             */
            public FunctionImageJPluginWrapper(Element elemFunc) throws
        ClassNotFoundException {...}
```

Клас “DialogFunctionEditor” як можна побачити з рисунку 3.12 використовується для редагування користувачем функції “обгортки” (здебільшого опис параметрів).

На рисунку 3.18 зображена діаграма класів із їх методами, що відносяться до інтерпретатора.

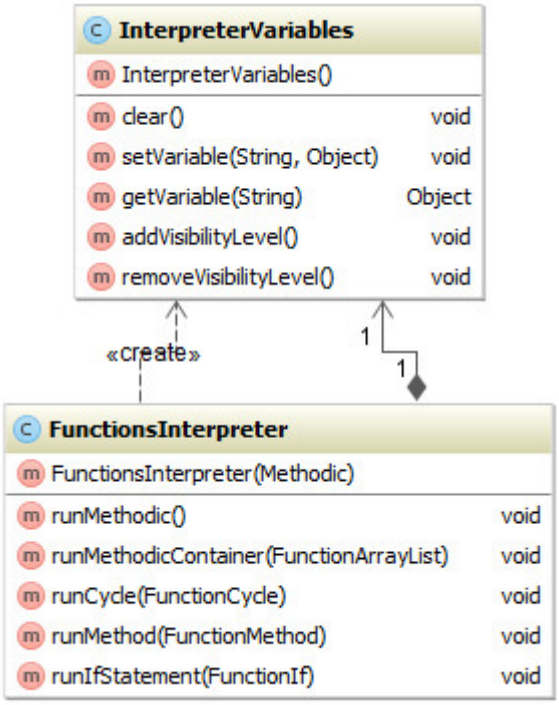


Рисунок 3.18 – Діаграма класів із методами, що відноситься до інтерпретатора

Клас “InterpreterVariables” містить змінні, що отримуються під час виконання інтерпретатором. Код класу наведений нижче:

```
package Diagnosis.AmsMethodic.functions.CompilerAndInterpreter;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

/**
 * Контейнер змінних для передачі у функції та їх отримання з них
 */
public class InterpreterVariables {
    private List<HashMap<String, Object>> listVariables = new ArrayList<HashMap<String, Object>>();

    InterpreterVariables() {
        listVariables.add(new HashMap<String, Object>());
    }

    /**
     * Видалити усі змінні
     */
    void clear() {
        listVariables.clear();
        listVariables.add(new HashMap<String, Object>());
    }

    /**
     * Пошук змінної у рівнях видимості, якщо зміна знайдена то переписується, якщо ні то
     * створюється у поточному рівні видимості
     */
    void setVariable(String varName, Object varValue) {
        for (int i = listVariables.size() - 1; i >= 0; i--) {
            if (listVariables.get(i).containsKey(varName)) {
                listVariables.get(i).put(varName, varValue);
                return;
            }
        }
        listVariables.get(listVariables.size() - 1).put(varName, varValue);
    }

    /**
     * @param varName Назва змінної для отримання її значення
     * @return Повертає значення змінної з заданим іменем
     * @throws IllegalArgumentException Якщо змінної з заданим іменем не знайдено
     */
    public Object getVariable(String varName) {
        for (int i = listVariables.size() - 1; i >= 0; i--) {
            if (listVariables.get(i).containsKey(varName)) {
                return listVariables.get(i).get(varName);
            }
        }
        throw new IllegalArgumentException("Variable with name '"+varName+"' not found");
    }

    /**
     * Додання рівня видимості
     */
    void addVisibilityLevel() {listVariables.add(new HashMap<String, Object>());}

    /**
     * Видалення рівня видимості
     */
    void removeVisibilityLevel() {
        if (listVariables.size() <= 1) throw new IllegalArgumentException("Can't remove first visibility level");
        listVariables.remove(listVariables.size() - 1);
    }
}
```

Клас “FunctionsInterpreter” запускає методику на виконання. Його код

наведений нижче:

```
package Diagnosis.AmsMethodic.functions.CompilerAndInterpreter;

import Diagnosis.AMS_Diagnosis;
import Diagnosis.AmsMethodic.functions.function.Function;
import Diagnosis.AmsMethodic.functions.function.FunctionCycle;
import Diagnosis.AmsMethodic.functions.function.FunctionIf;
import Diagnosis.AmsMethodic.functions.function.FunctionMethod;
import Diagnosis.AmsMethodic.functions.pluginWrappersAMS.FunctionMethodPluginWrapper;
import
Diagnosis.AmsMethodic.functions.pluginWrappersAMS.functionPluginImageJ.FunctionImageJPluginWrapper;
import Diagnosis.AmsMethodic.integratedFunctions.statementIF.FunctionPluginElseIfWrapper;
import Diagnosis.AmsMethodic.integratedFunctions.statementIF.FunctionPluginElseWrapper;

/**
 * Інтерпретатор функцій
 */
class FunctionsInterpreter {
    /**
     * Методика для виконання
     */
    private final Methodic methodic;

    /**
     * Контейнер змінних для передачі у функції та їх отримання з них
     */
    private final InterpreterVariables vars = new InterpreterVariables();

    /**
     * @param methodic методика для виконання
     */
    FunctionsInterpreter(Methodic methodic) {
        this.methodic = methodic;
    }

    /**
     * Запуск методики на виконання
     */
    void runMethodic() {
        if    (!methodic.isAllowedToRun())    throw    new    IllegalArgumentException("Методику
'"+methodic.getNameMethodic()+"' не дозволено запускати!");
        try{
            AMS_Diagnosis.FRAME_METHODIC.setLocked(true);
            // interpreter
            runMethodicContainer(methodic.getListFunctions());
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        finally {
            vars.clear();
            AMS_Diagnosis.FRAME_METHODIC.setLocked(false);
        }
    }

    /**
     * Виконання контейнера функцій
     */
    private void runMethodicContainer(FunctionArrayList container) throws Exception {
        for (Function function: container) {
            Class clsFunc = function.getClass();
            if (clsFunc.equals(FunctionIf.class)) {
                runIfStatement((FunctionIf) function);
            }
            else if (clsFunc.equals(FunctionCycle.class)) {
                runCycle((FunctionCycle) function);
            }
            else if (clsFunc.equals(FunctionMethod.class)) {
                runMethod((FunctionMethod) function);
            }
            else throw new IllegalArgumentException("Не відома функція");
        }
    }
```

```
        /**
         * Виконання функції типу "Цикл"
         */
        private void runCycle(FunctionCycle function) throws Exception {
            Object [] inputArgs = function.getInputArguments(vars);
            Iterable iterable = (Iterable) function.iAmsFunction.invoke(inputArgs);
            for (Object valueIteration: iterable) {
                vars.addVisibilityLevel();
                vars.setVariable(function.getReturnParamName(), valueIteration);
                runMethodicContainer(function.containFunctions);
                vars.removeVisibilityLevel();
            }
        }

        /**
         * Виконання функції типу "Метод"
         */
        private void runMethod(FunctionMethod function) throws Exception {
            //INPUT ARGUMENTS
            Object[] inputArgs = function.getInputArguments(vars);

            // RETURN ARGUMENT
            if (function.getReturnParamName().equals("void")) { // does not return parameter
                function.iAmsFunction.invoke(inputArgs);
            }
            else { // return parameter
                Object returnValue = function.iAmsFunction.invoke(inputArgs);
                vars.setVariable(function.getReturnParamName(), returnValue);
            }
        }

        /**
         * Виконання функції типу "Умова"
         */
        private void runIfStatement(FunctionIf functionIf) throws Exception {
            if (functionIf.containFunctions.size() == 0) throw new IllegalArgumentException("Size
of 'If statement can't be 0'");

            for (Function function: functionIf.containFunctions) {
                if (function.iAmsFunction instanceof FunctionPluginElseIfWrapper) {
                    if ( ((FunctionIf) function).isStatementTrue(vars) ) {
                        vars.addVisibilityLevel();
                        runMethodicContainer(function.containFunctions);
                        vars.removeVisibilityLevel();
                        break;
                    }
                }
                else if (function.iAmsFunction instanceof FunctionPluginElseWrapper) {
                    vars.addVisibilityLevel();
                    runMethodicContainer(function.containFunctions);
                    vars.removeVisibilityLevel();
                    break;
                }
                else throw new IllegalArgumentException("Не допустима функція");
            }
        }
    }
```

На рисунку 3.19 наведена діаграма класів, про які необхідно знати користувачеві, щоб додати власну функцію, яка написана на Java .

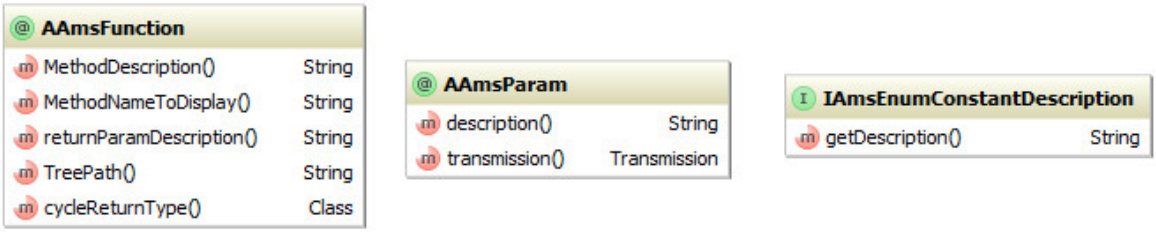


Рисунок 3.19 – Діаграма класів для додання функцій написаних на Java

Їх код та повний опис наведено нижче:

файл AAmsFunction.java:

```
package Diagnosis.AmsMethodic.functions.forUsers.anotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * Метод, що помічається даною анотацією появляється у вікні "AMS - Методика" в
 * якості функції.<br>
 * Якщо анотація використовується для не статичного метода то клас повинен
 * реалізовувати конструктор без параметрів (не обов'язково public).
 * Конструктор без параметрів буде
 * використовуватися для створення об'єкту кожен раз, коли в методиці буде
 * викликатися його функція (дійсно для не статичних методів),
 * Якщо анотація використовується для конструктора, то даний клас повинен
 * реалізовувати інтерфейс Iterable.
 *  Використання анотації над конструктором значить, що дана функція появиться в
 * якості цикла.
 */
@Target(value = {ElementType.METHOD, ElementType.CONSTRUCTOR})
@Retention(value = RetentionPolicy.RUNTIME)
public @interface AAmsFunction {
    /**
     * @return Опис методу, що буде відображатись у програмі.<br>
     * Допускається використання html тегів
     */
    String MethodDescription();

    /**
     * @return Назва методу, що буде відображатись у програмі
     */
    String MethodNameToDisplay();

    /**
     * @return Опис значення, що повертається (якщо метод типу void, то ця
     * строка ігнорується)<br>
     * Допускається використання html тегів
     */
    String returnParamDescription();

    /**
     * @return шлях, де буде розміщена функція. В якості розділення
     * використовується символ крапки ".".<br>
     *    Для прикладу, якщо getTreePath повертає "Відкриття файлів.Зображення", то
     * функція буде розміщена
     * у "Відкриття файлів->Зображення->Назва функції"
     */
    String TreePath();

    /**
     *  Якщо анотація використовується для конструктора (цикл), то цей параметр
     * мусить вказати дійсний тип значення (клас об'єкта), що повертається
     * ітератором.
     * Для методів ігнорується
     */
```

```
        Class cycleReturnType() default Object.class;
    }
```

файл AAmsParam.java:

```
    package Diagnosis.AmsMethodic.functions.forUsers.anotations;

    import java.lang.annotation.*;

    /**
     * Опис параметра
     */
    @Target(ElementType.PARAMETER)
    @Retention(RetentionPolicy.RUNTIME)
    public @interface AAmsParam {
        /**
         * Спосіб передачі параметра
         */
        enum Transmission {
            /**
             * Передача параметра допускається лише по силці
             */
            BY_LINK,
            /**
             * Передача параметра допускається лише за значенням
             */
            BY_VALUE,
            /**
             *  Спосіб передачі задається користувачем при введені вхідних
параметрів
             * функції (використовується по замовчуванням)
             */
            USER_SELECT}

        /**
         * @return Опис параметра, що буде відображатись у програмі<br>
         * допускається використання html тегів
         */
        String description();

        /**
         * @return Спосіб передачі параметра у функцію
         */
        Transmission transmission() default Transmission.USER_SELECT;
    }
```

Після цього, щоб метод появився у списку доступних функцій методики, потрібно архівувати ці скомпільовані класи у “.jar” архів та помістити його у папку “plugins” плагіна “AMS Diagnosis”. Результат додання вищенаведеного Java методу наведений на рисунку 3.20.

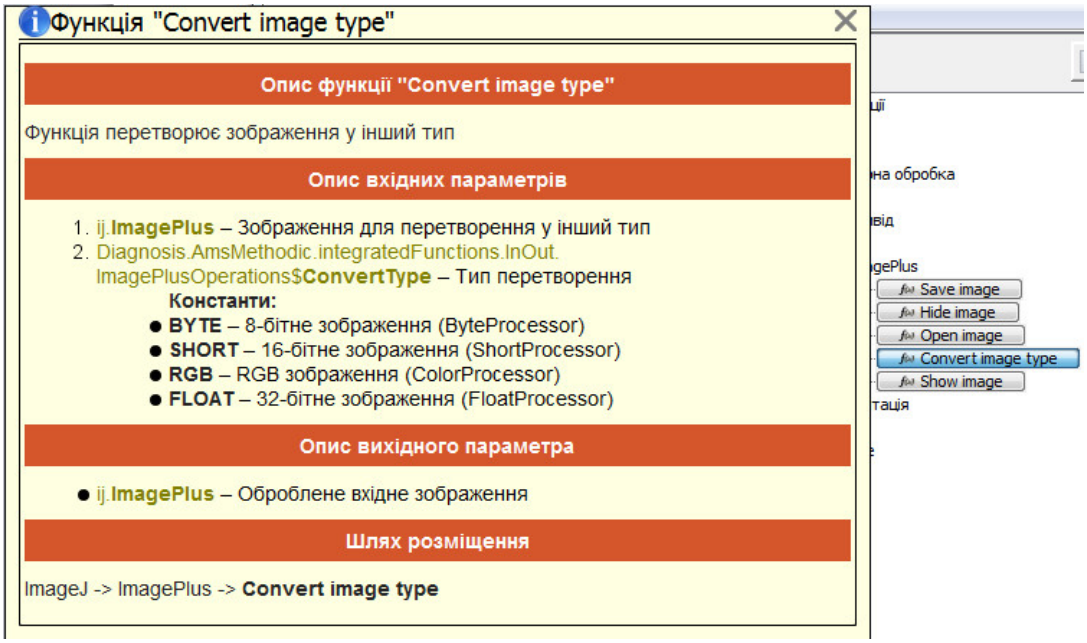


Рисунок 3.20 – Доданий Java метод у список доступних функцій методики

Нижче наведений приклад створення циклу для завантаження по одному зображенні за ітерацію з вказаного каталогу:

```
package Diagnosis.AmsMethodic.integratedFunctions.cycles;

import Diagnosis.AMSUtils;
import Diagnosis.AmsMethodic.functions.CompilerAndInterpreter.Interpreter.MethodicException;
import Diagnosis.AmsMethodic.functions.forUsers.anotations.AAmsFunction;
import Diagnosis.AmsMethodic.functions.forUsers.anotations.AAmsParam;
import ij.ImagePlus;

import javax.imageio.ImageIO;
import javax.swing.*;
import java.io.File;

import java.util.Iterator;

public class ForEachImageInFolder implements Iterable<ImagePlus> {
    private final File[] images;

    @AAmsFunction(
            MethodDescription = "Перебирає в циклі усі зображення, що знаходяться у вказаній папці. Зображення завантажуються по одному за ітерацію",
            MethodNameToDisplay = "For each image in folder",
            returnParamDescription = "Зображення",
            TreePath = "Цикли",
            cycleReturnType = ImagePlus.class
    )
    public ForEachImageInFolder(
            @AAmsParam(description = "Шлях до папки із зображеннями або 'null' для відкриття діалогу вибору папки із зображеннями")
            String pathToFolderWithImages
    )
    {
        if (pathToFolderWithImages == null) {
            JFileChooser fileChooser = new JFileChooser();
            fileChooser.setDialogTitle("Вибір папки із зображеннями");
            fileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
            if (fileChooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
                images = fileChooser.getSelectedFile().listFiles(AMSUtils.getImageFileFilter());
```

```
                }
                else images = new File[0];
            }
            else {
                images                                          =                                                   new
File(pathToFolderWithImages).listFiles(AMSUtils.getImageFileFilter());
            }
        }

        @Override public Iterator<ImagePlus> iterator() {
            return new ForEachImageInFolderIterator();
        }

        private class ForEachImageInFolderIterator implements Iterator<ImagePlus> {

            private int cursor = 0; // index of next element to return

            @Override public boolean hasNext() {
                return cursor < images.length;
            }

            @Override public ImagePlus next() {
                try {
                    return                  new                  ImagePlus(images[cursor].getName()
,ImageIO.read(images[cursor++]));
                } catch (Exception e) {
                    throw new MethodicException(e);
                }
            }

            @Override public void remove() {throw new UnsupportedOperationException();}
        }
    }
```

Результат додання цього користувацького плагіну-циклу наведений на рисунку 3.21.

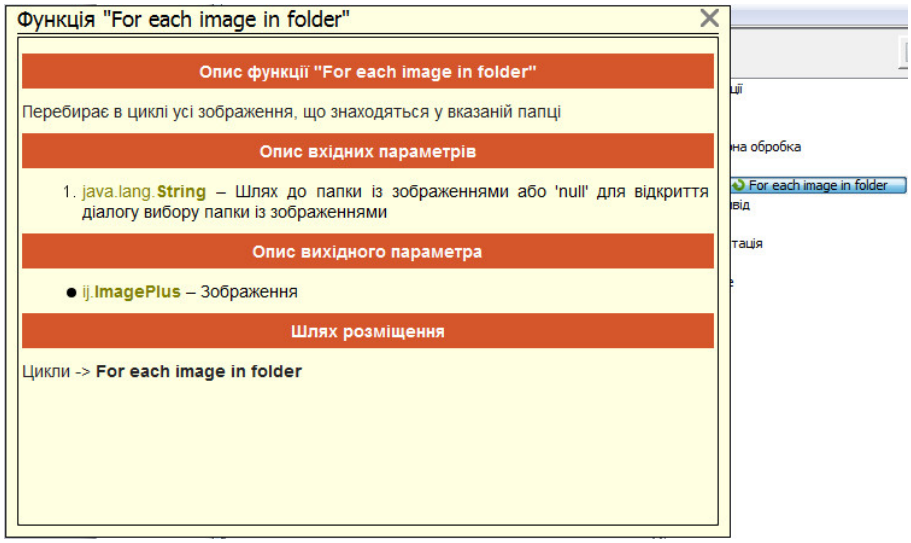


Рисунок 3.21 – Результат додання користувацького циклу у список доступних функцій методики

3.5 База даних

В програмі використовується база даних MySQL для зберігяння даних про дослідження. Також, щоб спростити взаємодію з БД використовується JPA, точніше одна з його реалізацій – Hibernate. Нижче наведено призначення таблиць:

– PatientHbm – таблиця із даними про пацієнта;

– ResearchHbm – таблиця із даними про дослід;

– ImageHbm – таблиця із даними про зображення;

– ImageAttributeHbm – таблиця із даними про атрибути зображення (назва та значення);

– ImageAttributeNameHbm – таблиця, що містить назви атрибутів зображень;

– ImageAttributeValueHbm – таблиця, що містить значення можливі значення для атрибутів зображень;

– ImageMeasurementNameHbm – таблиця, що містить назви числових вимірювань для зображень;

– ImageMeasurementsContainer – таблиця, що містить значення числових вимірювань певної назви для зображень;

– ImageMeasurementsContainer_Measurements – таблиця, що містить значення для вищенаведеного контейнера;

– RoiClassHbm — таблиця, що містить класи об'єктів, які належить зображенню;

– ClassNameHbm – таблиця, що містить назви класів об'єктів;

– ClassName_ClassName – таблиця, що містить ієрархія назв класів об'єктів (надкласи, підкласи);

– RoiGroupHbm – таблиця, що містить групи об'єктів (одна група може мати багато об'єктів, ця група вважається як цілий об'єкт);

– RoiGroupContainer – таблиця, що містить контейнер підкласів об'єктів для надкласу;

– roi_group_container_roi_group – таблиця, що містить виділені об'єкти

для вищенаведеного контейнера;

– RoiGroupMeasurementNameHbm – таблиця, що містить назви числових вимірювань для об'єктів;

– RoiGroupHbm_RoiMeasurementHbms – таблиця, що містить числові вимірювання об'єктів для вищенаведених назв;

– RoiGroupAttributeHbm – таблиця, що містить атрибут об'єкта (назва та значення);

– RoiGroupAttributeNamehbm – таблиця, що містить назви атрибутів об'єктів;

– RoiGroupAttributeValuehbm – таблиця, що містить можливі значення для вищенаведених назв;

– RoiHbm – таблиця, що містить ROI;

На рисунках 3.22 - 3.23 наведена діаграма зв'язків між таблицями БД.

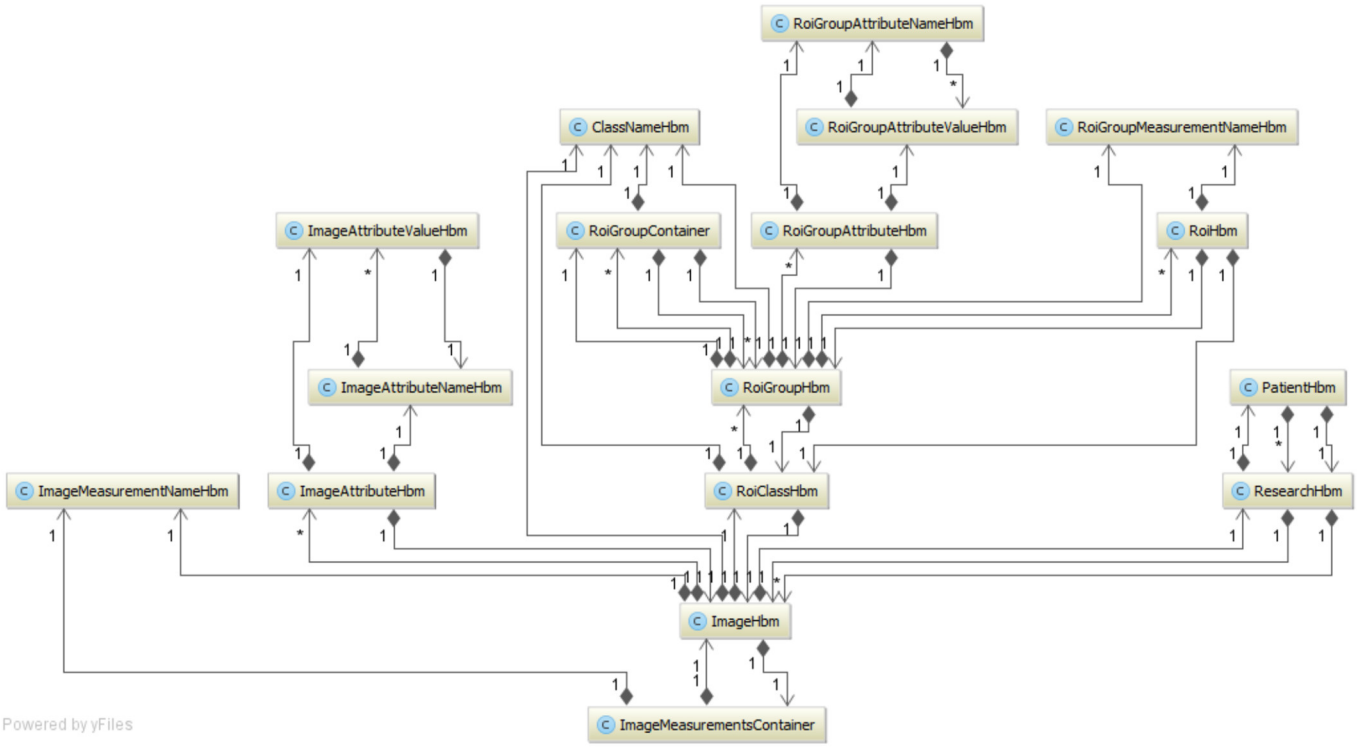


Рисунок 3.22 – Діаграма зв'язків між таблицями БД

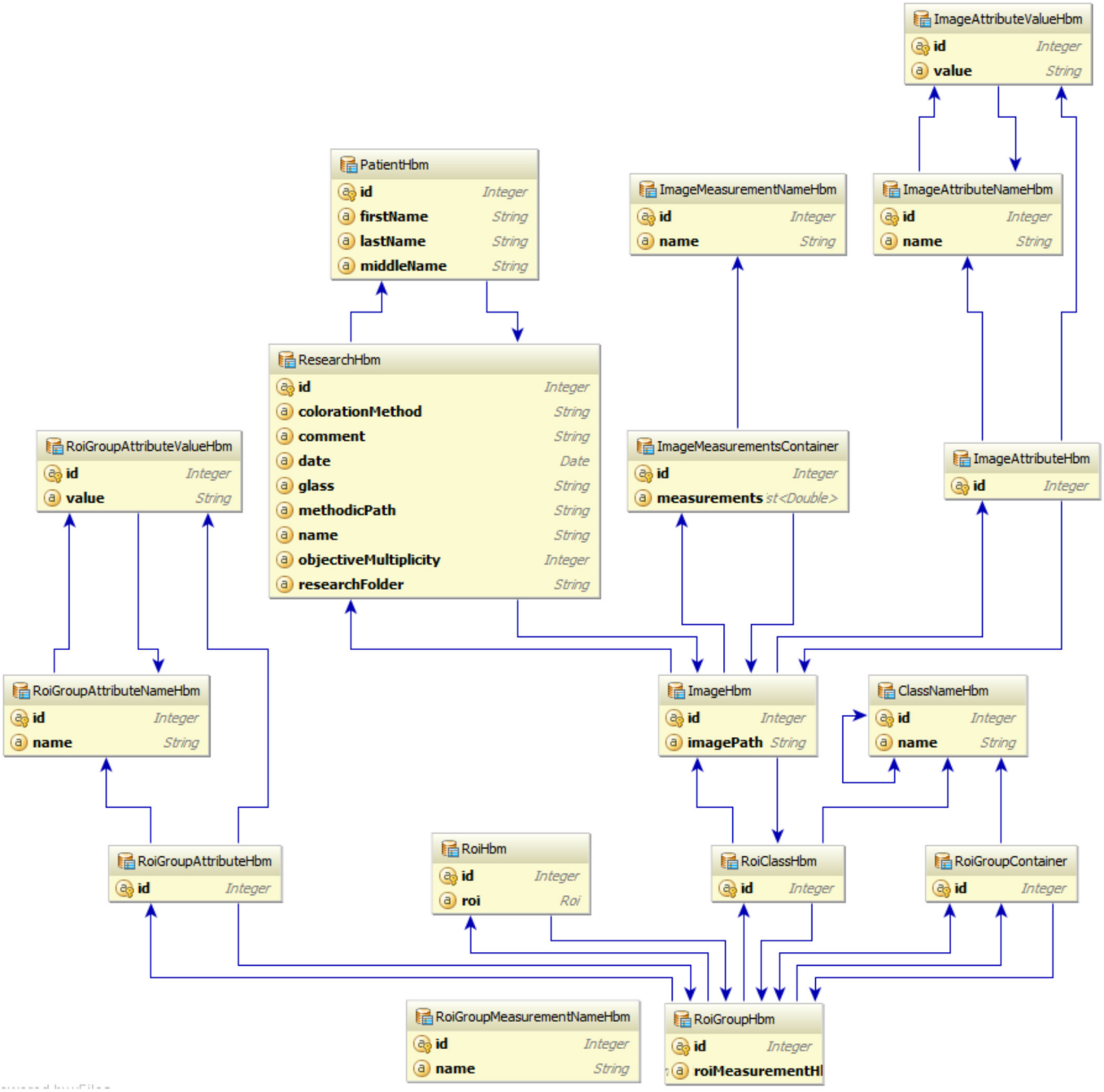


Рисунок 3.23 – Діаграма зв'язків класів сутностей, що представляють БД

Створення дослідження. Після натиснення на кнопку для початку нового дослідження відкривається відповідне вікно для заповнення необхідної інформації (рисунок 3.24). На рисунку 3.25 наведено вікно для вибору методики по якій буде виконуватись дослідження. На рисунках 3.26 - 3.27 наведено інші частини вікна.

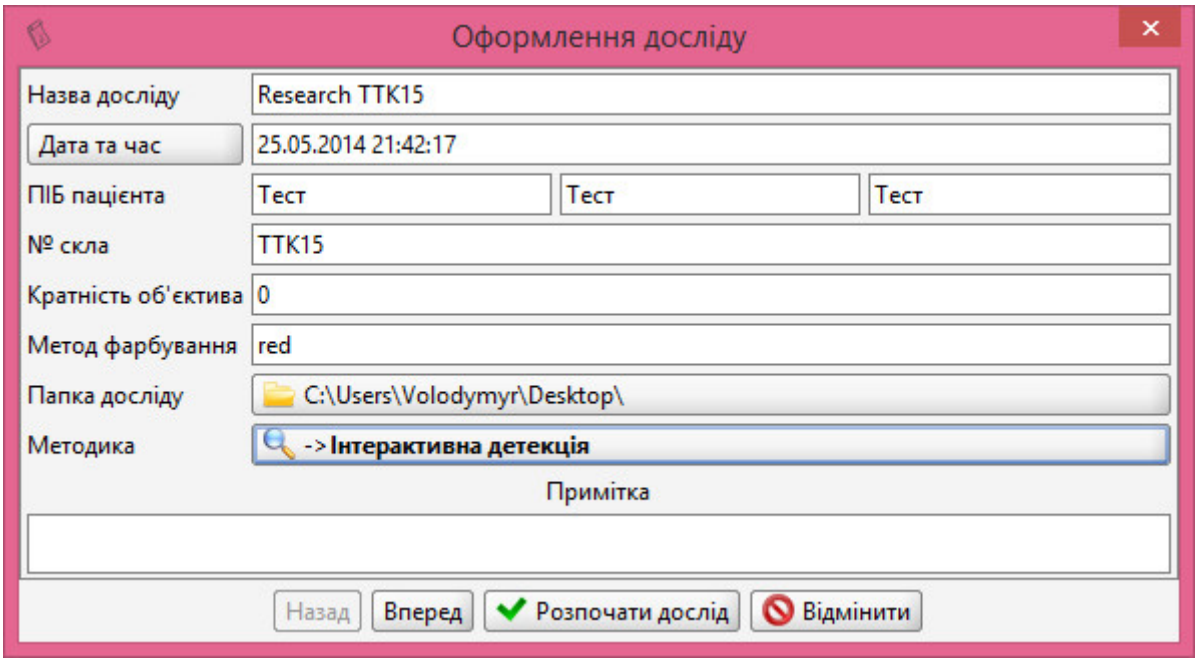


Рисунок 3.24 – Введення загальної інформації про дослідження

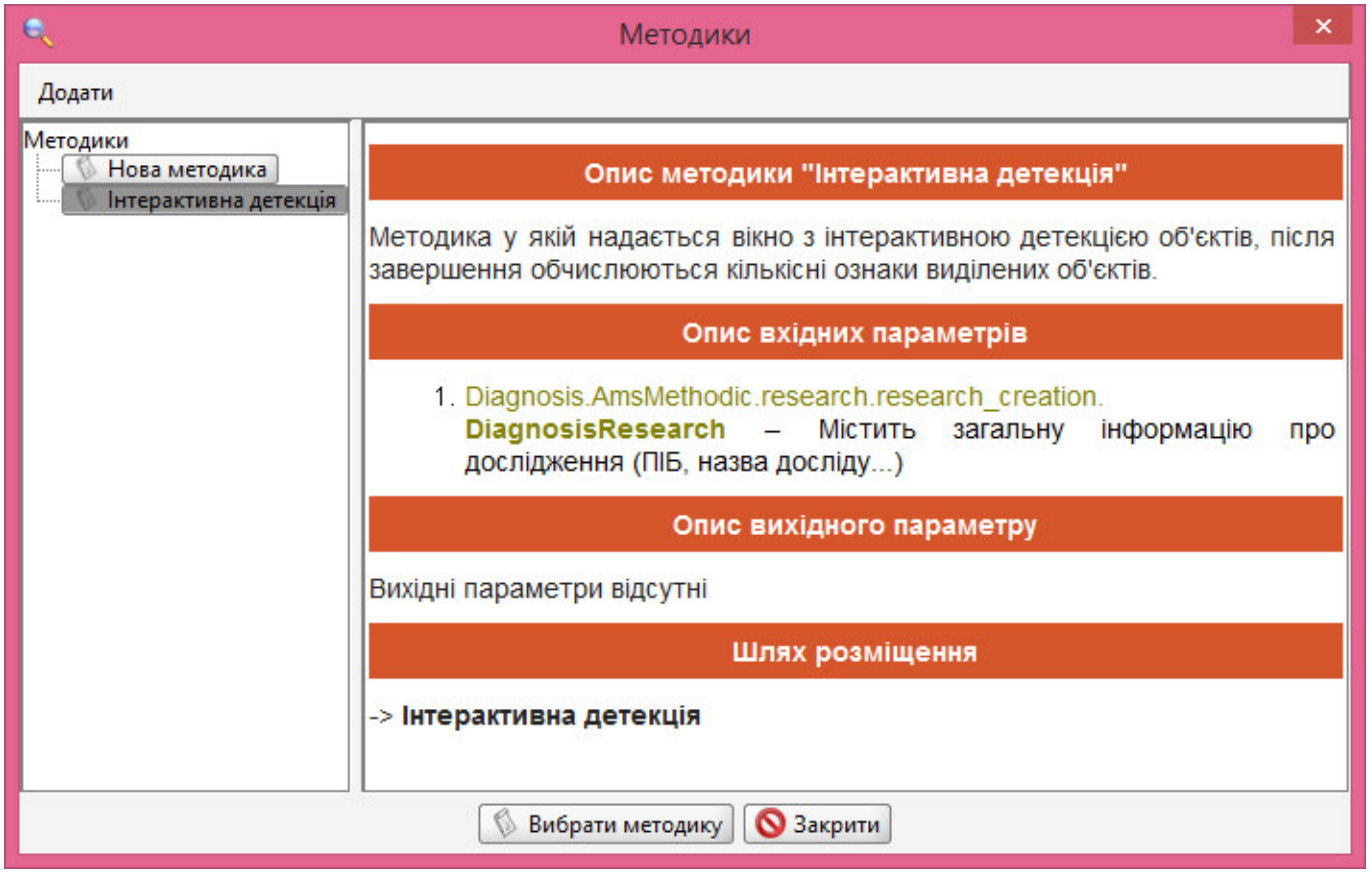


Рисунок 3.25 – Вікно для вибору методики

Рисунок 3.26 – Вибір об’єктів для дослідження

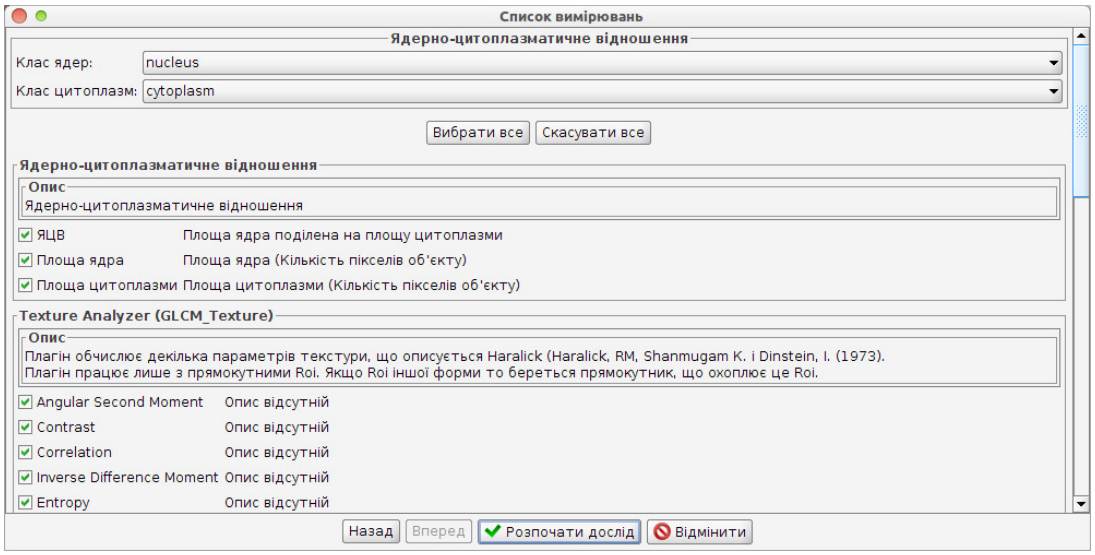


Рисунок 3.27 – Вибір ознак для обчислення

З рисунку 3.24 можуть викликати деякі запитання про поля “Папка досліду” та “Методика”. “Папка досліду” – це директорія з якої будуть вибрані зображення для досліду. “Методика” – це скрипт, що буде викликаний після натиснення на кнопку “Розпочати дослід”.

У вікні, що наведено на рисунку 3.26 потрібно вибрати об’єкти для дослідження. За допомогою кнопки “Добавити” можна додати нову назву класу об’єктів (рисунок 3.28).

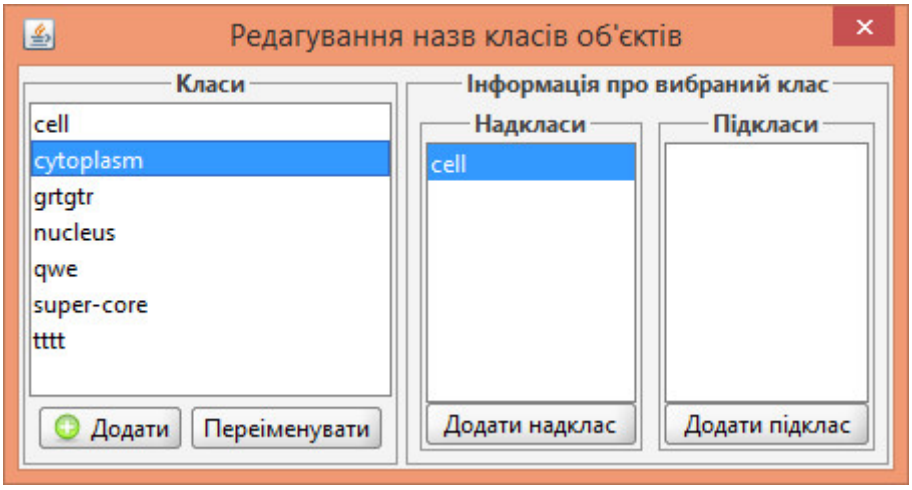


Рисунок 3.28 – Вікно для редагування класів об'єктів

У вікні, що наведено на рисунку 3.27 здійснюється вибір ознак для обчислення. Питання можуть викликати поля “Клас ядер” та “Клас цитоплазми”. У цих полях потрібно вказати клас ядер та цитоплазми, по яких буде обчислюватися ядерно-цитоплазматичне відношення.

Після запуску дослідження запускається вибраний скрипт. В даному випадку для прикладу був вибраний скрипт “Інтерактивна детекція”. Даний скрипт показує користувачеві вікно, що наведено на рисунку 3.29 для ручного виділення об’єктів. Також якщо дослід був перезапущений то за допомогою даного вікна можна відредагувати раніше виділенні об’єкти.

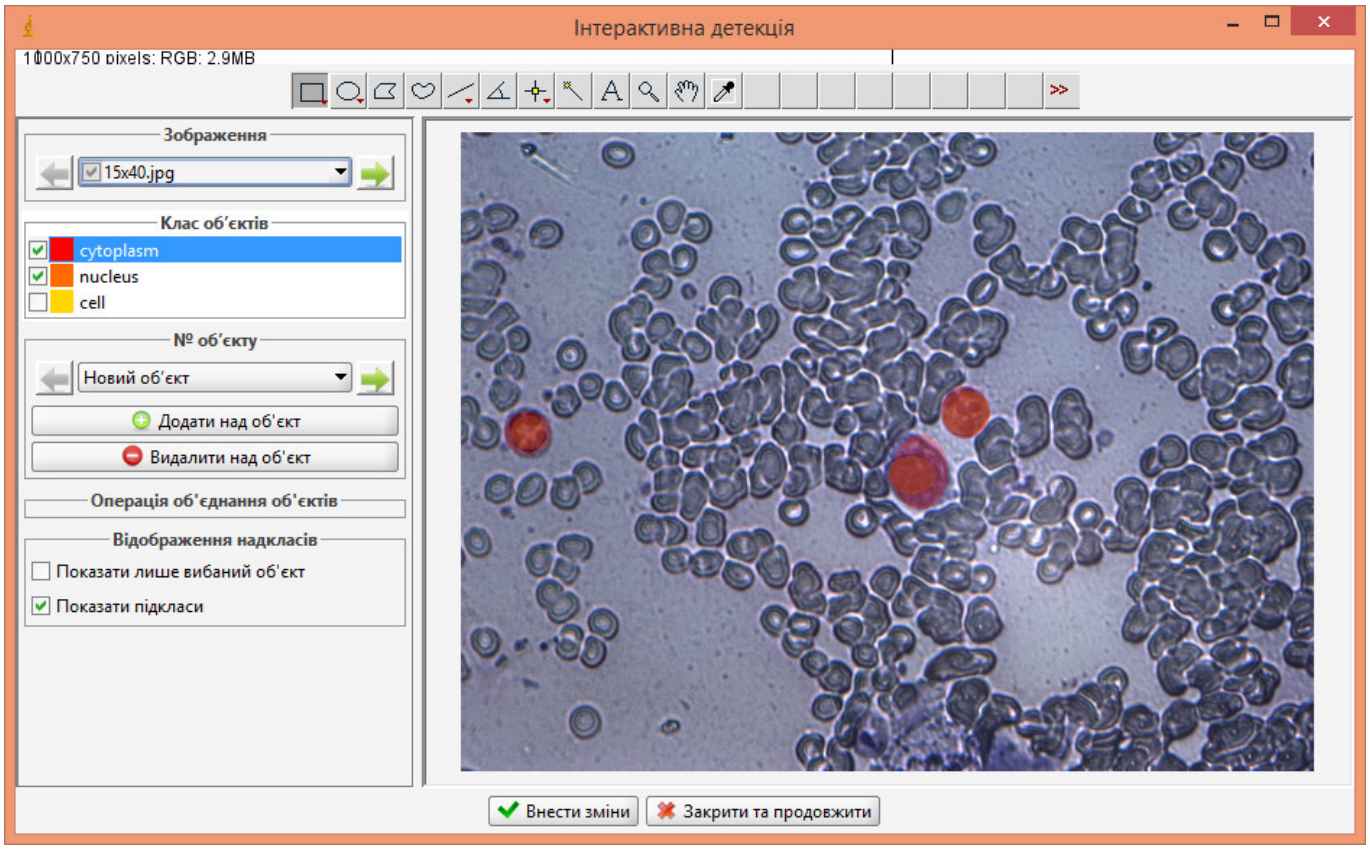


Рисунок 3.29 – Інтерактивна детекція

3.6 Висновки до розділу 3

Отже, в даному розділі спроектовано інтерфейс ImageJ, здійснено налаштування БД, спроектовано функції графічного інтерфейсу користувача, розроблено класи інтерфейсу користувача та спроектована та програмно реалізована база даних.

ВИСНОВКИ

На основі виконання завдання на КР отримано такі результати:

1. Проаналізовано програмну систему ImageJ, розглянуто, структура класів ImageJ, структуру плагінів ImageJ і здійснено аналіз імуногістохімічних зображень.

2. Розроблені алгоритми визначення числових характеристик статистичних даних біомедичних зображень, які лягли в основу побудови системи аналізу біомедичних зображень.

3. Спроектовано інтерфейс ImageJ, здійснено налаштування БД.

4. Спроектовано функції графічного інтерфейсу користувача, розроблено класи інтерфейсу користувача та спроектована та програмно реалізована база даних.

5. Розроблену програмну систему протестовано на прикладі імуногістохімічних зображень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Свідоцтво про реєстрацію авторського права на твір №75359. База даних цифрових гістологічних та цитологічних зображень передракових та ракових станів молочної залози «BPCI2100». / О.М. Березький, Г.М. Мельник, С.О. Вербовий, О.Й. Піцун, В.Д. Николюк, Т.В. Дацко. Дата реєстрації 14.12.2017 р.

2. База даних цифрових гістологічних та цитологічних зображень різних форм раку молочної залози «CIFDB» («CIFDB») / Березький О. М., Мельник Г.М., Николюк В.Д., Дацко Т.В.: Свідоцтво про реєстрацію авторського права на твір № 52743 від 23.12.2013 р.

3. Березький О. М. Розроблення реляційної бази даних інтелектуальної системи автоматизованої мікроскопії / О.М. Березький, О.Й. Піцун, С.О. Вербовий, Т.В. Дацко // Науковий вісник національного лісотехнічного університету України: Збірник науково-технічних праць. - Львів: РВВ НЛТУ України. - 2017. –Т. 27, № 5. - С.125 -129.

4. Березький О. М. База даних цитологічних та гістологічних зображень ауто- та ксеногенних тканин / О. М. Березький, Г.М. Мельник, Т.В. Дацко, С. О. Вербовий // Науковий вісник НЛТУ України: зб. наук.-техн. праць. – Львів: РВВ НЛТУ України. – 2014. – Вип. 24.10. – С. 338-345.

5. Березький О. М. Нечітка база знань інтелектуальної системи діагностування видів раку молочної залози / О. М. Березький, Г. М. Мельник, К. М. Березька // Вісник Хмельницького національного університету. Технічні науки. – 2013. – №6. – С.284-291.

6. Комп'ютерна програма «Інформаційно-аналітична система для дослідження та діагностування пухлинних (ракових) клітин людини «Morphosys» / Березький О. М., Батько Ю.М., Дацко Т.В., Мельник Г.М.: Свідоцтво про реєстрацію авторського права на твір № 35888 від 30.11.2010 р.

7. Berezsky O. Fuzzy system diagnosing of precancerous and cancerous conditions of the breas / O. Berezsky, S. Verbovyy, L. Dubchak, T. Datsko // Proceedings of the XIth International Scientific and Technical Conference Computer Sciences and Information Technologies (CSIT'2016), Lviv, 6-10 September, 2016. – Lviv, 2016. – P. 200–203.

8. Галан В. Ю., Мачуляк М. В., Іпіроті В. О., Николин І. П. Моделювання знань в системах медичної діагностики. Інтелектуальні комп'ютерні системи та мережі : тези доп. V Наук.-практ. конф. молодих вчених і студентів (2 груд. 2021 р.). Тернопіль : ЗУНУ, 2021. С. 13.

9. Мачуляк М. В., Галан В. Ю., Іпіроті В. О., Николин І. П. Системи підтримки прийняття рішень в медичній діагностиці. Інтелектуальні комп'ютерні системи та мережі : тези доп. V Наук.-практ. конф. молодих вчених і студентів (2 груд. 2021 р.). Тернопіль : ЗУНУ, 2021. С. 12.

10. Березький О. М., Дубчак Л. О., Мельник Г. М. Методичні рекомендації до виконання кваліфікаційної роботи з освітнього ступеня "Магістр". Спеціальність: 123 - Комп'ютерна інженерія. Магістерська програма – «Комп'ютерна інженерія». Тернопіль : ЗУНУ, 2021. 32 с.

11. The ImageJ API documentation . URL: http://rsb.info.nih.gov/ij/docs/api/index.html.

12. API documentation and source code. URL: http://rsb.info.nih.gov/ij/download.html.

13. ImageJ plugins (with source code). URL: http://rsb.info.nih.gov/ij/plugins/index.html.

14. Rajwa B, McNally H, Varadharajan P, Sturgis J, Robinson J (2004). «AFM/CLSM data visualization and comparison using an open-source toolkit». Microsc Res Tech 64 (2): 176–84. DOI:10.1002/jemt.20067. PMID 15352089.

15. Gering E, Atkinson C (2004). «A rapid method for counting nucleated erythrocytes on stained blood smears by digital image analysis». J Parasitol 90 (4): 879–81.DOI:10.1645/GE-222R. PMID 15357090.

16. Burger W, Burge M Digital Image Processing: An Algorithmic Approach Using Java. — Springer.

17. Dougherty, G Digital Image Processing for Medical Applications. — Cambridge University Press.

18. Collins TJ (July 2007). «ImageJ for microscopy». BioTechniques 43 (1 Suppl): 25–30.DOI:10.2144/000112517. PMID 17936939.

19. Girish V, Vijayalakshmi A (2004). «Affordable image analysis using NIH Image/ImageJ». Indian J Cancer 41 (1): 47. PMID 15105580.

20. Barboriak D, Padua A, York G, Macfall J (2005). «Creation of DICOM-aware applications using ImageJ». J Digit Imaging 18 (2): 91–9. DOI:10.1007/s10278-004-1879-4. PMID 15827831.

21. Rueden CT, Eliceiri KW (July 2007). «Visualization approaches for multidimensional biological image data». BioTechniques 43 (1 Suppl): 31, 33–6. DOI:10.2144/000112511. PMID 17936940.

22. NIH Image: About.

23. Імуногістохімія: веб-сайт. URL: https://uk.wikipedia.org/wiki/Імуногістохімія.

24. Coons AH, Creech HJ, Jones RN: Immunological properties of an antibody containing a fluorescent group. Proc Soc Exp Biol Med 1941; 47: 200–202.

25. Ramos-Vara, JA (2005). Technical Aspects of Immunohistochemistry. Vet Pathol 42 (4): 405–426.

26. Бухвалов И. Б., Бекер В. Иммуногистохимия: основы и методы. Берлин, Гейдельберг: шпрингер, 2010, 153 с.

27. Иммуногистохимические методы: Руководство / Ed. by George L. Kumar, Lars Rudbeck.: DAKO / Пер. с англ. под ред. Г. А. Франка и П. Г. Малькова. – М.: ЗАО АМТЕО-М, 2011. – 224 с.

28. Эллиниди В.Н., Аникеева Н.В. Иммуногистоцитохимия: теория и практика: учебное пособие. СПб.: Политехника-сервис, 2013. 52 с.

29. Гланц С. Медико-биологическая статистика. Пер. с англ. – М.: Практика, 1998. – 459 с.

30. Гмурман В.Е. Теория вероятностей и математическая статистика: Учеб. пособие для вузов – 9-е изд., стер. – М.: Высшая школа, 2003. – 479 с.

31. Вентцель Е.С. Теория вероятностей. – М.: Наука, 1964. – 576 с.

32. Автандилов Г.Г. Основы количественной патологической анатомии: Учебное пособие. – М.: Медицина, 2002. – 240 с.

33. Автандилов Г.Г. Медицинская морфометрия. Руководство. – М.: Медицина, 1990. – 384 с.

34. David Flanagan. Java in a Nutshell: A Desktop Quick Reference (Java Series) 648 p. 3rd edition 1999) O'Reilly & Associates.

35. David Flanagan. Java Examples in a Nutshell 500 p. 2nd edition (September 2000) O'Reilly & Associates.

36. Java . URL: http://java.

37. Java API documentation and many tutorials are available from Sun Microsystems at. URL: http://java.sun.com/ under “Docs & Training”.

38. Методи, алгоритми і програмні засоби опрацювання біомедичних зображень / Березький О. М., Батько Ю.М., Березька К.М. і ін. Тернопіль: Економічна думка, ТНЕУ, 2017. 330 с.