МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Чернігівська політехніка»

МОДЕЛЮВАННЯ СИСТЕМ

КОНСПЕКТ ЛЕКЦІЙ

для здобувачів вищої освіти
за освітньою програмою "Комп'ютерна інженерія"
(освітній ступінь бакалавр)

Обговорено і рекомендовано
на засіданні кафедри
інформаційних і комп'ютерних систем
Протокол № 5 від 26.01.23

Чернігів 2023

Моделювання систем. Конспект лекцій для здобувачів вищої освіти за освітньою програмою “Комп’ютерна інженерія” (освітній ступінь бакалавр). /Укл.: ”. /Укл.: Бивойно П.Г. - Чернігів: НУ «Чернігівська політехніка», 2023. - 140 с.

Укладач: Бивойно Павло Георгійович, канд. техн. наук, доцент,

Відповідальний за випуск: В.М. Базилевич, зав. кафедрою інформаційних і комп'ютерних систем, к.е.н.

Рецензент: І.В. Білоус, канд. техн. наук, зав. кафедрою інформаційних технологій та програмної інженерії НУ «Чернігівська політехніка».

ЗМІСТ

Вступ

У цьому методичному виданні розглядається цілісний комплекс проблем, що виникають та мають бути вирішені в процесі розробки імітаційної моделі, плануванні та проведенні модельних експериментів, та статистичній обробці результатів цих експериментів. Наведений перелік відповідає вимогам освітньо-професійної програми підготовки бакалаврів спеціальності 123 – „Комп'ютерна інженерія” до дисципліни «Моделювання систем.

Значення дисципліни для реалізації вимог освітньо-професійної програми та вивчення наступних дисциплін полягає в тому, що її вивчення сприяє формуванню наукового рівня інженерного мислення майбутнього фахівця, містить в собі теоретичну базу, яка необхідна при засвоєнні прикладних питань моделювання та вирішуванні конкретних задач дослідження технічних та економічних систем.

Необхідними передумовами для вивчення дисципліни “Моделювання систем" є засвоєння студентами матеріалу дисциплін "Теорія ймовірностей і математична статистика" та "Об'єктно-орієнтоване програмування".

Конспект лекцій призначений для використання при вивченні теоретичного матеріалу дисципліни «Моделювання систем», виконанні лабораторних робіт та розрахунково-графічної роботи.

1 ТЕМА 1. ВСТУП

План лекції

- Предмет курсу.
- Завдання курсу.
- Види моделювання та їхні особливості.
- Особливості моделювання обчислювальних систем.
- Основні етапи моделювання.

1.1 Предмет курсу

Моделювання – це спосіб дослідження явищ, процесів або систем шляхом побудови та аналізу їх моделей.

Модель - це спрощений опис або представлення досліджуваного явища, процесу або системи. Модель може представляти собою креслення, схему, формулу, пристрій, алгоритм і т.д.

Модель простіша, ніж реальна система. При побудові моделі використовують такі засоби як аналогія, абстракція, спрощення. Для оцінки ступеня відповідності між моделлю та системою іноді використовують якісне поняття «ступінь ізоморфізму». Домагатися високого ступеня ізоморфізму зазвичай недоцільно. Важливо, щоб модель забезпечувала досягнення цілей моделювання. У цьому випадку говорять, що модель адекватна досліджуваній системі.

1.2 Завдання курсу

Метою викладення дисципліни ”Моделювання систем” є формування у студентів чіткої системи уявлень про цілісний комплекс проблем, що виникають та мають бути вирішені в процесі розробки імітаційної моделі, плануванні та проведенні модельних експериментів, та статистичній обробці результатів цих експериментів, а також отримання навичок практичного розв’язання цих проблем, що відповідає вимогам кваліфікаційної характеристики фахівця .

Після вивчення дисципліни студенти **повинні знати**:

- про типові моделі стохастичних процесів;
- про існуючи засоби побудови відповідних програмних об'єктів;
- про методи тестування моделей;
- про тенденції розвитку програмних засобів моделювання паралельних процесів;
- про методи планування експериментів з моделями;
- про методи статистичного аналізу отриманих результатів.

У результаті опанування навчальною дисципліною студенти **повинні вміти**:

- проектувати моделі систем масового обслуговування;

– програмувати моделі таких систем ;
– перевіряти адекватність створених моделей;
– планувати та проводити експерименти з моделлю;
– проводити статистичний аналіз отриманих результатів.

Значення дисципліни для реалізації вимог кваліфікаційної характеристики фахівця та вивчення наступних дисциплін полягає в тому, що її вивчення сприяє формуванню наукового рівня інженерного мислення майбутнього фахівця, містить в собі теоретичну базу, яка необхідна при засвоєнні прикладних питань моделювання та вирішуванні конкретних задач дослідження технічних та економічних систем.

Необхідними передумовами для викладання дисципліни "Моделювання систем" є засвоєння студентами матеріалу дисциплін "Теорія ймовірностей і математична статистика" та "Об'єктно-орієнтоване програмування".

1.3 Види моделювання

Моделювання використовується дуже широко, практично у всіх областях людської діяльності, і в кожній області моделі мають свою специфіку. Тому говорити про моделювання «взагалі» досить складно. Більше того, для того щоб побудувати модель і експериментувати з нею, необхідно мати чітке уявлення про саму систему. Наприклад, шляхом моделювання можна вивчати аеродинамічні властивості літальних апаратів. Але при цьому необхідно досить добре знати закони аеродинаміки. Системи автоматичного регулювання можна досліджувати на моделях, побудованих за допомогою пакета MatLab. Але дослідження неможливо провести, не знаючи теорії автоматичного регулювання.

Окрім того сам процес моделювання може відбуватися по різному. Адже це дуже поширений спосіб дослідницької діяльності. Тому не дивно, що існує багато його різновидів. Дя прикладу можна навести такі:

– уявне моделювання, коли дослідник експериментує із системою у своїй уяві;

– фізичне моделювання, коли модель являє собою якийсь предмет, схему, конструкцію. Наприклад дослідження теплових потоків шляхом експериментів з потоками рідини;

– математичне моделювання, коли модель формалізована, наприклад, у вигляді системи рівнянь;

– імітаційне моделювання, коли модель являє собою пристрій або програму, що імітує поведінку реальної системи.

1.4 Моделювання у комп'ютерній інженерії

У нашому курсі ми будемо моделювати системи масового обслуговування. Це можна пояснити тим, що з такими системами ми часто зустрічаємося у комп'ютерній інженерії.

Система масового обслуговування (СМО), або система з чергами (queued

system) - це система, для якої характерна наявність таких компонент:

- випадковий потік заявок (транзакцій) на обслуговування;
- черги заявок, що чекають обслуговування;
- пристрої, що обслуговує ці заявки.

Зі СМО ми зустрічаємося на самих різних рівнях проектування комп'ютерних систем. Наприклад, такою є локальна обчислювальна мережа з сервером і клієнтськими місцями. Багатозадачну операційну систему теж можна розглядати як СМО. Навіть мікропроцесор можна розглядати як СМО, в якій джерелом заявок є шинний інтерфейс, а обслуговуючим приладом - операційний пристрій.

При моделюванні СМО досліднику відомі характеристики окремих компонент системи, а невідомим є результат їх взаємодії. У результаті дослідження найчастіше отримують таку інформацію:

- статистичні характеристики для довжини черг;
- статистичні характеристики для часу очікування в чергах;
- статистичні характеристики для часу обслуговування заявок;
- коефіцієнт завантаження обслуговуючих приладів.

Окрім того метою дослідження може бути отримання залежності наведених вище характеристик від параметрів СМО (кількості обслуговуючих пристроїв, характеристик закону обслуговування і таке інше).

Перелічені вище задачі стосуються сталого режиму роботи СМО, але інколи досліджують і перехідні процеси у СМО.

1.5 Етапи моделювання

Незалежно від виду моделювання, однією з найважливіших умов для початку роботи по створенню моделі є чітке формулювання мети моделювання.

Метою моделювання може бути:

- пошук оптимальних значень параметрів системи;
- визначення статистичних характеристик показників роботи системи;
- вивчення поведінки системи при зміні зовнішніх факторів;
- вивчення поведінки великих груп людей.

Визначившись із метою моделювання можна переходити до вивчення системи, що підлягає дослідженню шляхом моделювання. На цьому етапі необхідно провести декомпозицію системи на окремі компоненти і встановити зв’язки між ними. Детально ці питання розглядаються у курсі «Об’єктно-орієнтований аналіз і проектування». Не слід забувати, про системний підхід до аналізу. Особливу увагу слід приділити зв’язкам системи із зовнішнім світом, та інформації, що надходить до системи зовні.

Наступним етапом є визначення кількісних характеристик компонентів системи, зокрема визначення ймовірносних характеристик потоків заявок та процесів обробки заявок у системі.

Виконавши усю підготовчу роботу, можна починати розробку моделі. Для побудови моделі у наш час існує багато різноманітних засобів, від

універсальних мов програмування до вузькоспеціалізованих пакеті. У нашому курсі ми будемо орієнтуватися на фйремворк Simulation, розроблений на кафедрі спеціально для курсу «Моделювання». Розробка моделі закінчується її всебічним тестуванням. Бажано отримані результати порівняти з відомими даними, що можливо існують для деяких варіантів використання моделі.

Після побудови моделі можна починати моделювання, тобто експерименти з моделлю. На цьому етапі важливо керуватися рекомендаціями теорії планування експериментів. У зв'язку з тим, що результати модельних експериментів зазвичай є випадковими величинами, вони потребують статистичної обробки і визначення характеристик достовірності результатів. Досить часто доводиться проводити дисперсійний та регресійний аналіз отриманих результатів.

КОНТРОЛЬНІ ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Дайте визначення понять «модель», «моделювання».
2. Що студенти повинні знати після вивчення дисципліни.
3. Якими навичками мають володіти студенти у результаті опанування навчальною дисципліною.
4. Назовіть види моделювання та дайте їх характеристику.
5. Що таке «система масового обслуговування».
6. Наведіть приклади систем масового обслуговування у комп'ютерній інженерії.
7. З якою метою може проводитися моделювання.
8. Назвіть основні етапи моделювання.

2 ТЕМА 2. КОРОТКІ ВІДОМОСТІ З ТЕОРІЇ ЙМОВІРНОСТЕЙ ТА МАТЕМАТИЧНОЇ СТАТИСТИКИ

План лекції

– Дискретні та безперервні випадкові величини.
– Функція щільності розподілу ймовірностей.
– Інтегральна функція розподілу.
– Основні числові характеристики випадкових величин.

2.1 Функція щільності розподілу ймовірностей для випадкової дискретної величини

Випадкові величини можуть бути дискретними і безперервними.

Випадкова дискретна величина приймає значення з деякої обмеженого набору. Наприклад, оцінка у системі ESTS (A, B, C, D, E, F), або у національній системі (5, 4, 3, 2).

Значення, які може приймати випадкова дискретна величина, з'являються з деякою частотою. Характеристикою частоти появи деякого значення випадкової дискретної величини є ймовірність появи цього значення. Імовірність - це межа відношення кількості випробувань, в яких відбулося деяка подія до загальної кількості випробувань. Звідси випливає, що сума ймовірностей для всіх можливих значень з набору, дорівнює 1. Наприклад, для результатів іспиту ймовірності появи оцінок можуть бути такими, як показано в таблиці 2.1.

Таблиця 2.1 – Імовірності оцінок на іспиті.

Оцінка	Ймовірність
2	0.1
3	0.4
4	0.3
5	0.2

Прийнято говорити, що така таблиця задає розподіл ймовірностей для випадкової дискретної величини.

Розподіл ймовірностей може бути представлено і графічно, так як це зроблено на рисунку 2.1.

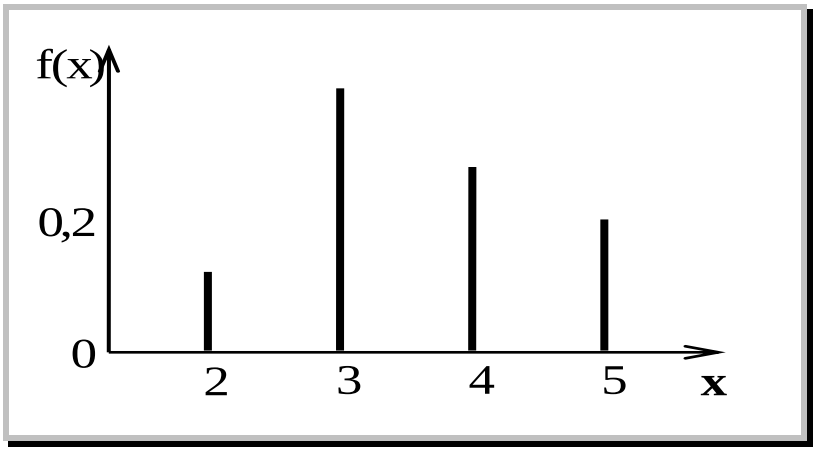

Рисунок 2.1 – Функція щільності розподілу ймовірностей для оцінок на іспиті

2.2 Функція щільності розподілу ймовірностей для випадкової безперервної величини

Випадкові безперервні величини можуть приймати будь-яке значення з деякого інтервалу, хоча межі цього інтервалу часто невідомі. У цьому випадку можна вважати, що значення випадкової неперервної величини теоретично може бути будь-яким. Наприклад, час перебування студента в аудиторії під час іспиту є випадковою безперервною величиною. Але оскільки випадкова величина може приймати будь-яке значення, значить таких значень може бути нескінченно багато, а тому ймовірність кожного з них дорівнює нулю. Не нульовою може бути тільки ймовірність того, що випадкова величина потрапить у деякий інтервал. Наприклад, ймовірність того, що іспит для студента триватиме рівно 0.25 години, дорівнює нулю. Але ймовірність того, що студент пробуде на іспиті від 0.2 до 0.3 години, може бути вже й не рівною нулю.

З цієї причини, коли оцінюють імовірності появи випадкових безперервних величин, говорять про ймовірності попадання цієї випадкової величини в деякий інтервал. Очевидно, що ця ймовірність залежить як від властивостей самої випадкової величини, так і від ширини інтервалу. Ймовірність того, що випадкова величина потрапить в інтервал нескінченної ширини, дорівнює одиниці.

Якщо взяти інтервали однакової ширини, тоді ймовірності попадання випадкової величини в різні інтервали можна порівнювати, а в якості міри порівняння брати висоту прямокутника. Сума висот усіх прямокутників має дорівнювати одиниці.

Одержану таким чином фігуру називають гістограмою, рисунок 2.2. Гістограма зводить випадкову безперервну величину до деякої умовної дискретної величини, значення якої інколи приймають рівним значенню середини інтервалу.

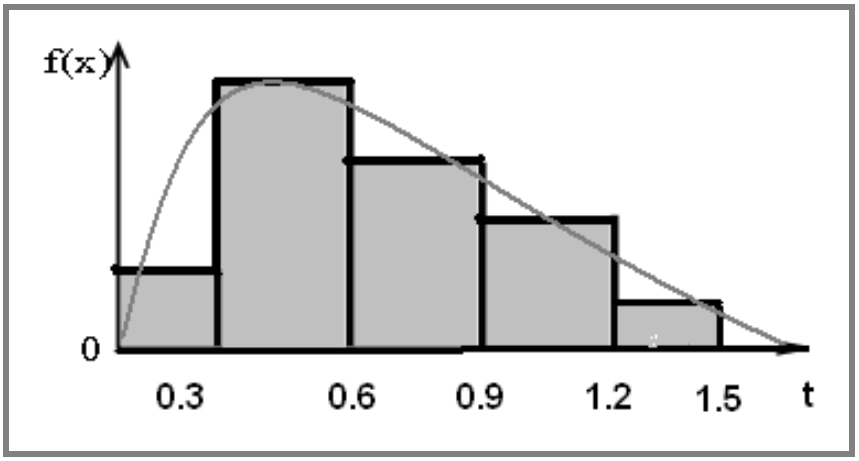

Рисунок 2.2 - Функція щільності розподілу ймовірностей для часу перебування на іспиті

Якщо ширину прямокутників гістограми зменшувати, а кількість їх пропорційно збільшувати, то лінія, що огинає вершини прямокутників буде поступово згладжуватися і, в межі, стане гладкою лінією, а для того, щоб висота одержуваної фігури не перетворилася на нуль, витримують площу фігури рівною одиниці. У цьому випадку ймовірність попадання випадкової величини в деякий інтервал буде дорівнює площі фігури утвореної межами інтервалу і огинаючої.

Отриману таким чином огинаючу лінію називають функцією щільності ймовірностей для випадкової безперервної величини.

Оскільки ймовірність випадкової величини не може бути негативною, то функція щільності ймовірностей не може мати негативних значень.

2.3 Інтегральна функція розподілу ймовірностей для випадкової величини

Поряд з функцією розподілу щільності ймовірностей часто використовується інтегральна функція розподілу ймовірностей, яка утворюється шляхом інтегрування функції щільності ймовірностей по значенню випадкової величини. Оскільки імовірності не можуть бути негативними, а сума ймовірностей дорівнює одиниці, то інтегральна функція - це функція яка не може зменшуватися, у якої мінімальне значення дорівнює 0, а максимальне значення дорівнює 1.

На рисунках 2.3 та 2.4 наведено інтегральні функції розподілу для наведених вище функцій щільності ймовірностей.

Значення інтегральної функції для деякого даного значення випадкової величини дорівнює ймовірності того, що випадкова величина прийме значення, що не перевищує дане значення.

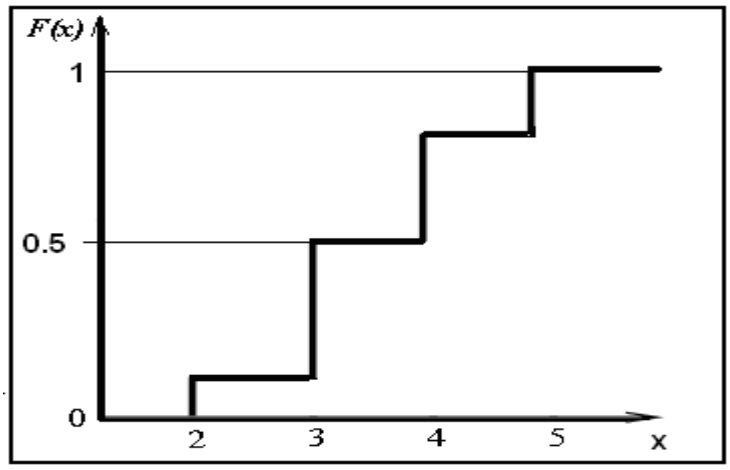

Рисунок 2.3 – Інтегральна функція розподілу ймовірностей для оцінок на іспиті

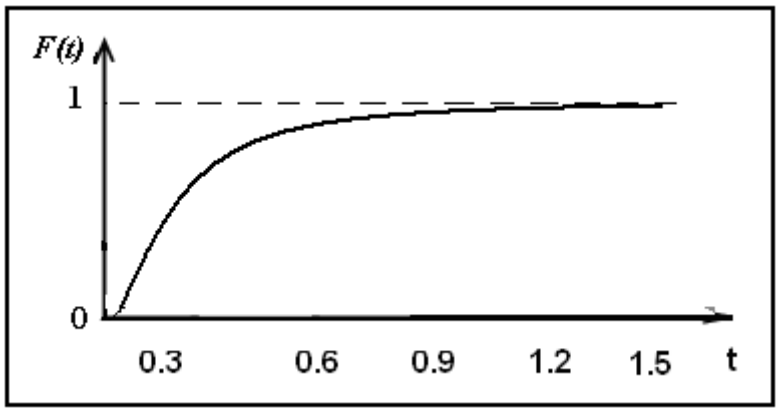

Рисунок 2.4 – Інтегральна функція розподілу ймовірностей для часу, що витрачено на іспит

2.4 Середнє значення та дисперсія випадкової величини

Функції щільності ймовірностей або інтегральна функція розподілу ймовірностей дають найбільш повну характеристику випадкової величини. Але поряд з цими функціями часто використовують числові характеристики випадкової величини – середнє значення і дисперсію.

Для безперервної випадкової величини середнє значення дорівнює математичному очікуванню цієї випадкової величини, і визначається з функції щільності ймовірностей f (x) за формулою 2.1.

$$m = \int_{-\infty}^{+\infty} f(x) \cdot x \cdot dx \qquad (2.1)$$

Для дискретної випадкової величини середнє значення визначається за формулою 2.2

$$m = \sum_{i=1}^{n} x_i \cdot p_i \qquad (2.2)$$

де x – це значення випадкової величини, а p – імовірність цього значення.

За цією ж формулою можна визначити середнє значення і по гістограмі. У цьому випадку x – це значення випадкової величини у середині інтервалу, а p – імовірність того, що випадкова величина потрапить у цей інтервал.

Якщо ж ми маємо просто вибірку випадкових чисел, то середнє значення знаходиться як середнє арифметичне (2.3)

$$m = (\sum_{i=1}^{n} x_i) / n \tag{2.3}$$

Дисперсію D для безперервної випадкової величини можна знайти за формулою 2.4.

$$D = \int_{-\infty}^{\infty} f(x) \cdot (x - m)^2 dx \tag{2.4}$$

Для дискретної випадкової величини дисперсію можна знайти за формулою 2.5. Змінні у цій формулі ті ж самі, що у формулі 2.2.

$$D = \sum_{i=1}^{n} (x_i - m)^2 \cdot p_i \tag{2.5}$$

Формулу 2.5 можна використовувати і для визначення дисперсії по гістограмі. У цьому випадку x – це середина інтервалу.

Якщо ж ми маємо просто вибірку випадкових чисел, то дисперсію можна знайти за формулою 2.6.

$$D = \sum_{i=1}^{n} (x_i - m)^2 / n \tag{2.6}$$

Поряд з дисперсією випадкової величини часто використовують середнє квадратичне відхилення або стандартне відхилення (2.7).

$$\sigma = \sqrt{D} \tag{2.7}$$

2.5 Компоненти, що використовуються для моделювання

Лабораторні роботи та курсовий проект з дисципліни виконуються на основі фреймворку Simulation, створеного на кафедрі ІКС. Фреймворк Simulation містить пакети з класами, що дозволяють створювати моделі систем масового обслуговування та досліджувати їх. Тут ми розглянемо тільки декілька компонентів фреймворку.

2.5.1 *Клас ChooseRandom*

Цей клас визначає візуальний компонент, що забезпечує вибір і налаштування потрібного генератора випадкових чисел. Елементом, які постійно знаходяться на формі є панель з кнопкою і полем для виведення

інформації про вибраний законі розподілу. Вигляд цієї панелі представлений на рисунку 2.5.

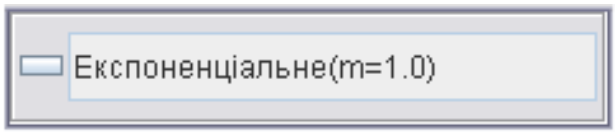

Рисунок 2.5 – Вигляд компонента ChooseRandom

При натисканні на кнопку відкривається діалогова панель, до якої підключено меню «Можливі розподіли». Інтерфейс користувача, який використовується для налаштування експоненціального закону розподілу, представлено на рисунку 2.6.

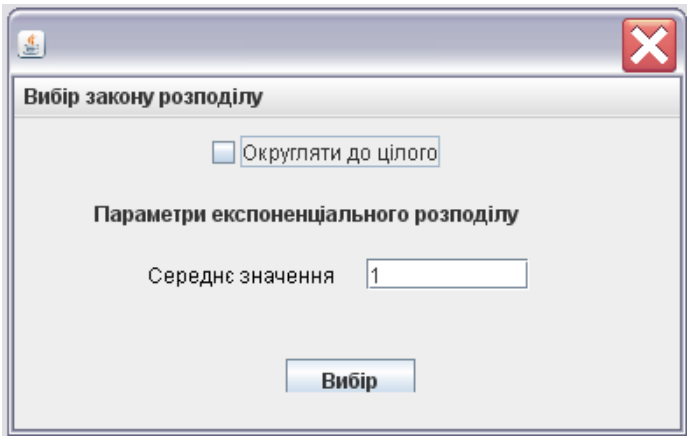

Рисунок 2.6 – Налаштування компоненту ChooseRandom

На діалоговій панелі, одна під одною розташовані невидимі панелі для вибору параметрів різних законів розподілу. Меню робить видимою одну з них. На кожній з цих панелей є кнопка «Вибір», що пов'язана з методом створення відповідного генератора випадкових чисел. Після натискання на кнопку «Вибір», діалогова панель закривається.

Даний компонент реалізують інтерфейс Randomable. Основні методи цього інтерфейсу наступні:

– next() - повертає наступне випадкове число;

– probability(double) - повертає значення інтегральної функції розподілу ймовірностей для заданого параметра.

2.5.2 *Класи Diagram і Painter*

Для графічного відображення результатів моделювання використовуються класи Diagram і Painter.

Клас Diagram представляє візуальний компонент для відображення графіків і діаграм. Вид компоненту класу Diagram показано на рисунку 2.7.

Компонент дозволяє настроювати мінімальні і максимальні значення по осях координат, заголовок діаграми. Можна настроювати координатну сітку. Розміри компоненту можна змінювати.

Нижче наведено деякі методи з класу Diagram.

Рисунок 2.7 – Вигляд компоненту класу Diagram

Методи setGridByX(int), setGridByY(int) використовуються для налаштування масштабної сітки на діаграмі.

Методи setHorizontalMinText(String), setHorizontalMaxText(String), setVerticalMinText(String), setVerticalMaxText(String) використовуються для налаштування діапазону діаграми по горизонтальній та вертикальній висях.

Метод setTitle(String) використовуються для налаштування заголовку діаграми.

Метод clear() очищає діаграму.

Зображення геометричних фігур на діаграмі виконуються за допомогою об'єкту класу Painter, що входить до складу об'єктів класу Diagram. З його допомогою можна зображувати лінії, прямокутники, овали. Колір ліній налаштовується. Особливість цих об'єктів полягає в тому, що після зображення лінії, вони пам'ятають своє становище на діаграмі, що спрощує зображення графіків

Об'єкти класу Painter можуть існувати і незалежно від діаграми, а мати тільки посилання на неї. Завдяки цьому, кілька об'єктів класу Painter можуть використовувати одну й ту ж діаграму, що дозволяє зображати на одній діаграмі кілька графіків.

Метод placeToXY(float, float) використовуються для переміщення пера у точку із заданими відносно діаграми координатами.

Метод drawToXY(float, float) використовуються для проведення лінії у задану точку на діаграмі.

Метод drawDependency(…) можна використовувати для виведення графіків. Як параметри в останній метод передається два масиви, які задають координати точок графіка, колір графіка і параметр логічного типу, який вказує, чи слід підлаштовувати діаграму під дані графіка.

Метод drawBarsDiagram(…) цього класу можна використовувати для виведення стовпчастих діаграм. Як параметри в метод передається масив границь інтервалів по горизонтальній вісі, масив значень по вертикалі, ширина стовпчика в долях ширини інтервалу, зсув стовпчика від лівої межі проміжку і параметр логічного типу, який вказує, чи потрібна зміна налаштування діаграми під дані.

Метод drawNeedleDiagram(…) цього класу можна використовувати для відображення голчастої діаграми. Перелік параметрів цього методу майже такий самий як і у попереднього.

2.5.3 *Класи Histo і DiscretHisto*

Класи Histo і DiscretHisto дозволяють накопичувати інформацію про значення випадкових безперервних і дискретних величин, і подавати інформацію про ці величини у вигляді гістограм. Гістограма може бути представлена у вигляді стовпчастої діаграми або у вигляді таблиці.

Перед використанням гістограма має бути проініціалізована. Найчастіше використовують такі методи ініціалізації:

– init(). Цей метод без параметрів можна використовувати для об'єктів обох класів.

– initFromTo(int, int). Цей метод можна використовувати для об'єктів класу DiscretHisto. Два цілих числа, що передаються в якості параметрів, визначають межі гістограми.

– initFromTo(double, double, int). Цей метод можна використовувати для об'єктів класу Histo. Два дійсних числа, що першими передаються в якості параметрів, визначають межі гістограми, а ціле число визначає кількість інтервалів.

Для занесення даних у гістограми найчастіше використовуються такі методи, що можна використовувати для об'єктів обох класів:

– add(double). Цей метод додає у гістограму число, що передається до методу в якості параметру.

– addFrequencyForValue(double, double). В якості першого параметра задається вага переданого випадкового числа, а другим передається саме випадкове число. У простих випадках вага може дорівнювати 1. Саме так реалізовано метод add(double).

Для відображення результатів обробки накопичених даних можна використовувати такі методи:

– showRelFrec(diagram). Використовується для виведення результатів у вигляді стовпчастої діаграми. До методу як параметр передається посилання на об'єкт класу Diagram, де буде відображатися діаграма. Можлива й інша модифікація методу, з більшим числом параметрів і, відповідно, більшими можливостями.

– toString(). Використовується для виведення результатів обробки накопичених даних у вигляді тексту з таблицею відносних частот.

2.5.4 *Клас ChooseData*

Цей клас успадковує клас JTextField. Поряд із усіма можливостями стандартного текстового компоненту користувач класу ChooseData отримує додаткові послуги. Вигляд компоненту представлено на рисунку 2.8.

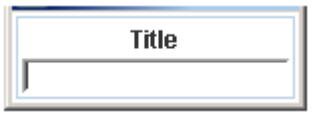

Рисунок 2.8 – Вигляд компоненту класу ChooseData

Метод setTitle(String) дозволяє позначити призначення компоненту.

Методи getInt() та getDouble() дозволяють отримати числа відповідних типів.

Методи getIntArray() та getDoubleArray() дозволяють отримати масиви відповідних типів

Приклад використання компонент пакету зображено на рисунку 2.9.

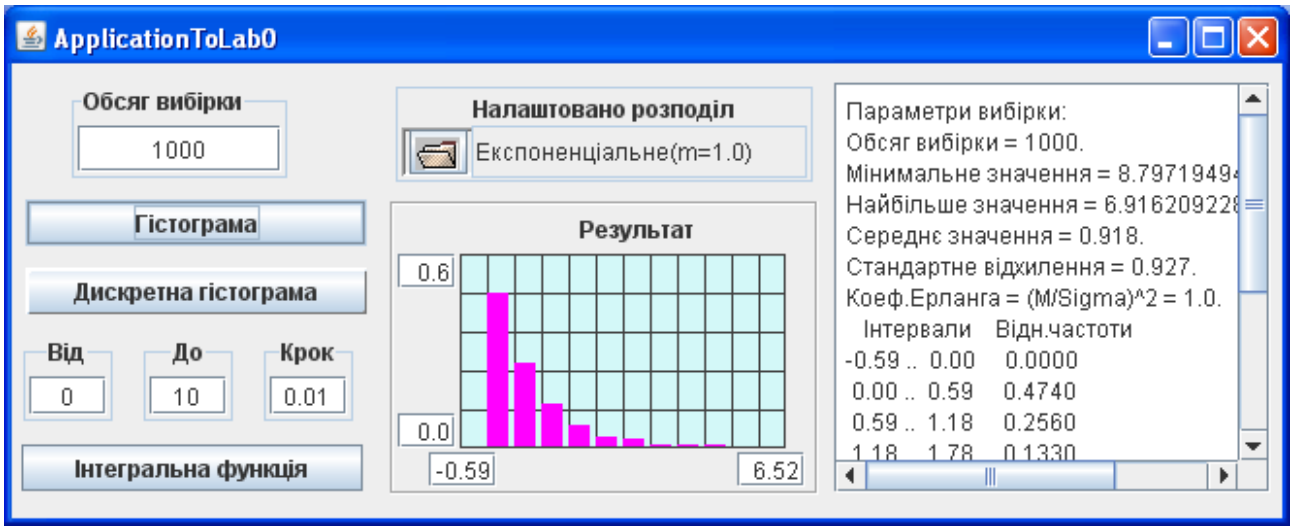

Рисунок 2.9 – Приклад використання компонентів пакету Simulation

КОНТРОЛЬНІ ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Дайте визначення понять «дискретна випадкова величина», «безперевна випадкова величина».
2. Наведіть графічне зображення функції щільності розподілу ймовірностей для дискретної випадкової вличини.
3. Наведіть графічне зображення функції щільності розподілу ймовірностей для безперервної випадкової вличини.
4. Назвіть характерні особливості інтегральної функції розподілу.
5. Як обчислити основні числові характеристики випадкових величин.
6. Як побудувати гістограму?
7. Охарактеризуйте компоненти фреймворку Simulation для роботи з випадковими величинами.

3 ТЕМА 3.ГЕНЕРАТОРИ РІВНОМІРНО РОЗПОДІЛЕНИХ ЧИСЕЛ

План лекції

– Характеристики рівномірного розподілу.
– Способи отримання рівномірно розподілених випадкових чисел.
– Алгоритмічні методи формування рівномірно розподілених випадкових чисел
– Генератор рівномірно розподілених випадкових чисел в Java
– Методи тестування випадкових послідовностей.

3.1 Рівномірний розподіл

Рівномірно розподілені випадкові числа - це випадкові числа, які підпорядковуються рівномірному закону розподілу. Цей закон розподілу має дві характерні риси.

Перша полягає в тому, що випадкові числа можуть приймати значення тільки з деякого інтервалу, що визначається лівою та правою границями інтервалу.

Друга особливість полягає в тому, що щільність ймовірності для всього інтервалу однакова.

На рисунку 3.1 зображена функція щільності ймовірності для випадкової величини, рівномірно розподіленої в діапазоні від ***a*** до ***b***.

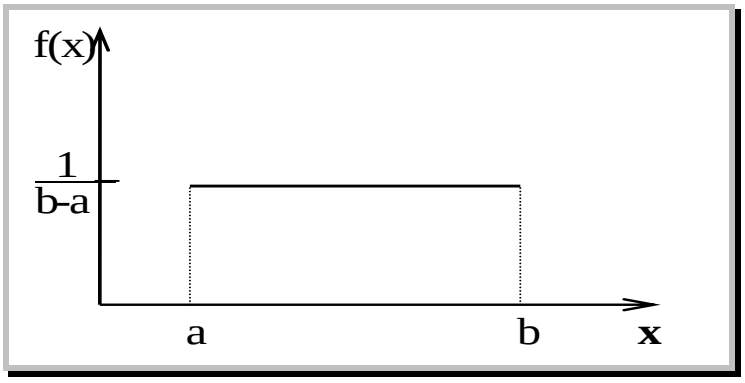

Рисунок 3.1 – Функція щільності ймовірності для рівномірного розподілу

Значення функції щільності ймовірності визначається виходячи з того факту, що площа фігури, що обмежується цією функцією повинна дорівнювати одиниці. Отже, значення щільності ймовірності на інтервалі від ***a*** до ***b*** буде ***1/(b-a)***.

Математичне очікування ***m*** цієї випадкової величини визначається за формулою 3.1.

$$m = \int_a^b \frac{1}{b-a} x \cdot dx = \frac{b^2 - a^2}{2 \cdot (b-a)} = \frac{b+a}{2} \qquad (3.1)$$

Дисперсію D можна знайти за формулою 3.2.

$$D = \int_a^b \frac{1}{b-a} \cdot (x-m)^2 dx = \frac{(b-a)^2}{12} \qquad (3.2)$$

Рівномірно розподілені випадкові числа використовуються для отримання випадкових чисел розподілу, що підпорядковуються іншим законам. При цьому найчастіше розглядаються числа, рівномірно розподілені на інтервалі від 0 до 1. Математичне очікування такої випадкової величини в цьому випадку дорівнює 0.5, а дисперсія 1/12 або 0.083(3).

3.2 Способи отримання РРВЧ

Існують наступні способи одержання РРВЧ – фізичний, табличний і алгоритмічний.

Фізичний метод заснований на перетворенні у число випадкових вихідних сигналів, що виникають під час роботи якогось пристрою, найчастіше електронного або електромеханічного. Отримані в такий спосіб числа дійсно є випадковими, однак такий спосіб досить складний і в цей час використовується рідко.

Табличний метод заснований на використанні спеціальних таблиць, з яких вибирається послідовність РРВЧ.

На рисунку 3.2 наведений фрагмент таблиці випадкових чисел

10 09 73 25 33	76 52 01 35 86	34 67 35 48 76	80 95 90 91 17
37 54 20 48 05	64 89 47 42 96	24 80 52 40 37	20 63 61 04 02
08 42 26 89 53	19 64 50 93 03	23 20 90 25 60	15 95 33 47 64
99 01 90 25 29	09 37 67 07 15	38 31 13 11 65	88 67 67 43 97
12 80 79 99 70	80 15 73 61 47	64 03 23 66 53	98 95 11 68 77

Рисунок 3.2 – Фрагмент таблиці випадкових чисел

Числа із цієї таблиці можна вибирати у будь-якому порядку. Наприклад, якщо потрібні цілі числа не більші 10, то можна брати цифри підряд по горизонталі 1, 0, 0, 9, 7, 3, ..., або вертикалі 1, 3, 0, 9, 1, ...

Якщо потрібні дробові числа, що не перевищують 1, з точністю до чотирьох знаків, то отримаємо 0.1009, 0.7325, 0.3376, 0.5201, ...

Таблиці зручні при ручній роботі з випадковими числами, однак при роботі з комп'ютером, цей спосіб незручний.

Алгоритмічні методи засновані на застосуванні різних обчислювальних процедур. Ці процедури звичайно використовують рекурентні формули, що дозволяють отримувати послідовності випадкових чисел, у яких кожне наступне число визначається попереднім. Початкове число послідовності, іноді називається «зерном». Його потрібно або довільно задати, або воно

автоматично визначається по лічильнику мілісекунд системного таймера. Отримані числа називаються псевдо випадковими тому, що з одного боку послідовності таких чисел задовольняють вимогам тестів на рівномірність розподілу і на випадковість, а з іншого боку залишаються повністю відтворюваними, тому що при тому самому «зерні» формується та сама послідовність випадкових чисел.

3.3 Алгоритмічні методи отримання РРВЧ

3.3.1 *Метод серединних квадратів*

Цей метод, розроблений фон-Нейманом, використовувався на перших комп'ютерах, у середині минулого століття. Суть його полягала в наступному. Ціле число підносили до квадрата і якщо була потреба, доповнювали ліворуч нулями до восьми знаків. Як наступне число брали число, представлене середніми чотирма цифрами отриманого. Наприклад, початкове число 1234. Після піднесення його у квадрат одержуємо 01522756. Середні чотири цифри дають число 5227. Підносячи його у квадрат, одержуємо 27321529. Отже, наступним випадковим числом буде 3215, і так далі.

Цей метод виявився недостатньо ефективним і за якістю отримуваних чисел і за швидкістю.

3.3.2 *Адитивно-мультиплікативний конгруентний метод*

Адитивно-мультиплікативний конгруентний метод (АМКМ) є досить поширеним методом отримання рівномірно розподілених випадкових чисел, хоча і не проходять тести «diehard». Саме цей метод використовується у класі java.util.Random. Тому розглянемо його тут докладніше.

Цей метод заснований на застосуванні рекурентного співвідношення 3.3 для формування псевдо випадкової послідовності цілих чисел.

$$U_{i+1} = (A + B \cdot U_i) \bmod M, \tag{3.3}$$

де U_i, U_{i+1} – попереднє і наступне значення послідовності;

M - модуль;

A - адитивна константа;

B – мультиплікативна константа;

Кожне випадкове число, отримуване за цією формулою, являє собою цілочислову остачу від ділення виразу ($A + B*U$) на модуль M.

Як приклад, отримаємо послідовність випадкових чисел для модуля 10, прийнявши A=9 і B=7.

Нехай зерно дорівнює 0, тобто $U_0=0$;

Тоді $U_1 = (9+7*0) \bmod 10 = 9$;

Зверніть увагу, якщо попереднє число дорівнює 0, то наступне дорівнює адитивній константі.

$U_2 = (9+7*9) \bmod 10 = 2$;

$U_3 = (9+7*2) \bmod 10 = 3;$
$U_4 = (9+7*3) \bmod 10 = 0.$

Як бачимо, числа почали повторюватися. Раніше чи пізніше, але це завжди відбувається. Послідовність неповторюваних цілих чисел не може бути довша, ніж значення модуля.

Якість одержуваної послідовності випадкових чисел залежить від вибору констант *A* і *B,* та модуля.

Вибір модуля впливає на довжину неповторюваної послідовності, що називається періодом. Чим більший модуль, тим більшим може бути період.

Крім того, від вибору модуля залежить швидкість роботи генератора. Якщо вибрати модуль рівний степеню двійки, то операція mod може бути виконана дуже швидко за рахунок використання побітової операції логічного «і». Якщо модуль відповідає розрядності шини даних комп'ютера, то тоді результат обчислення цілочислового виразу (*A+B*U*) буде автоматично обрізатися, і операцію mod взагалі не потрібно буде застосовувати. Саме такі значення модуля найчастіше використовуються, коли продуктивність комп'ютера недостатньо висока.

Якість отримуваної послідовності багато в чому залежить від вибору мультиплікативної константи. Вона повинна бути достатньо велика, інакше на деяких ділянках послідовності числа будуть підпорядковуватися лінійному закону й, отже, не будуть випадковими. Однак значення близькі до модуля теж не можна вважати гарними. Мультиплікативна константа не повинна мати спільних дільників з модулем, щоб уникнути різкого скорочення довжини періоду.

Питання вибору значення мультиплікативної константи добре вивчене для значень модуля, що дорівнюють степеню два. Наприклад, в [2] для вибору значення мультиплікативної константи пропонується формула $B=8p\pm3$, а в [4] $B=2^p+1$, де ***p*** - довільне ціле додатне число.

Адитивна константа - це непарне ціле число більше 0. Вона оберігає генератор від зриву, коли чергове U отримує значення 0.

3.3.3 *Метод "Mother-of-All"*

Метод запропоновано відомим науковцем з проблем генераторів випадкових чисел Джорджем Марсалією.

Обчислювальна формула має вигляд 3.4

$$\begin{cases} S = 21111111111x_{n-4} + 1429x_{n-3} + 1776x_{n-2} + 5115x_{n-1} + c \\ x_n = \frac{S}{2^{32}} \\ C = \left[\frac{S}{2^{32}}\right] \end{cases} \quad (3.4)$$

Початкові значення можна задавати довільні.

Як бачимо, цей метод розширює попередній за рахунок використання попередніх членів послідовності, яких може бути і більше чотирьох. За рахунок цього метод забезпечує дуже великий період і проходить тести «diehard», але

він має невелику швидкість, бо використовує багато операцій множення. Алгоритм може бути застосований в прикладних науках, там де не потрібна велика швидкість.

3.3.4 *Вихор Мерсена*

Метод "Mersenne twister" розроблено в 1997 році японськими науковцями Мацумото і Нішімура. Переваги методу - величезний період (2 ^ 19937-1) і швидка генерація випадкових чисел. Але метод не забезпечує достатню випадковість послідовності чисел. Тому галузь застосування алгоритму дещо обмежена. Незважаючи на це вихор Мерсенна реалізований у стандартних бібліотеках для PHP, Python і Рубі.

3.3.5 *Генератори типу "Xorshift"*

Одні з найновіших (2003) генераторів запропонований Джорджем Марсалією. В даний час це один з найбільш вживаних алгоритмів. У методі розглядається деяка послідовність, до елементів якої застосовуються операції xor та побітові зсуви. Ці операції полягають в наступному:

$$x_n = x_{n-1} \text{ xor } (x_{n-1} \text{ shl } a)$$

Підсумкове випадкове число може бути одержано за допомогою підсумовування окремих членів послідовності, або застосування до них операції xor. Метод достатньо швидкий, бо використовує тільки побітові операції. Послідовність, що генерується, достатньо випадкова, має період 2^128 − 1 та проходить тести Diehard.

Нижче наведено реалізацію даного алгоритму на C:

Листинг 3.1 – Реалізація методу Xorshift

```
unsigned long xor128() {
   static unsigned long x=123456789,
   y=362436069, z=521288629,      w=88675123;
   unsigned long t;
   t=(x^(x<<11));
   x=y; y=z; z=w;
   w=(w^(w>>19))^(t^(t>>8))
   return w;
}
```

3.4 Приведення отримуваних чисел до інтервалу 0..1

Розглянуті вище алгоритмічні методи, як правило, базуються на роботі із цілими числами. Для того щоб звести ці числа до інтервалу 0..1 досить розділити отримувані числа на величину модуля. Якщо модуль досить великий, то дискретністю отримуваних чисел можна знехтувати й вважати, що ми маємо

справу з безперервною, випадковою величиною.

3.5 Реалізація генератора РРВЧ в Java

В Java для одержання РРВЧ використовується описаний вище адитивно-мультиплікативний метод, який реалізований у класі java.util.Random. Цей клас, крім реалізації методів, що забезпечують отримання рівномірно розподілених чисел, дозволяє отримувати випадкові числа, що підпорядковуються розподілу Гауса (нормальний розподіл). Тому в деяких методах виконуються операції, що стосуються обох розподілів. Але ми будемо розглядати тільки те, що стосується рівномірного розподілу.

3.5.1 *Константи АМКМ*

Константи адитивно-мультиплікативного конгруентного методу, які використовуються у класі java.util.Random, задаються змінними класу, які наведені в листингу 3.2.

Листинг 3.2 – Константи класу java.util.Random

```
private final static long multiplier = 0x5DEECE66DL;
private final static long addend = 0xBL;
private final static long mask = (1L << 48) - 1;
private long seed;
```

Тут адитивна й мультиплікативна константи представлені у вигляді цілих чисел шістнадцятого порядку типу **long**. Мультиплікативна константа 0x5DEECE66DL=25214903917. Адитивна константа 0xBL=11. Модулем є число 2^{48} , однак замість модуля в класі зберігається константа 2^{48}-1, що складається з 48 одиниць у молодших розрядах 64-розрядного двійкового коду. Ця константа використовується як маска для отримання остачі від ділення на 2^{48} за допомогою операції логічного побайтового «і».

Для зберігання поточного значення числа у випадковій послідовності використовується приватна змінна **seed**.

3.5.2 *Ініціалізація об'єктів класу java.util.Random*

Початкове значення змінної **seed**, у якій зберігається поточне значення випадкової послідовності, установлюється при виклику конструктора класу. Основний конструктор класу представлений у листингу 3.3.

Листинг 3.3 – Конструктор класу java.util.Random

```
public Random(long seed) {
   setSeed(seed);
}
```

Конструктору передається ціле число, що визначає вид формованої

випадкової послідовності. Це число перетворюється в 48-бітове «зерно» формованої послідовності за допомогою методу **setSeed(long),** текст якого наведений у листингу 3.4.

Листинг 3.4 – Метод створення «зерна» для випадкової послідовності

```
synchronized public void setSeed(long seed) {
    this.seed = (seed ^ multiplier) & mask;
}
```

У цьому методі параметр, переданий конструктору, модифікується за допомогою мультиплікативної константи й логічної побітової операції xor, а потім обрізається до 48 біт шляхом накладення маски.

В разі виклику конструктора без параметрів, текст якого наведений у листингу 3.5, значення змінної **seed** установлюється відповідно до мілісекунд системного таймера.

Листинг 3.5 – Конструктор класу java.util.Random без параметрів

```
public Random() {
   this(System.currentTimeMillis());
}
```

3.5.3 *Реалізація АМКМ*

У класі java.util.Random, АМКМ реалізується в методі **next(int bits).** Цей метод виконує дві функції. Насамперед, він забезпечує формування чергового випадкового цілого двійкового числа, що зберігається в змінній **seed**. Друга функція методу полягає у тому, що він повертає випадкове число із заданою кількістю двійкових біт, шляхом видалення зайвих біт за допомогою зміщення вправо. Необхідна довжина (у бітах) числа, що повертається, передається в метод як параметр.

Таким чином, базова послідовність випадкових чисел складається з 48-розрядних двійкових чисел, а випадкові числа, що повертаються, можуть мати й меншу розрядність. Текст методу **next(int)** наведений у листингу 3.6.

Листинг 3.6 – Метод next(int) класу java.util.Random

```
synchronized protected int next(int bits) {
    long nextseed = (seed * multiplier + addend) & mask;
    seed = nextseed;
    return (int)(nextseed >>> (48 - bits));
}
```

Метод **next(int)** є базовим для отримання рівномірно розподілених чисел різних типів.

3.5.4 *Отримання випадкових чисел різних типів*

Метод **nextInt()** повертає цілі числа типу **int**, у тому числі й від'ємні, рівномірно розподілені по всьому діапазоні цього типу.

Метод **nextLong** повертає випадкове число типу **long**.

Метод **nextInt(int n)** повертає ціле, додатне, рівномірно розподілене число, яке менше ніж число n, що передається до методу як параметр.

Метод **nextFloat()** повертає випадкові числа типу **float**.

Метод **nextDouble()** повертає випадкові числа типу **double**.

Метод **nextBoolean()** повертає значення **true** або **false**.

3.6 Тестування генераторів РРВЧ

Ефективність статистичного моделювання багато в чому залежить від якості використовуваних випадкових чисел, ось чому використовувані генератори випадкових чисел повинні пройти тестування. Найважливіші характеристики послідовностей випадкових чисел це:

- період,
- рівномірність,
- стохостичність,
- незалежність.

Перевірка на період полягає у визначенні довжини послідовності, де числа не повторюються.

Перевірка на рівномірність визначає, чи відповідають числа отриманої послідовності рівномірному розподілу.

Перевірка на стохостичність визначає, наскільки якісно перемішано числа послідовності у межах періоду.

Перевірка на незалежність має підтвердити відсутність взаємозв'язків (кореляції) між числами послідовності.

Відомо багато способів тестування таких генераторів [1,2]. Поряд з найпростішими класичними тестами, такими що будуть розглянуті у лабораторній роботі, існує і багато інших тестів. На сьогодні одним із найбільш суворих наборів тестів для оцінки якості набору випадкових чисел вважаються тести «diehard», розроблені Джорджем Марсалі. Недаремно назва цього набору з 12 тестів перекладається фразеологізмом «міцний горішок». Докладніше з цими тестами ви можете познайомитися в Інтернеті.

3.6.1 *Визначення періоду випадкових чисел*

Ця перевірка полягає у визначенні періоду послідовності, тобто у визначенні довжини послідовності, де числа не повторюються. У зв'язку з тим, що при деяких значеннях констант генератора початкові числа послідовності можуть приймати унікальні значення, які не з'являються пізніше, то при визначенні періоду зазвичай пропускають початкову ділянку, а потім фіксують значення якого-небудь числа й підраховують кількість чисел у послідовності доти, доки чергове число не співпаде із зафіксованим числом.

Для послідовностей, що отримані за допомогою таких методів як "Mother-of-All", д в якості зерен використовується декілька чисел пошук періоду ускладнюється. Тут в унікальних послідовностях числа можуть повторюватися. Для таких послідовностей треба порівнювати не одне число, а декілька послідовних чисел.

3.6.2 *Перевірка на рівномірність*

При перевірці на рівномірність перевіряється гіпотеза про рівномірний розподіл псевдо випадкових чисел. Оцінити рівномірність розподілу можна за допомогою гістограми. Для цього діапазон можливих значень випадкових чисел, наприклад (0,1) розбивається на N рівних частин. Число інтервалів за звичай вибирають пропорційно логарифму від обсягу вибірки. Після цього підраховують скільки разів випадкові числа потрапляли до кожного з інтервалів. Очевидно, що якщо випадкова величина є рівномірно розподіленою, то при великій кількості членів послідовності, значення частот у кожному інтервалі повинні наближатися до величини V/N, де V - число членів послідовності (обсяг вибірки), а N - число інтервалів. Але у зв'язку з тим, що обсяг вибірки не буває нескінченним, завжди існують якісь відхилення дійсних (емпіричних) частот від теоретичних частот.

Розглянемо, наприклад, випадкові числа, що наведені на рисунку 1.2.

З кожної пари чисел можна сформувати дробову частину числа меншого одиниці. Таким чином отримаємо 100 чисел у діапазоні від 0 до 1.

Розіб'ємо весь діапазон на 5 рівних частин і підрахуємо, скільки чисел потрапляє в кожний інтервал. Результати представимо у вигляді таблиці 3.1.

Таблиця 3.1 – Абсолютні емпіричні частоти для інтервалів

Границі інтервалів	0 .. 0.199	0.2 .. 0.399	0.4 .. 0.599	0.6 .. 0.799	0.8 .. 1
Абсолютні частоти	22	19	15	20	24

Теоретична абсолютна частота при рівномірному розподілі однакова для всіх інтервалів (звичайно, якщо інтервали однакової ширини) і визначається як частка від ділення обсягу вибірки на кількість інтервалів. У нашім випадку обсяг вибірки дорівнює 100, а інтервалів 5. То виходить, що теоретична частота дорівнює 20.

Аналізуючи таблицю 3.1 бачимо, що майже в кожному інтервалі спостерігаються відхилення абсолютних частот від теоретичного значення. Виникає питання, чи є ці відхилення випадковими, або дійсно числа розподілені не рівномірно?

Для відповіді на це питання можна скористатися статистичними критеріями перевірки гіпотез про відповідність емпіричних розподілів теоретичним розподілам.

3.6.3 *Перевірка стохастичності*

Стохастичність (випадковість) отримуваної послідовності рівномірно розподілених чисел можна оцінювати різними методами.

Один з них називається методом серій. У цьому методі випадкова послідовність чисел *X* розбивається на елементи двох видів (*A* і *B*) відповідно до умови 3.5.

$$E = \begin{cases} A, \text{ якщо } X < r \\ B, \text{ якщо } X \geq r \end{cases} \tag{3.5}$$

де r довільно задається з інтервалу існування змінної *X*.

Після такої заміни послідовність отримує, наприклад,

такий вид: AAABBAABBBABABAA....

Ділянки цієї послідовності, що складаються з елементів «А», називають серіями. Елемент «В» виступає у ролі границі серії. Якщо два елементи «В» розташовані поруч, то довжина серії між ними дорівнює 0.

У наведеному прикладі маємо серії такої довжини: 3,0,2,0,0,1,1,2...

Таким чином, ми отримуємо нову послідовність випадкових цілих чисел.

Визначимо теоретичні значення імовірності появи кожного з цих чисел, що визначають довжину серії.

Для випадкових чисел, рівномірно розподілених на інтервалі 0..1, ймовірність події *A* дорівнює *r*, а ймовірність події *B* дорівнює 1-*r*.

Таким чином, ймовірність серії довжиною 0 дорівнює ймовірності появи події *B* , тобто $P_0 = 1-r$.

Ймовірність серії довжиною 1 дорівнює ймовірності появи послідовності подій AB, тобто $P_1 = r\,(1-r)$.

Ймовірність серії довжиною 2 дорівнює ймовірності появи послідовності подій AAB, тобто $P_2 = r^2\,(1-r)$.

Ймовірність появи серії довжиною n буде дорівнювати

$$P_n = r^n \cdot (1-r) \tag{3.6}$$

Для перевірки стохастичності генератора необхідно сформувати послідовність випадкових чисел, задатися деяким r, виділити в цій послідовності серії різної довжини, підрахувати їхню кількість і знайти відносні частоти їхньої появи, а потім порівняти їх з теоретичними значеннями.

Звичайно частоти, що знайдені експериментально, не співпадуть з теоретичними ймовірностями внаслідок обмеженості обсягу вибірки. Тому необхідно перевірити, чи випадкові отримані відхилення чи ні. Внаслідок того, що обсяг вибірки в цьому випадку - це сумарна кількість серій, він може вийти невеликим. Тому для перевірки гіпотези про стахастичність послідовності доцільно використовувати критерій Колмогорова-Смирнова, який більш ефективний для малих вибірок, ніж критерій Пірсона.

3.6.4 *Перевірка незалежності*

Перевірка незалежності проводиться шляхом обчислення кореляційного моменту між випадковими величинами даної послідовності, і деякої іншої. Інша послідовність може бути отримана з даної послідовності шляхом її зміщення. Зміщення послідовності можна зробити, якщо відкинути в ній

декілька перших елементів. Наприклад, якщо вихідна послідовність складається із чисел 3 8 5 0 2 5 7 1 3 8 1 ..., то, відкинувши перші 3 елементи, отримаємо послідовність із чисел: 0 2 5 7 1 3 8 1..., яку можна порівнювати з вихідною.

Можна перевіряти на незалежність і послідовності, отримувані за допомогою різних генераторів.

Величина кореляційного моменту k визначається за формулою 3.7,

$$k = \frac{\overline{x \cdot y} - \bar{x} \cdot \bar{y}}{\sigma_x \cdot \sigma_y} \tag{3.7}$$

де $\overline{xy}$ - середнє значення добутку елементів двох послідовностей;

$\bar{x} \cdot \bar{y}$ - добуток середніх значень елементів двох послідовностей;

σ_x, σ_y - середньоквадратичні відхилення випадкових величин.

Кореляційний момент двох незалежних випадкових величин при досить великій довжині вибірки буде прямувати до нуля. Однак внаслідок того, що обсяг вибірки обмежений, значення кореляційного моменту звичайно відрізняється від нуля. Для того щоб визначити, чи випадкове це відхилення, знаходять значення випадкової допоміжної величини T за формулою 3.8.

$$T = \sqrt{\frac{k^2}{1-k^2}(n-2)} \tag{3.8}$$

де n- обсяг вибірки,

k - кореляційний момент.

Відомо, що випадкова величина T повинна задовольняти розподіл Ст'юдента з n-2 ступенями вільності. У статистичних таблицях [3] можна знайти критичні значення для величини T. Фрагмент такої таблиці наведений нижче. Якщо отримане значення критерію T не перевищує критичного значення, то можна вважати, що випадкові величини незалежні.

Таблиця 3.2 – Критичні точки розподілу Ст'юдента для рівня значимості 0.05

Число ступенів волі	Критичне значення
5	2.01
10	1.81
20	1.73
40	1.68

Варто мати на увазі, що при використанні декількох генераторів, залежність між ними може виникнути не через погані налаштування, а внаслідок того, що вони будуть ініціалізовані у межах однієї мілісекунди. У цьому випадку вони будуть формувати однакові послідовності випадкових

чисел, і коефіцієнт кореляції вийде рівним одиниці. Щоб уникнути цього треба перед ініціалізацією кожного генератора робити невелику затримку.

3.7 Статистичні критерії відповідності

3.7.1 *Критерій відповідності Пірсона ("Хі-квадрат").*

У тих випадках, коли обсяг вибірки не менше 100, для перевірки гіпотези про передбачуваний закон розподілу можна скористатися критерієм відповідності Пірсона ("хі-квадрат").

У цьому критерії розбіжність теоретичних і емпіричних абсолютних частот, отриманих для деякої вибірки, оцінюється величиною, що визначається за формулою 3.9.

$$\chi^2 = \sum_{i=1}^{n} \frac{(w_i^{Э} - w_i^{T})^2}{w_i^{T}} \qquad (3.9)$$

де $w^{Э}$, w^{T} - емпіричні і теоретичні абсолютні частоти для інтервалів. Інтервали, для яких значення абсолютної частоти менше 5, поєднують із сусіднім інтервалом

Отримане за формулою 3.9 значення "хі-квадрат" порівнюють із критичним значенням.

"Критичне" значення "Хі-квадрат" визначається із статистичних таблиць відповідно до числа степенів вільності й прийнятим рівнем значимості "альфа".

Число степенів вільності визначається за формулою S-*R*, де *S* - число інтервалів, а *R* - число параметрів розподілу, визначених по вибірці.

Рівень значимості (ймовірність того що, відкидаючи висунуту гіпотезу, ми зробимо помилку) звичайно приймають рівним 0.05.

Якщо отримане значення "Хі-квадрат" менше "критичного", гіпотеза приймається.

Нижче приводяться деякі критичні значення "Хі-квадрат" для рівня значимості "альфа"=0.05

Таблиця 3.3 – Критичні значення "Хі-квадрат" для α=0.05

Число ступенів вільності	Критичне значення
3	7.8
5	11.1
7	14.1
10	18.3

Критичні значення "Хі-квадрат" для інших значень числа степенів вільності й рівнів значимості можна знайти в довідковій літературі [3].

Для приклада розглянутого у пункті 3.6.2 значення «хі-квадрат», знайдене по формулі 3.9, буде наступним.

$$\chi^2 = \frac{(22-20)^2}{20} + \frac{(19-20)^2}{20} + \frac{(15-20)^2}{20} + \frac{(20-20)^2}{20} + \frac{(24-20)^2}{20} = 2.3$$

Число степенів вільності в нашім випадку дорівнює числу інтервалів,

тобто п'яти. Для цього числа знаходимо критичне значення з таблиці 3.3, воно дорівнює 11.1.

Отримане (спостережуване) значення «хі-квадрат», що дорівнює 2.3, менше критичного, котре дорівнює 11.1, отже гіпотезу про рівномірний розподіл випадкових чисел можна прийняти.

3.7.2 *Критерій Колмогорова-Смірнова*

У тих випадках, коли обсяг вибірки невеликий, перевірку статистичних гіпотез про ймовірні закони розподілу рекомендується проводити, використовуючи критерій Колмогорова-Смірнова. У цьому критерії порівнюються **накопичені відносні** значення емпіричних і теоретичних частот, і визначається найбільше відхилення між ними. Отримане значення порівнюють із критичним значенням, що знаходять із таблиць по обсягу вибірки й прийнятому рівню значимості. Якщо отримане відхилення менше критичного, то гіпотезу про відповідність емпіричного й теоретичного розподілів приймають. Для прикладу, перевіримо гіпотезу про рівномірний розподіл для даних, наведених у таблиці 1.1.

Для того щоб перейти від абсолютних частот до відносних частот, потрібно значення абсолютної частоти для кожного інтервалу розділити на обсяг вибірки.

Накопичену частоту для деякого інтервалу знаходять як суму частот всіх попередніх інтервалів і даного.

У таблиці 3.4 наведені результати перерахунку даних з таблиці 3.1.

Таблиця 3.4 – Абсолютні емпіричні частоти для інтервалів

Границі інтервалів	0.. 19	20.. 39	40.. 59	60.. 79	80.. 99
Відносні емпіричні частоти	0.22	0.19	0.15	0.20	0.24
Накопичені відносні емпіричні частоти	0.22	0.41	0.56	0.76	1
Накопичені відносні теоретичні частоти	0.20	0.40	0.60	0.80	1
Відхилення	0.02	0.01	0.04	0.04	0

З таблиці видно, що найбільше відхилення накопичених відносних значень емпіричних частот від відповідних теоретичних значень дорівнює 0.04.

Критичні числа Колмогорова-Смірнова наведені нижче, у таблиці 3.5.

Таблиця 3.5 – Критичні числа Колмогорова-Смірнова для рівня значимості 0.5

Обсяг вибірки	Критичне значення
5	0.565
10	0.410
30	0.240
50	0.207

Для вибірки розміром 100 критичне значення відхилення не перевищує 0.207. Визначене відхилення дорівнює 0.04, тому гіпотезу про рівномірний розподіл можна прийняти.

КОНТРОЛЬНІ ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Дайте характеристику рівномірного розподілу.
2. Які способи можна використовувати для отримання випадкових числ.
3. Наведіть перелік сучасних алгоритмічних методів, та охарактеризуйте їх особливості.
4. Чому послідовності отримані за допомогою алгоритмічних методів називають псевдо випадковими?
5. Навдіть алгоритм отримання послідовності випадкових чисел адитивно-мультипликативним конгруентним методом.
6. Яким вимогам мають відповідати константи АМКМ.
7. Як цілі випадкові числа привести до діапазону 0..1.
8. Охарактеризуйте клас java.util.Random. Які варіанти існують для створення обєктів цього класу. Які методи використовуються для отримання випадкових величин різних типів.
9. Що таке період псевдо випадкової послідовності і як його визначити.
10. Як протестувати випадкову послідовність на рівномірність.
11. Як протестувати випадкову послідовність на стохастичність.
12. Як протестувати дві випадкові послідовності на незалежність.
13. Як перевірити емпіричний розподіл на відповідність ймовірному закону розподілу за критерієм Персона.
14. Як перевірити емпіричний розподіл на відповідність ймовірному закону розподілу за критерієм Колмогорова-Смирнова.
15. Які межі застосування критеріїв Персона та Колмогорова-Смирнова.

4 ТЕМА 4. ГЕНЕРАТОРИ ПСЕВДОВИПАДКОВИХ ЧИСЕЛ ДЛЯ РІЗНИХ ЗАКОНІВ РОЗПОДІЛЕННЯ

План лекції

– Метод оберненої функції.
– Приклади використання метода оберненої функції.
– Спеціальні методи формування випадкових чисел.
– Потоки випадкових подій.

4.1 Метод оберненої функції

Метод оберненої функції є універсальним способом формування випадкових чисел з будь-яким законом розподілення. В цьому методі використовується властивість інтегральної функції розподілення відображати випадкові числа, які підпорядковуються її закону розподілення, у випадкові числа, які рівномірно розподілені в діапазоні від 0 до 1. Зворотне судження також є дійсним. Графічну інтерпретацію цього методу розглянемо на прикладі рівномірного розподілення. На рисунку 4.1 зображений графік інтегральної функції розподілення імовірностей для випадкової величини x, рівномірно розподіленої в діапазоні від **a** до **b.**

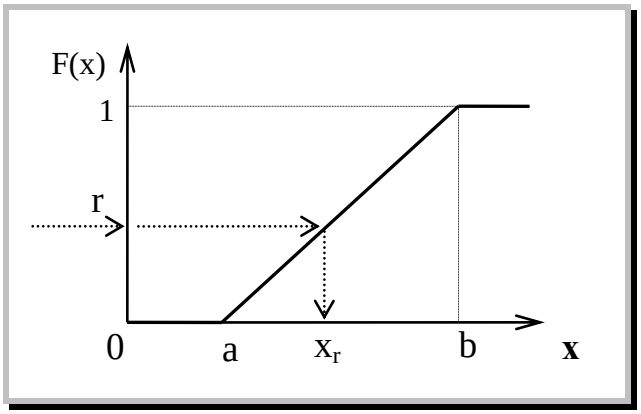

Рисунок 4.1 – Графічна інтерпретація методу оберненої функції для рівномірного розподілення

Якщо отримати деяке випадкове число **r,** рівномірно розподілене в діапазоні від 0 до 1, і відкласти його на осі ординат графіка інтегральної функції розподілення, то відповідні йому числа на осі абсцис будуть підпорядковуватись закону, який задається цією інтегральною функцією. Тобто обернена функція перетворює **r** в $\mathbf{x_r}$.

Для рівномірно розподілених чисел зв'язок між **r** та $\mathbf{x_r}$ може бути заданий аналітично, у вигляді формули 4.1, яку можна одержати, якщо розглянути подібні трикутники на рисунку 4.1.

$$x_r = a + r * (b - a) \tag{4.1}$$

де r – рівномірно розподілене від 0 до 1 випадкове число.

Саме ця формула використовується для генерації рівномірно розподілених в заданому діапазоні випадкових чисел у класі Uniform, який входить до складу пакета rnd.

4.1.1 *Генерація довільного закону розподілення.*

При використанні методу оберненої функції для отримання випадкових чисел з довільним законом розподілення, інтегральна функція для цього закону розподілення представляється відрізками прямих ліній (кусково-лінійна апроксимація). В результаті такої заміни вона може бути задана за допомогою двох масивів - X і F, що задають координати стиковки відрізків.

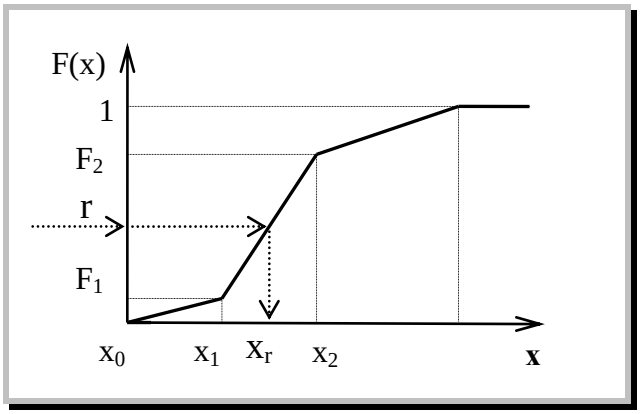

Рисунок 4.2 - Кусково-лінійна апроксимація інтегральної функції розподілення

Алгоритм формування випадкового числа такий:

- генерується рівномірно розподілене від 0 до 1 випадкове число r;
- визначаємо границі інтервалу на осі ординат, в який потрапляє це випадкове число, F1 та F2;
- визначаємо границі відповідного інтервалу на осі абсцис x1, x2;
- визначаємо кутовий коефіцієнт оберненої прямої k = (x2 - x1) / (F2 - F1);
- обчислюємо випадкове число x_r = x1 + k*(r - F1).

Для генерації таких чисел в пакеті rnd знаходиться клас Linear.

4.1.2 *Дискретне розподілення*

Дискретне розподілення звичайно задається масивом можливих значень випадкової величини і масивом ймовірностей цих значень. Наприклад, оцінки на екзамені можуть приймати значення 2, 3, 4, 5, а імовірності їх появи 0.1, 0.4, 0.3, 0.2.

Графічне зображення такого розподілення представлено на рисунку 4.3.

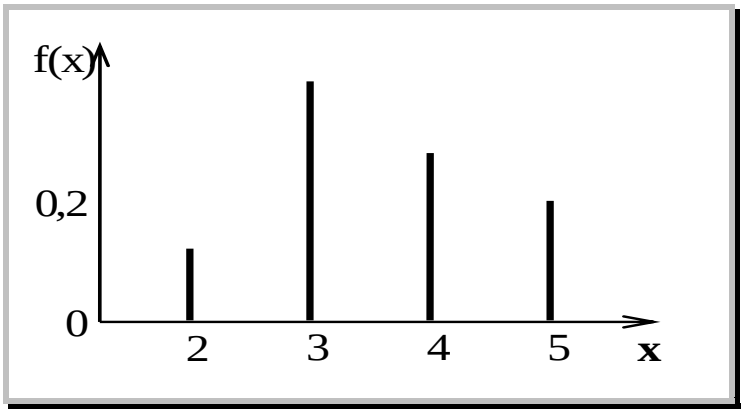

Рисунок 4.3 – Функція щільності розподілення ймовірностей
для дискретного закону розподілення

Для генерації дискретних, випадкових чисел також можна використовувати метод оберненої функції. Для цього необхідно сформувати масив накоплених значень ймовірностей. В наведеному вище прикладі з оцінками на екзамені він буде таким: 0.1, 0.5, 0.8, 1.0. Цей масив задає значення інтегральної функції розподілення в точках, які відповідають можливим значенням випадкової величини. Графік інтегральної функції розподілення для дискретної, випадкової величини має ступінчатий характер, рисунок 4.4..

Внаслідок цього випадкові величини, отримані методом оберненої функції, можуть приймати лише дискретні значення.

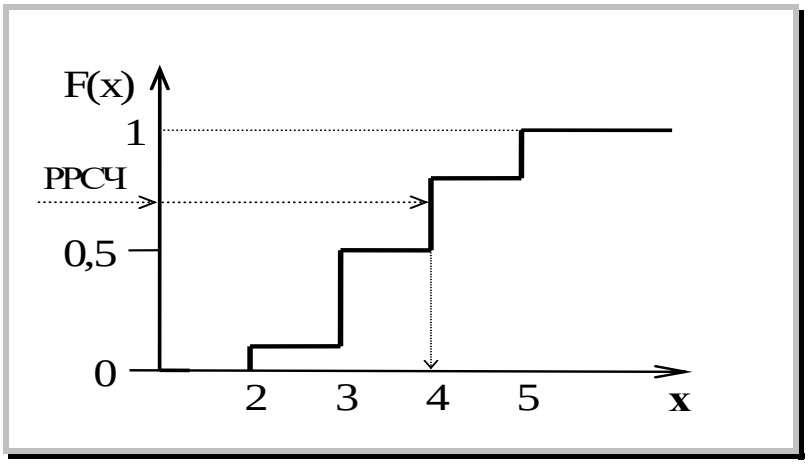

Рисунок 4.4 – Використання методу оберненої функції
для дискретного закону розподілення

Описаний метод генерації випадкових дискретних чисел реалізований у класі Discret пакету rnd.

4.1.3 *Експоненціальний закон розподілення*

Метод оберненої функції можна використовувати не лише в графічній інтерпретації, але й аналітично. Однак для цього необхідно отримати аналітичний вираз оберненої функції. Інколи це можливо, наприклад, для

експоненціального закону розподілення, графік функції щільності ймовірностей якого представлений на рисунку 4.5.

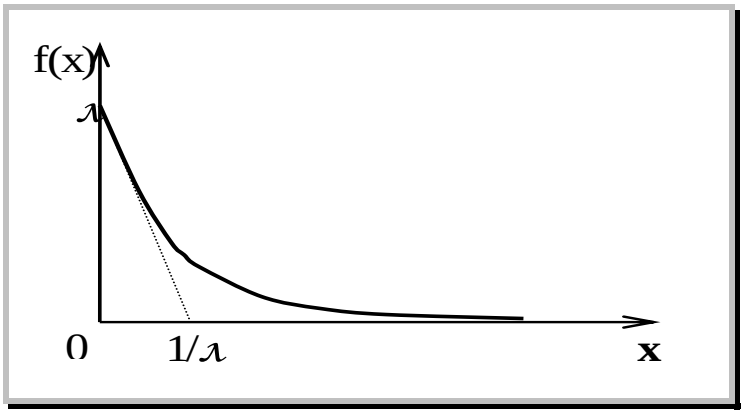

Рисунок 4.5 – Функція щільності ймовірностей для експоненціального розподілення

Аналітичний вираз для цієї функції має вигляд 4.2.

$$f(x) = \lambda e^{-\lambda x}, \tag{4.2}$$

де λ - параметр розподілення, який називається інтенсивністю, величина обернена середньому значенню випадкової величини.

Інтегральна функція розподілення для цього закону описується формулою 4.3.

$$F(x) = 1 - e^{-\lambda x}, \tag{4.3}$$

Для функції F(x) можна знайти обернену функцію, тобто існує формула для обчислення x.

$$\begin{aligned} & 1 - F(x) = e^{-\lambda x}; \\ & -\lambda x = \ln(1 - F(x)); \\ & x = -\frac{1}{\lambda}\ln(1 - F(x)). \end{aligned} \tag{4.4}$$

Використовуючи метод оберненої функції, в якості F(x) ми беремо рівномірно розподілені в діапазоні від 0 до 1 випадкові числа, тому величина 1-F(x) також буде випадковою й рівномірно розподіленою в діапазоні від 0 до 1. Отже, для обчислення випадкових чисел, які підпорядковуються експоненціальному закону розподілення можна використовувати формулу 4.5.

$$x = -(1/\lambda)\ln r, \tag{4.5}$$

де r – РРВЧ в діапазоні [0;1].

Для генерації таких чисел в пакеті rnd міститься клас Negexp.

4.2 Спеціальні методи формування випадкових чисел

Метод оберненої функції є досить ефективним у графічній інтерпретації а також тоді, коли є аналітичний вираз для оберненої функції. Але для багатьох законів розподілу отримати такий вираз не вдається. У таких випадках доводиться використовувати методи, спеціально розроблені для конкретних законів розподілу. Нижче розглядаються деякі із них.

4.2.1 *Формування випадкових чисел, які підпорядковуються закону Ерланга*

Графічне зображення функції розподілення щільності ймовірностей для закону Ерланга представлено на рисунку 4.6. Це розподілення займає проміжне місце між нормальним розподіленням і експоненціальним. При великих значеннях коефіцієнту **k** розподілення наближається до нормального розподілення, а при **k=1** співпадає з експоненціальним.

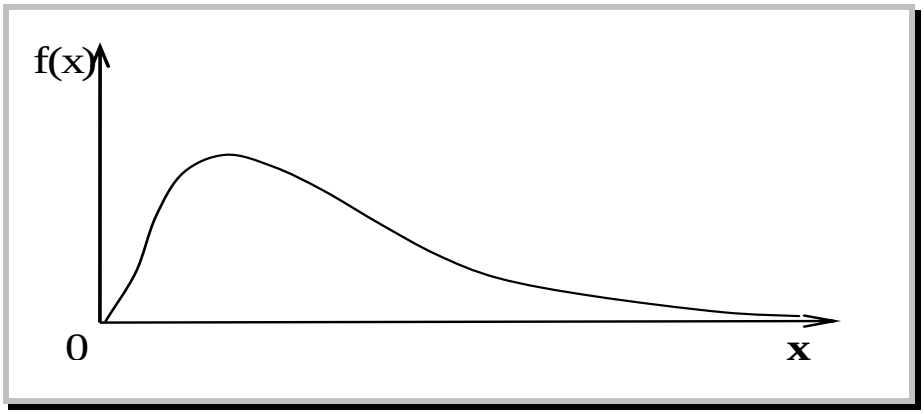

Рисунок 4.6 – Функція щільності ймовірностей для розподілення Ерланга

Інтегральна функція для розподілення Ерланга має два параметри: інтенсивність **λ** - це величина обернена середньому значенню випадкової величини, і коефіцієнт **k**, який називається степенем післядії, або коефіцієнтом Ерланга. Ця функція описується досить складним виразом 4.6. і наводиться тут тільки для того, щоб продемонструвати її складність.

$$F(x) = 1 - e^{-\lambda kx} \sum_{i=0}^{k-1} \frac{(\lambda kx)^i}{i!} \qquad (4.6)$$

З наведеної формули отримати вираз для обчислення x важко. Тому для формування чисел, які підпорядковуються закону Ерланга, використовують властивість експоненціального розподілення випадкових чисел, яка полягає в тому, що сума **k** експоненціально розподілених випадкових чисел з інтенсивністю λ є випадковим числом Ерланга з коефіцієнтом **k** та інтенсивністю λ/k. Отже, для того щоб отримати число Ерланга з інтенсивністю λ**э** і коефіцієнтом **k,** необхідно просумувати **k** експоненціально розподілених випадкових чисел, що мають інтенсивність kλэ.

$$x_э = \sum_{i=1}^{k}(-\frac{1}{k\lambda_э}\ln r_i) = -\frac{1}{k\lambda_э}\sum_{i=1}^{k}(\ln r_i) = -\frac{1}{k\lambda_э}\ln\prod_{i=1}^{k} r_i \qquad (4.7)$$

Для генерації таких чисел в пакеті rnd знаходиться клас Erlang.

4.2.2 *Формування випадкових чисел, які підпорядковуються нормальному закону розподілення*

Нормальне розподілення є одним з найбільш поширених розподілень для випадкової величини.

Функція щільності імовірності для цього розподілення визначається виразом 4.8.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(x-m)^2}{2\sigma}} \qquad (4.8)$$

де m – середнє значення випадкової величини;
σ - середньоквадратичне відхилення.

Графічне зображення функції щільності імовірності для цього закону представлено на рисунку 4.7.

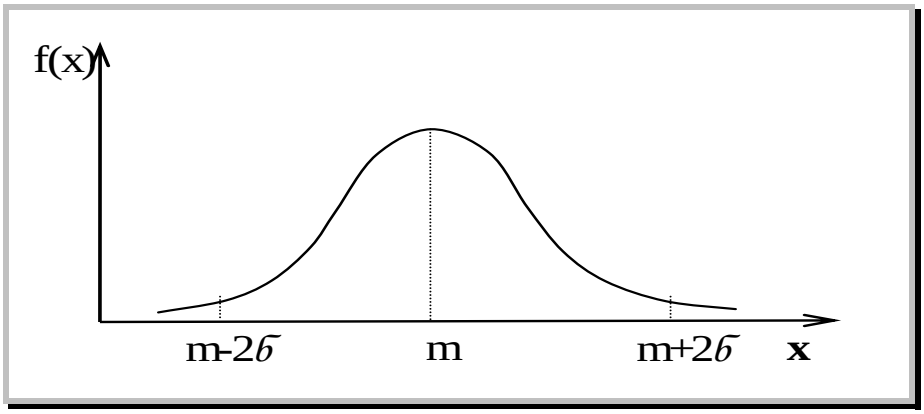

Рисунок 4.7 – Функція щільності ймовірностей для нормального розподілення

Значення інтегральної функції розподілення ми можемо представити лише як інтеграл від функції щільності ймовірностей, але цей інтеграл не береться. Тому знайти x = f(F(x)) не можна і, отже, метод оберненої функції тут не використовується.

Але для формування нормально розподілених випадкових чисел можна використовувати центральну граничну теорему теорії ймовірностей (теорему Ляпунова), яка формулюється наступним чином. *Якщо випадкова величина X є сумою великої кількості взаємно незалежних випадкових величин, вплив кожної з яких на всю суму незначний, то X має розподілення, близьке до нормального розподілення.*

Для отримання взаємно незалежних випадкових величин, які мало впливають на суму, можна використовувати генератор РРВЧ і тоді формула для отримання нормально розподілених чисел X матиме вигляд:

$$X = \sum_{i=1}^{N} r_i \tag{4.9}$$

де r_i – рівномірно розподілене випадкове число в діапазоні від 0 до 1.

Математичне очікування **m** цього числа дорівнює сумі математичних очікувань доданків. А так як математичне очікування випадкових чисел, рівномірно розподілених в діапазоні від 0 до 1 дорівнює 0.5, то математичне очікування випадкової величини X, отриманої по формулі 4.9, можна знайти за формулою 4.10.

$$m = 0.5N \tag{4.10}$$

Дисперсія випадкових чисел, рівномірно розподілених в діапазоні від 0 до 1, дорівнює 1/12. А виходячи з того, що дисперсія суми випадкових чисел дорівнює сумі дисперсій доданків, можемо отримати дисперсію сформованої випадкової величини X за формулою 2.11.

$$D_X = \frac{1}{12} \cdot N \tag{4.11}$$

Середньоквадратичне відхилення випадкової величини X можна знайти за формулою 4.12.

$$\sigma_X = \sqrt{N/12} \tag{4.12}$$

Для того, щоб нормувати отримане число (тобто зробити його математичне очікування рівним нулю і дисперсію рівну 1) потрібно від числа відняти його математичне очікування і поділити на середньоквадратичне відхилення. Формула 4.13 дозволяє отримати нормоване нормально розподілене випадкове число Z.

$$Z = ((\sum_{i=1}^{N} r_i) - 0.5N) / \sqrt{N/12} \tag{4.13}$$

З практики відомо, що числа, які отримані по цій формулі, мають розподілення близьке до нормального вже при N > 6. Тому N можна взяти рівним 12, тоді формула для отримання нормованих нормально розподілених випадкових чисел спрощується.

$$Z = ((\sum_{i=1}^{12} r_i) - 6) \tag{4.14}$$

Для отримання нормально розподіленого випадкового числа із заданими параметрами – математичним очікуванням **m** і середньоквадратичним відхиленням **σ** достатньо нормоване випадкове число **z** помножити на потрібне

середньоквадратичне відхилення і додати потрібне математичне очікування. Формула для отримання нормально розподілених випадкових чисел Y із заданими параметрами **m** і **σ** буде такою:

$$Y = ((\sum_{i=1}^{12} r_i) - 6)\,\sigma_y + m_y \qquad (4.15)$$

Для генерації таких чисел пакет rnd містить клас Norm.

4.3 Реалізація генераторів випадкових чисел у фреймворці Simulation

До складу фреймворку входить група класів, що реалізують інтерфейс Randomable. і успадковують клас RandomGenerators. Саме ці класи використовуються для створення об'єктів, що забезпечують генерацію псевдо випадкових чисел для різних законів розподілу.

4.3.1 *Ієрархія класів RandomGenerators*

Клас RandomGenerators. Є базовим класом для всіх генераторів випадкових чисел, які використовуються в лабораторній роботі.. Цей абстрактний клас містить поле rnd, яке наслідується в усіх наслідуваних класах, і в яке при ініціалізації генератора заноситься стандартний генератор рівномірно розподілених випадкових чисел (ГРРВЧ). Діаграма класу RandomGenerators представлена на рисунку 4.8.

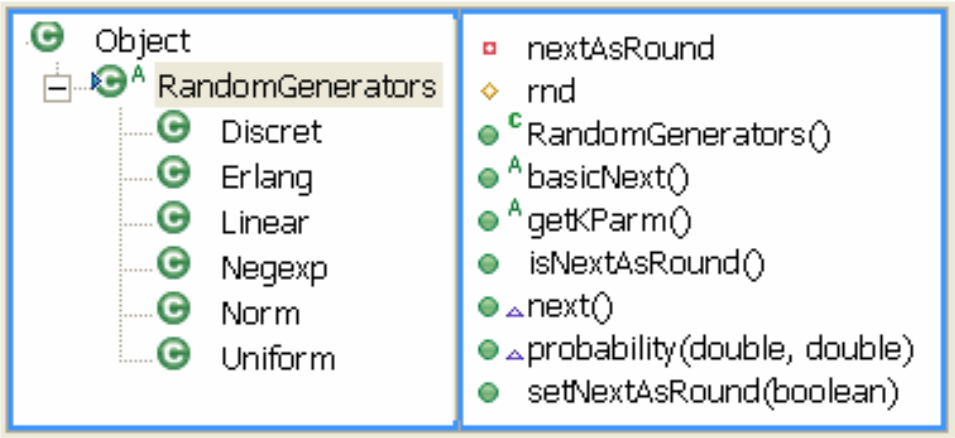

Рисунок 4.8 – Діаграма класу RandomGenerators

Змінна rnd створюється в конструкторі класу, який забезпечує затримку перед створенням наступного генератора, що виключає можливість однакової ініціалізації послідовно створюваних генераторів.

Клас реалізує інтерфейс Randomable. В цьому інтерфейсі необхідно визначити методи next(), probability(double) и probability(double, double).

Метод next() повинен повертати наступне випадкове число. Цей метод реалізований у класі RandomGenerators і викликає метод basicNext(), який об'явлений в даному класі як абстрактний і реалізований в кожному з підкласів. Використання проміжного методу дозволяє при необхідності отримувати на

виході генератору цілі числа. Для цього поле nextAsRound повинно мати значення true.

Метод probability(double) фактично визначає інтегральну функцію розподілення. Цей метод повинен повертати імовірність отримання числа меншого, ніж число, яке задається в якості параметру. Цей метод є специфічним для кожного закону розподілення і тому визначається у підкласах.

Метод probability(double, double) повинен повертати імовірність появи числа із діапазону, границі якого задаються параметрами методу. Його легко реалізувати використовуючи попередній метод, і він не залежить від особливостей закону розподілення.

Абстрактний метод getKParm() повинен повертати кількість параметрів розподілення, визначених по вибірці. Ця величина залежить від закону розподілення і тому визначається у підкласах. Це значення потрібно для визначення числа ступенів вільності під час перевірки статистичних гіпотез щодо законів розподілу.

З діаграми видно, які класи наслідують RandomGenerators. В цих класах реалізовані методики отримання випадкових чисел, описані в попередньому підрозділі.

4.3.2 *Клас TestFile*

Інтерфейс користувача класу TestFile представлено на рисунку 4.9.

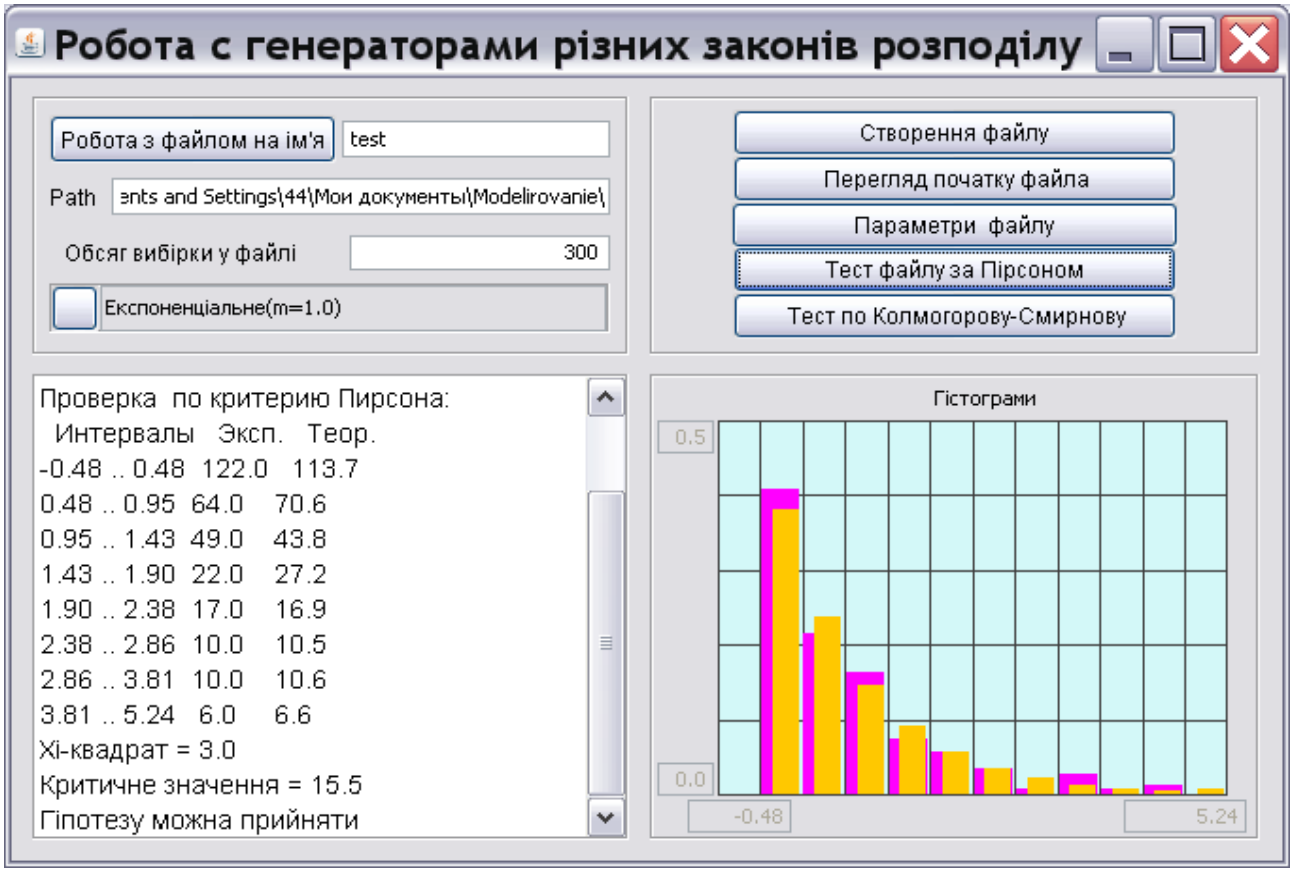

Рисунок 4.9 – Інтерфейс користувача проекту TestFile

Цей візуальний клас надає користувачеві такі можливості:

– переглядати дані файлу дійсних чисел;

– створювати файли випадкових величин із заданим законом розподілу,

– визначати статистичні характеристики файлів з випадковими величинами,

– проводити перевірки статистичних гіпотез щодо ймовірних законів розподілу.

Група елементів у лівій верхній частині форми містить поля через які задаються параметри файлу.

Ім'я файлу вводиться в поле «Робота з файлом на ім'я». Шлях до файлу задається у вікні «Path». При необхідності, за допомогою кнопки «Робота з файлом на ім'я» можна відкрити діалогове вікно вибору файлу. Ім'я вибраного файлу і шлях до нього фіксуються у відповідних полях. Поле «Обсяг вибірки» містить кількість чисел у файлі.

Компонент ChooseRandom дозволяє вибрати закон розподілення та його параметри, що використовуються при створенні файлу і при тестуванні.

Перелік можливих операцій із файлом випадкових чисел визначається набором кнопок у верхній правій частині форми. Результати тесту виводяться як у вигляді тексту, так і у вигляді гістограм.

Порядок роботи із застосуванням може бути наступним:

– обираємо файл з яким необхідно провести тест;

– за допомогою функції «Перегляд початку файлу» аналізуємо дані, і визначаємо, з якою випадковою величиною доводиться мати справу – дискретною або неперервною;

– за допомогою функції «Параметри файлу» отримуємо емпіричне розподілення випадкової величини у вигляді гістограми і числові характеристики вибірки випадкових чисел;

– на підставі аналізу отриманих даних висовуємо гіпотезу про закон розподілення, якому відповідає досліджувана вибірка;

– налаштовуємо компонент ChooseRandom згідно з передбачуваним розподіленням і його характеристиками;

– перевіряємо висунуту гіпотезу за критерієм Пірсона або Колмогорова-Смирнова.

4.4 Потоки випадкових подій

Потік випадкових подій – це послідовність якихось подій у часі, розділених випадковими інтервалами часу.

Існує декілька класифікацій таких потоків.

Перш за все слід розрізняти стаціонарні потоки подій і нестаціонарні. У стаціонарному потоці закон розподілу випадкової величини інтервалу між подіями та його параметри не змінюються з часом. Якщо ж закон розподілу для інтервалу, або його параметри з час змінюються, то такий потік буде нестаціонарним. Наприклад, потік тролейбусів на зупинці у години «пік»

можна вважати стаціонарним, але той же потік на протязі доби явно нестаціонарний, хоча б тому, що вночі тролейбуси зовсім не ходять.

Для стаціонарних потоків основна ознака для класифікації - це закон розподілу випадкової величини інтервалу часу між двома послідовними подіями.

Експоненціальний потік – це потік у якому інтервал між подіями підкоряється експоненціальному закону розподілу. Це дуже розповсюджений потік подій. За звичай він виникає у тих випадках, коли події виникають під впливом різноманітних, не пов'язаних між собою обставин. Прикладами таких подій можуть бути поява покупця у магазині, виклик лікаря швидкої допомоги і таке інше. Внаслідок своєї популярності потік має і декілька інших назв – пуасонівський потік, найпростіший потік.

Потік Ерланга – цей потік у якому інтервал між подіями підпорядковується закону Ерланга. Він формується з попереднього, шляхом вилучення із нього деякої кількості послідовних подій. Внаслідок цього потік ще звуть просіяним пуасонівським потоком. Такий потік може виникнути, наприклад, під час опитування перехожих на вулиці, коли опитують кожного третього.

Нормальний, або гаусовський потік - це потік у якому інтервал підпорядковується нормальному розподілу. Інтервали між подіями у такому потоці хоча і випадкові, але не дуже відрізняються один від одного. Приклад такого потоку – поява покупців на виході з супермаркету, де вони перед цим стояли у черзі до каси, поява тролейбуса на зупинці, у разі, якщо розкладом передбачено сталий інтервал руху. Такий потік може сформуватися і з найпростішого, якщо коефіцієнт просіювання достатньо великий. Так, наприклад, потік бажаючих виїхати до Києва у недільний вечір буде пуасонівським, але потік автобусів, що від'їжджають, як тільки завантажаться, можна вважати нормальним.

КОНТРОЛЬНІ ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Дайте характеристику методу оберненої функції.
2. Намалюйте графіки фунцій щільності ймовірностей для найбільш розповсюджених законів розподілу.
3. Для яких законів розподілу можна отримати аналітичний вираз для оберненої функції.
4. Сформулюйте теорему Ляпуноа і поясніть, як вона використовується для формування випадкових чисел.
5. Яким чином отримати випадкові числа Ерланга?
6. Яке призначення компоненту ChooseRandom та як його використовувати?
7. Які методи передбачені інтерфейсом Randomable?
8. Наведіть перелік класів, що успадковують клас RandomGenerators, та поясніть їх призначення.
9. Як використовувати застосування TestFile?

5 ТЕМА 5. ЗАСОБИ ІМІТАЦІЙНОГО МОДЕЛЮВАННЯ ПАРАЛЕЛЬНИХ ПРОЦЕСІВ

План лекції

– Особливості паралельних процесів.
– Шляхи реалізації моделей паралельних процесів.
– Класи для реалізації псевдопаралельних процесів.
– Приклад реалізації моделі з псевдопаралельними процесами.

5.1 Особливості паралельних процесів

Розглянемо як приклад модель, що імітує деякі змагання, наприклад, програмістів. Змагаються команди, що складаються із декількох гравців. Команди повинні послідовно виконати кілька завдань, кожне з яких складається з модулів. Кожен модуль створюється індивідуально одним із членів команди. Завдання вважається виконаним після того, як всі модулі будуть готові, об’єднані і налагоджені. Після завершення виконання чергового завдання команда отримує наступне завдання.

Правила дії кожного із членів команди полягають у багаторазовому повторенні наступних операцій:

1. Одержання чергового завдання на розробку модуля.
2. Затримка у часі , що імітує розробку чергового модуля.
3. Очікування завершення роботи над завданням усієї команди.
4. Перехід до пункту 1.

Аналізуючи наведений вище приклад, неважко побачити його особливості:

– правила дії всіх членів команд мають виконуватися одночасно;
– у правилах дії є затримки на деякий час;
– у правилах дії є зупинки до виконання деякої умови.

Розглянемо можливості мови програмування Java для програмної реалізації паралельних процесів.

5.2 Шляхи програмної реалізація моделей паралельних процесів

Мова програмування Java надає багато можливостей для програмної реалізації моделей паралельних процесів.

При програмуванні кожний процес можна пов’язати з деяким об’єктом, атрибути якого містять характеристики даного процесу. У наведеному прикладі можна виділити дві абстракції – член команди і команда. Атрибутом члена команди є посилання на команду, атрибутом команди є перелік членів команди.

Правила дії кожного процесу можна реалізувати у вигляді методів цих об’єктів.

Для того щоб реалізувати паралельне виконання коду та забезпечити зупинки у процесі виконання коду, правила дій об’єктів слід виконувати у окремих потоках.

Поняття процес тісно пов'язане з поняттям час. Тому постає питання – як пов’язати виконання правил дії об’єктів з часом?

Один із шляхів, який, можливо, не є найпростішим, полягає у моделюванні в реальному часі у комп’ютерній мережі, де кожний процес виконується на окремому комп’ютері. У цьому випадку затримки у часі можна забезпечити за допомогою метода sleep(). Але недолік такого способу полягає в тому, що імітаційна модель буде працювати з великою кількістю зупинок, що призведе до уповільнення роботи програми. Тривалість зупинок можна зменшити за рахунок зміни масштабу часу, але у цьому випадку затримки, що імітують виконання завдань, будуть близькими за величиною до часу виконання коду, що призведе до значних похибок.

Другий спосіб полягає в тому, що моделювання проводиться у деякому віртуальному часі, який змінюється циклічно із невеликим шагом. Це аналогія з чисельними методами. Але і тут імітаційна модель буде працювати повільно, внаслідок великої кількості «пустих тактів». Кількість пустих тактів можна зменшити за рахунок збільшення шагу зміни часу, але це зменшує точність обчислення часу.

Але найкращий спосіб полягає в тому, що затримки створюються в деякому віртуальному часі, який змінюється дискретно, від події до події, і ніяк не пов'язаний з реальним часом, і внаслідок цього не залежить від часу виконання коду. Але для реалізації такого способу необхідно мати об’єкт, який буде керувати цим віртуальним часом. Назвемо цей об’єкт диспетчером.

Диспетчер визначає час на підставі списку майбутніх подій. Цей список уявляє собою колекцію, у якій розташовуються посилання на об'єкти, що затримані на деякий час. Кожен з об’єктів у цьому списку знає час відновлення своїх правил дії, і список може бути впорядкованим саме за цим атрибутом об’єкту. Об’єкти самі заносять себе до цього списку перед тим як призупинитися.

Диспетчер активізується тоді, коли припиняються правила дії об'єктів, якими він керує. Диспетчер вилучає з управляючого списку об’єкт з найменшим часом відновлення правил дії і присвоює віртуальному часу значення, відповідне запланованому моменту відновлення правил дії вилученого об’єкту. Тобто диспетчер, побачивши яким повинен бути час, таким його і встановлює. Виходить, що об’єкти управляють часом, а не навпаки, і час змінюється дискретно від події до події. Змінивши час, диспетчер відновлює правила дії об’єкту, а сам призупиняється.

Окрім списку майбутніх подій диспетчер має ще список об’єктів, правила дії яких зупинені до виконання деяких умов. Так само як і у попередньому випадку, об’єкти знають, чого чекають і самі заносять себе до цього списку перед зупинкою. Диспетчер переглядає цей список кожного разу перед зміною модельного часу, і якщо якась умова виконується, то правила дії відповідного об’єкту поновлюються без зміни часу.

Диспетчер не тільки відновлює правила дії об’єктів, що були призупинені, але й ініціює початок їх виконання. Для реалізації цього завдання диспетчер має ще так званий стартовий список. До цього списку перед початком моделювання заносяться об’єкти, що мають виконувати свої правила дії. Диспетчер починає свою роботу з опрацювання саме цього списку. Він по черзі вилучає об’єкти із списку і активізує їх правила дії, а сам після цього призупиняється доти, поки активізований об’єкт не призупиниться.

На рисунку 5.1 наведено схему алгоритму роботи диспетчера.

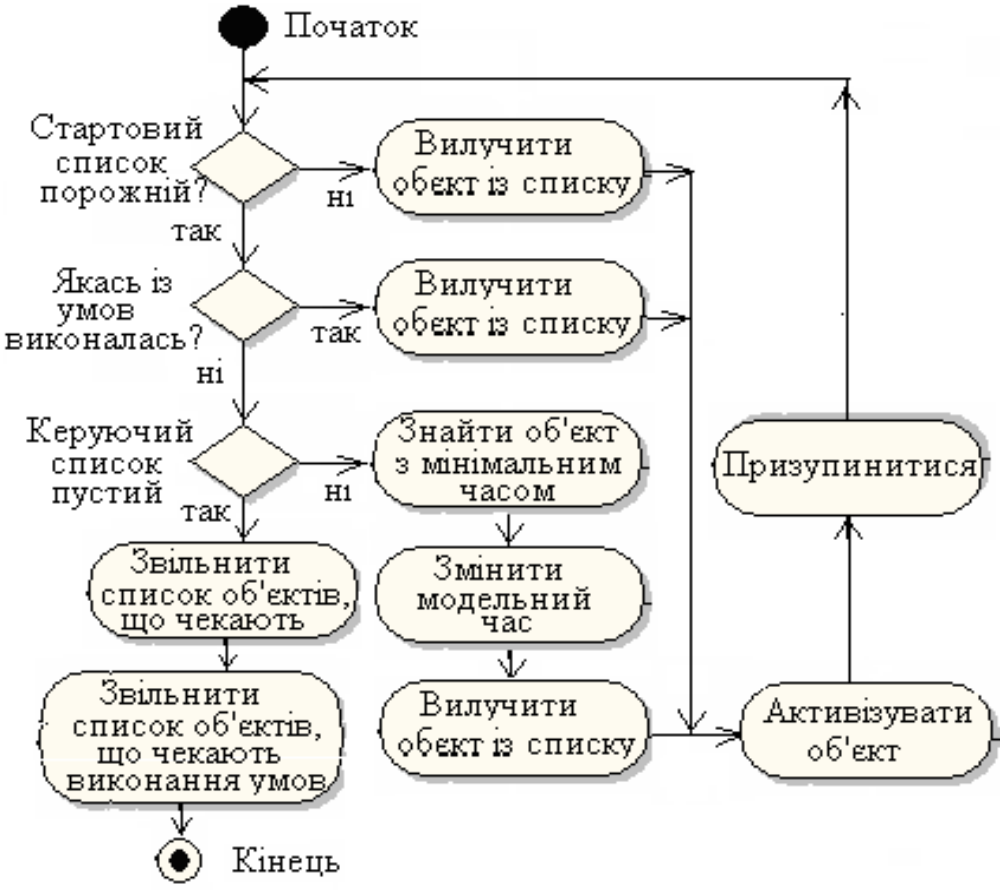

Рисунок 5.1 – Схема алгоритму роботи диспетчера

Особливість використання потоків при програмуванні імітаційних моделей полягає також і у тому, що під час роботи моделі не існує більше одного активного потоку, пов’язаного з роботою моделі. З цієї причини потоки, що імітують паралельне виконання правил дії об’єктів, називають псевдопаралельними. Це має місце внаслідок того, що диспетчер, породжуючи або активізуючи потік виконання правил дії якогось об’єкту, сам призупиняється доти, поки не призупиниться активізований потік. Таким чином виконується або потік якогось з об’єктів, або потік диспетчера. Завдяки цьому підтримується сувора послідовність подій у віртуальному (модельному) часі.

Внаслідок псевдопаралельного виконання потоків у імітаційних моделей практично не існує проблеми використання спільних ресурсів.

Тепер докладніше розглянемо вимоги до об’єктів, що мають виконувати правила дії та шляхи їх реалізації.

Як вже було сказано, правила дії мають виконуватися у окремих потоках. Базовим класом потоків у мові Java є клас Thread. Але потік виконання правил дії є тільки складовою частиною об’єкту. Диспетчер і об’єкти, що діють паралельно, можуть існувати і до створення потоку, і після того, як потік завершився. Тому створювати відповідні класи ми будемо не шляхом наслідування класу Thread, а шляхом реалізації інтерфейсу Runnable у цих класах. Як відомо, цей інтерфейс передбачає реалізацію публічного методу run(), який і є точкою входу до потоку, що створюється.

Таким чином клас, що реалізує правила дії процесу може мати структуру, представлену у лістингу 5.1.

Лістинг 5.1 – Структура класу, що реалізує правила дії процесу

```
public abstract class КЛАС implements Runnable{
   void start() {
      thread = new Thread(this);
      thread.start();
   }
   public void run() {
      rule(); //  метод реалізації правил дії, абстрактний у цьому класі
   }
}
```

Наведену структуру класу можна розглядати, як шаблон для усіх класів, пов’язаних з правилами дії. Тому у фреймворці доцільно створити абстрактний клас із такою структурою, до якого винести загальні характеристики об’єктів такого типу. А правила дії та інші особливості конкретних об’єктів реалізовувати у класах спадкоємцях.

Для того, щоб призупиняти виконання правил дії об’єктів та диспетчера можна використовувати метод wait() класу Object. Для відновлення виконання правил дії цих об’єктів можна використовувати метод notify() класу Object. Слід нагадати, що методи wait() і notify() класу Object не можна використовувати прямо, викликаючи їх для потоків, якими ми хочемо керувати. Це можна робити за допомогою деякого іншого, синхронізованого об'єкта з публічним доступом, що приймає повідомлення wait() і notify().

Загальний шаблон використання методу wait() наведено у лістингу 5.2.

Лістинг 5.2 – Шаблон використання методу wait()

```
while (УМОВА) {
   try {
        synchronized (монітор)
```

```
            монітор.wait();
        } catch (java.lang.InterruptedException e) {
            e.printStackTrace();
        }
    }
```

Аналіз цього шаблону свідчить про те, що для реалізації зупинки необхідні такі складові:

- об’єкт монітор;
- умова відновлення роботи;
- синтаксична конструкція з while і try..catch.

Монітором може бути об’єкт будь якого існуючого класу. Але зважаючи на громіздкість шаблону, краще створити спеціальний клас, у методі якого реалізувати даний шаблон.

Умовою є наявність, чи відсутність дозволу на продовження роботи, який надає диспетчер. Для цієї умови у створюваному класі можна виділити поле типу boolean.

У наступному підрозділі цей клас охарактеризовано більш докладно.

Ще одна проблема програмування імітаційних моделей полягає в тому, що об'єкти, які чекають виконання деяких умов, повинні ці умови пам'ятати. Причому, таких умов у одного об'єкта може бути декілька. У Java ми можемо оперувати тільки об'єктами, тому умова може бути створена тільки як метод деякого об'єкта. Для створення таких об'єктів використовується інтерфейс IWaitCondition, який потребує реалізації методу testCondition(), що забезпечує перевірку необхідної умови. Таким чином, якщо виникає потреба у передачі або зберіганні деякої умови, то створюється об’єкт класу, що реалізує інтерфейс IWaitCondition і в ньому реалізується метод testCondition(), який забезпечує перевірку цієї умови. У зв’язку з тим що такий клас має локальне значення, і використовується для створення єдиного об'єкта, то його доцільно створювати анонімним, безпосередньо у правилах дії об'єкту, що потребує використання умови, а для посилань на його тип використовувати ім'я інтерфейсу.

5.3 Класи, що забезпечують динамічну взаємодію псевдопаралельних процесів

Класи, що забезпечують динамічну взаємодію псевдопаралельних процесів об'єднані у пакеті process.

Найважливіші з них, що необхідні для розуміння особливостей роботи імітаційних моделей розглядаються нижче.

Клас Semaphore використовується для створення об’єктів, що дозволяють призупиняти і відновлювати потоки виконання правил дії об’єктів за допомогою методів wait() і notify().

Інтерфейс IWaitCondition використовується для створення об’єктів, що зберігають деякі умови у вигляді методу testCondition().

Клас Actor визначає найбільш загальні властивості й особливості поведінки об'єктів, які повинні виконувати свої правила дії в часі.

Клас Dispatcher описує поведінку і властивості об'єкта, що забезпечує синхронізацію виконання правил дії компонент моделі, зміну модельного часу й формування протоколу роботи моделі.

5.3.1 *Клас BooleanSemaphore*

Об'єкти цього класу використовуються для керування потоками. Клас має всього одне приватне поле логічного типу з ім'ям value, що може бути використане як індикатор стану потоку.

Для затримки потоків за допомогою об'єктів даного класу використовується метод waitForValue(boolean), приведений у лістингу 5.2.

Лістинг 5.3 – Метод waitForValue(boolean)

```
public synchronized void waitForValue(boolean state) {
   while (value != state) {
      try   {
                  wait();
            } catch (java.lang.InterruptedException e) {
                  e.printStackTrace();
            }
   }
}
```

Цей метод використовує стандартний метод wait(), який забезпечує затримку потоку доти, доки індикатор не перейде в необхідний стан.

Для зміни значень поля value використовується метод setValue(boolean).

Лістинг 5.3 – Метод setValue(boolean)

```
   public synchronized void setValue(boolean newValue) {
      if (newValue != value) {
         value = newValue;
         notifyAll();
      }
   }
```

Приведений метод не тільки змінює стан індикатора, але й сповіщає про це всі зацікавлені потоки за допомогою стандартного повідомлення notifyAll(), що приводить до поновлення виконання правил дії потоків, які чекали зміни стану індикатора.

5.3.2 *Клас Actor*

Цей абстрактний клас містить найбільш загальні властивості й методи об'єктів, які мають правила дії, розподілені в часі.

Клас реалізує два інтерфейси.

Інтерфейс Runnable дає можливість виконувати правила дії об'єктів як потоки. Відповідно до вимог цього інтерфейсу в класі реалізований метод run(), що забезпечує запуск правил дії об'єктів підкласів.

Інтерфейс Clonable дає можливість клонувати об'єкти класу або його підкласів. Відповідно до вимог цього інтерфейсу в класі реалізований метод clone(). Цей метод, а також особливості клонування об'єктів в Java, будуть розглянуті нижче, при розгляді класу MultyActor.

5.3.2.1 **Поля об'єктів класу та їх призначення**

Поле activateCondition зберігає посилання на об'єкт типу BooleanSupplier, що у вигляді методу getAsBoolean() містить умову, виконання якої чекає призупинений об'єкт.

Поле activateTime, типу double, містить час поновлення виконання правил дії, призупинених на якийсь час.

Поле dispatcher містить посилання на об'єкт класу Dispatcher, під керуванням якого будуть виконуватися правила дії об'єкта.

Поле nameForProtocol містить ім'я об'єкту, що використовується у протоколі роботи моделі.

Поле suspendIndicator містить посилання на об'єкт класу BooleanSemaphore і використовується для призупинення та відновлення правил дії.

Поле waitingTimeHisto призначено для зберігання посилання на об'єкт класу Histo. Якщо таке посилання передано актору, то у відповідній гістограмі буде накопичуватися інформація про час чекання актора.

Для доступу до полів є методи, названі у відповідності зі стандартними правилами.

5.3.2.2 **Методи для створення потоку виконання правил дії**

Створення та запуск потоку виконання правил дії для об'єктів класу Actor виконується за допомогою методу start(). Новий потік створюється за допомогою конструктора класу Thread, куди як параметр передається посилання на «актора», чиї правила дії повинні виконуватися в цьому потоці. Потоку привласнюється ім'я «актора». Цей метод призначений для використання тільки об’єктом класу Dispatcher, тому не має публічного доступу.

Публічний метод з ім'ям run(), є точкою входу потоку виконання правил дії «актора». Метод реалізований у класі Actor відповідно до вимог інтерфейсу Runnable. У методі викликаються правила дії об'єкта шляхом звертання до методу rule(). Метод rule() у класі Actor є абстрактним. Передбачається, що в підкласах цей метод буде реалізований відповідно до правил дії об’єктів підкласів.

Перед викликом правил дії метод run() виводить до протоколу диспетчера повідомлення про початок роботи об'єкта. Після завершення виконання правил дії метод виводить до протоколу повідомлення про те, що об'єкт роботу закінчив.

Закінчується метод run() викликом методу, що встановлює індикатору стану потоку значення, яке сигналізує про те, що виконання потоку припинено. Цей сигнал дозволяє активізувати роботу іншого потоку, що, можливо, очікує завершення роботи даного.

5.3.2.3 **Методи, що забезпечують призупинення правил дії об'єктів**

Для об'єктів класу Actor передбачено три види зупинок.

Перший вид зупинки – це зупинка на якийсь час (мається на увазі віртуальний модельний час), що за звичай імітує роботу об'єкта, на виконання якої потрібне певний час.

Затримка реалізується за допомогою методу holdForTime(double), схему алгоритму якого наведено на рисунку 5.2.

У метод передається параметр holdTime, значення якого дорівнює необхідному часу затримки.

Рисунок 5.2 – Алгоритму виконання методу holdForTime

При виконанні методу, визначається час майбутньої активізації об'єкта – activateTime. Значення цього поля встановлюється рівним сумі поточного значення модельного часу й часу затримки holdTime.

Після цього посилання на об'єкт додається до списку майбутніх подій диспетчера.

Виклик методу suspendIndicator.setValue(true) приводить до того, що об'єкти, що чекають зміни стану об’єкту, можуть відновити роботу. Найчастіше таким об'єктом є диспетчер.

Виклик методу suspendIndicator.waitForValue(false) приводить до призупинення правил дії об'єкта.

Другий вид зупинки виконання правил дії – це затримка до виконання деякої умови. Така затримка реалізується за допомогою методу waitForCondition(BooleanSupplier, String).

Першим параметром методу є об'єкт інтерфейсного типу BooleanSupplier. Цей інтерфейс передбачає реалізацію методу getAsBoolean(), що забезпечує перевірку необхідної умови. Таким чином, у метод waitForCondition() передається не сама умова (що в Java неможливо), а об'єкт класу, через який можна викликати метод перевірки цієї умови.

Другим параметром методу є рядок символів, що використовується для ідентифікації умови у протоколі.

Схема алгоритму цього методу наведена на рисунку 5.3.

Рисунок 5.3 – Алгоритму виконання методу waitForCondition

При виконанні методу waitForCondition(), насамперед, буде проводитись перевірка умови за допомогою виклику методу c.testCondition(). Якщо умова виконується, то робота методу закінчується. Якщо ж умова не виконується, то посилання на об'єкт підкласу IWaitCondition заносяться в поле activateCondition «актора», а посилання на самого «актора» передається диспетчерові.

Виклик методу suspendIndicator.setValue(true) приводить до того, що об'єкти, що чекають цього перемикання, можуть відновити роботу. Частіше за все таким об'єктом є диспетчер.

Виклик методу suspendIndicator.waitForValue(false) приводить до припинення правил дії об'єкта.

Особливість виконання цього методу полягає у тому, що очікувана умова можливо не буде виконана до закінчення роботи диспетчера. У цьому випадку диспетчер перед закінченням роботи примусово активізує потік незважаючи на невиконання умови. Для того, щоб цей випадок можна було б належним чином обробити, метод викликає виключну ситуацію DispatcherFinishException, яка має бути опрацьована у правилах дії об’єкту. За звичай таке опрацювання полягає у припиненні правил дії (return).

Третій вид зупинки виконання правил дії – це затримка до виконання деякої умови але не довше, ніж на визначений час. Така затримка реалізується за допомогою методу waitForConditionOrHoldForTime(BooleanSupplier, String, double), або методу holdForTimeOrWaitForCondition (double, BooleanSupplier, String,). Методи дуже схожі на обидва попередні методи. Різниця полягає тільки у тому, що посилання на актора розміщується у обох списках диспетчера і тому продовження правил дії відновляється або після виконання умови, або у призначений час. Наводити цей метод тут не будемо.

5.3.3 *Клас Dispatcher*

Головне призначення цього класу – керувати модельним часом і забезпечити синхронізацію виконання правил дії «акторів». Крім того диспетчер формує протокол роботи «акторів», що істотно спрощує налагодження застосувань.

5.3.3.1 **Поля диспетчера**

Поля об'єктів цього класу мають таке призначення:

currentTime – поточне значення модельного часу.

startList – список об'єктів, правила дії яких диспетчер повинен активізувати. На початку роботи цей список повинен містити посилання хоча б на один об'єкт, яким повинен керувати диспетчер. Поповнення цього списку можливо за допомогою методу addStartingActor(Actor).

timingActorQueue – список майбутніх подій, де перебувають посилання на об'єкти, правила дії яких припинені на деякий час. Об'єкти в цьому списку мають бути впорядковані за часом, на яке запланованe поновлення правил дії.

waitingActorQueue – список посилань на об'єкти, у яких правила дії припинені до виконання деяких умов. Об'єкти в цьому списку впорядковані по правилу FIFO.

protocolFileName – рядок, що визначає ім'я файлу, куди буде виводиться протокол роботи моделі. Якщо ця змінна має значення “Console”, протокол виводиться на консоль. Якщо значення цієї змінної пустий рядок, протокол не виводиться.

thread – посилання на потік виконання правил дії диспетчера.

active – поле логічного типу, що має значення true, поки диспетчер працює. Перед закінченням роботи «диспетчер» установлює значення поля рівним false.

dispatcherStartListeners, changeTimeListeners, dispatcherFinishListeners – це колекції, що зберігають посилання на слухачів подій, які генерує диспетчер під час своєї роботи. Подія dispatcherStartEvent сповіщає слухачів про початок роботи диспетчера. Подія changeTimeEvent сповіщає слухачів про зміну модельного часу. Подія dispatcherFinishEvent сповіщає слухачів про завершення роботи диспетчера.

5.3.3.2 **Робота диспетчера**

Диспетчер, так само як і «актори», має свої правила дії. Ці правила дії виконуються в окремому потоці, що вимагає реалізації інтерфейсу Runnable у класі Dispatcher і наявності в ньому методу run.

Запуск диспетчера відбувається шляхом виклику публічного методу start() класу Dispatcher. Виклик цього методу означає початок процесу моделювання.

Правила дії «акторів», що стартують, виконуються по черзі. Новий потік стартує тільки після того, як буде припинений попередній. Такий порядок забезпечує об’єкт класу Semaphore, який знаходиться у полі suspendIndicator класу Actor. Коли правила дії чергового «актора» припиняються, цикл активізації акторів продовжується і стартує новий актор. Виконання методу закінчується тільки тоді, коли правила дії останнього «актора», що стартує, будуть призупинені.

Після виконання методу runStartList() посилання на «акторів» потрапляють або у список майбутніх подій timingActorQueue, або в список waitingActorQueue, де знаходяться об'єкти, що чекають виконання умов. На цьому завершується перший етапу циклу правил дії диспетчера.

Наступні етапи циклу робота «диспетчера» полягають у обробці списків timingActorQueue та waitingActorQueue.

Насамперед, аналізується список об'єктів, що чекають виконання умов. Для цього використовується метод testWaitingQueue().

Метод забезпечує послідовний перегляд списку «акторів», що чекають виконання умов, і перевірку очікуваної умови для кожного з них. Нагадаємо, що в полі activateCondition акторів зберігаються посилання на об'єкт класу, що реалізує метод testCondition(), відповідно до вимог інтерфейсу IWaitCondition. Саме метод testCondition() перевіряє необхідну умову.

Якщо умова, що перевіряється, виконується для якогось «актора», то цей «актор» вилучається зі списку и повертається, як результат виконання методу. Посилання на цього актора вилучається також і із списку майбутніх подій якщо воно там є. У випадку коли актор, для якого виконалась очікувана подія, не знайдений, метод повертає null. Результат виконання методу привласнюється змінній readyActor.

Наступний етап основного циклу роботи диспетчера виконується в тому випадку, якщо значення змінної readyActor дорівнює null. На цьому етапі диспетчер працює зі списком «акторів», правила дії яких припинені на якийсь час. Якщо цей список порожній, то диспетчер закінчує свою роботу. Обробка не порожнього списку полягає в тім, що із списку вилучається «актор» з мінімальним часом активізації. Посилання на цей об'єкт записується в змінну readyActor Модельному часу currentTime привласнюється значення часу активізації об'єкта readyActor, що зберігається в змінній activateTime. Тобто диспетчер, побачивши, яким повинен бути час найближчої події, таким його й встановлює. Для зміни модельного часу викликається метод setCurrentTime, що забезпечує вивід рядка протоколу з новим значенням часу. Крім того, у методі setCurrentTime створюється подія ChangeTimeEvent, яка може бути використана

іншими об'єктами, наприклад, для індикації поточного часу або збору статистичної інформації про роботу об'єкта.

На завершальному етапі виконання циклу змінюється стан індикатора «актора» readyObject, внаслідок чого виконання правил дії «актора» може відновитися. А для того, щоб це дійсно відбулося, робота диспетчера припиняється за допомогою індикатора того «актора», чиї правила дії відновляються, шляхом виклику методу readyActor.getSuspendIndicator(). waitForValue(true).

Робота диспетчера буде відновлена тільки після того як правила дії активізованого «актора» призупиняться або закінчаться.

Цикл виконання правил дії диспетчера триває доти, поки список timingActorQueue не стане порожнім, що свідчить про зупинку модельного часу.

Після завершення циклу правил дії диспетчера виконуються операції, що завершують роботу диспетчера.

Насамперед, у поле active заноситься значення false. Після цього викликається метод releaseWaitingQueue().

Цей метод послідовно активізує правила дії «акторів», які чекають виконання умов, але вже не дочекаються цього. Передбачається, що після виходу з режиму очікування «актор» завершить свої правила дії, обробивши виключну ситуацію DispatcherFinishExeption.

Такий «дивний» спосіб пов'язаний з тим, що в Java неможливо переривати виконання потоку зовні. Потік повинен сам завершувати свою роботу.

Після виконання завершальних операцій диспетчер формує подію DispatcherFinishEvent, що сповіщає всіх зацікавлених слухачів про завершення роботи диспетчера.

5.4 Приклад використанням розглянутих класів

Можливості використання розглянутих класів для імітаційного моделювання продемонструємо на прикладі, який реалізований у лабораторному застосуванні.

У цьому прикладі побудована модель, що імітує процес просування деякого якісного товару на ринок за допомогою реклами. Модель імітує поведінку деякої кількості потенційних покупців товару (клієнтів). Кожний з них знаходиться під впливом реклами і раніше, чи пізніше вирішує придбати товар, що рекламується. Впевнившись у якості товару він починає рекомендувати його оточуючим, які довіряючи йому теж починають купувати цей товар.

5.4.1 *Проектування імітаційної моделі*

Об’єктно-орієнтований підхід до проектування моделі передбачає виділення у системі, що підлягає моделюванню, так званих ключових абстракцій, на основі яких будуть створені класи, необхідні для побудови

моделі. Поняття «абстракція» передбачає, що при розгляді якоїсь частини реальної системи ми беремо до уваги тільки ті її властивості, які мають значення для вирішення поставленого завдання. Так, наприклад, при розгляді абстрактного клієнта для задачі, що була сформульована, нас цікавить як на нього впливає реклама, але зовсім не цікавить його зріст та вага. Інколи може бути і так, що абстракція не відповідає жодній частині реальної системи.

Від правильної декомпозиції реальної систем і представлення її у вигляді сукупності абстракцій, що пов'язані між собою, залежить трудомісткість створюваного програмного продукту, його якість та життєздатність. Іноді ці показники входять у протиріччя.

Ми розділимо нашу систему на абстракції, що розглянуті нижче, і визначимо основні властивості і відповідальність кожної з них.

5.4.1.1 Інтерфейс користувача

Інтерфейс користувача має бути посередником між користувачем і моделлю . Його вигляд може бути яким, як показано на рисунку 5.4.

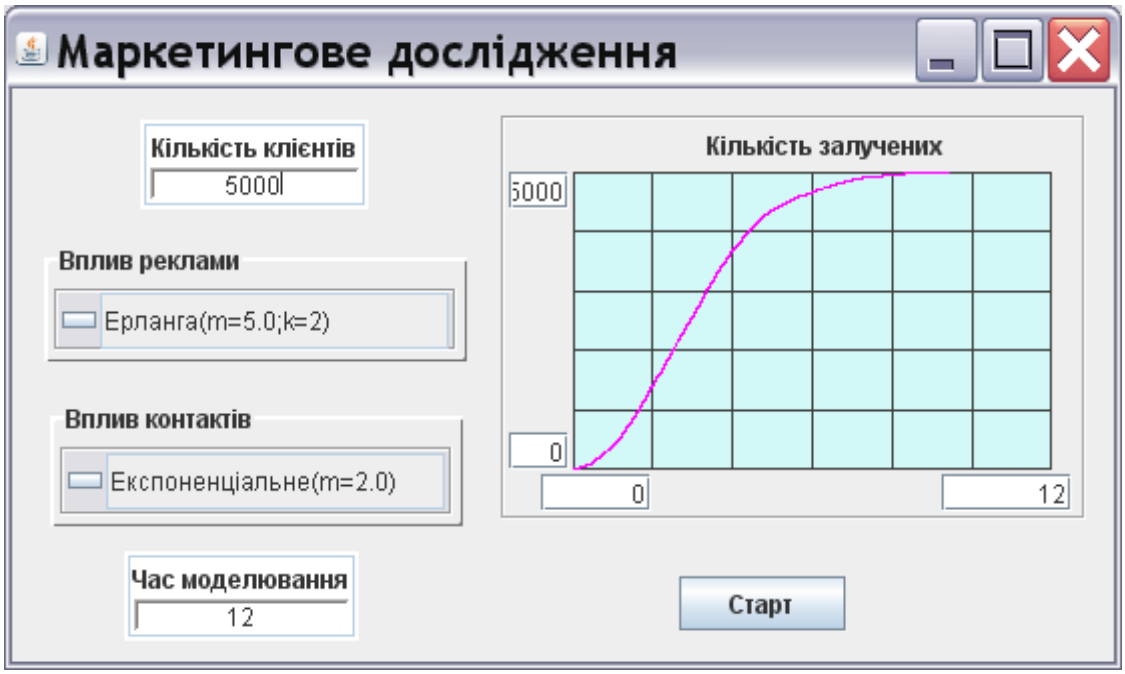

Рисунок 5.4 – Вигляд інтерфейсу користувача моделі

Інтерфейс повинен надати можливість:

– встановлювати кількість потенційних покупців;

– визначити закон розподілу для часу, за який реклама переконує потенційного покупця у необхідності придбати товар;

– визначити закон розподілу для інтервалу часу, через який покупець товару зустрічається з іншими потенційними покупцям;

– налаштувати час моделювання;

– ініціювати процес моделювання;

– надати можливість виведення графіку залежності кількості покупців товару від часу.

Для завдань цієї абстракції доцільно включати і створення об’єкту класу Dispatcher та передачу його моделі. Це пов’язано з тим, що під управлінням інтерфейсу користувача може знаходитися декілька моделей, що пов’язані між собою. У цьому разі важливо, щоб вони працювали під керівництвом одного й того ж диспетчера.

5.4.1.2 **Модель процесу.**

Модель створює потрібну кількість клієнтів, зберігає у себе їх список а також передає їх до стартового списку об’єкту класу Dispatcher. Модель також ініціює процес моделювання шляхом виклику для диспетчера методу start(). Під час моделювання модель має отримувати повідомлення від клієнтів про їх залучення до покупців товару, накопичувати їх кількість і відображати кількість залучених на екрані.

5.4.1.3 **Клієнт.**

Клієнт деякий час знаходиться під впливом реклами і нарешті стає покупцем, про що повідомляє модель. Крім того, можливо, що він стане покупцем і раніше під впливом інших. Ставши покупцем клієнт через випадковий час зустрічається з іншими клієнтами і залучає їх до кола покупців, якщо вони ще не є такими. Схема алгоритму правил дії клієнта наведено на рисунку 5.5.

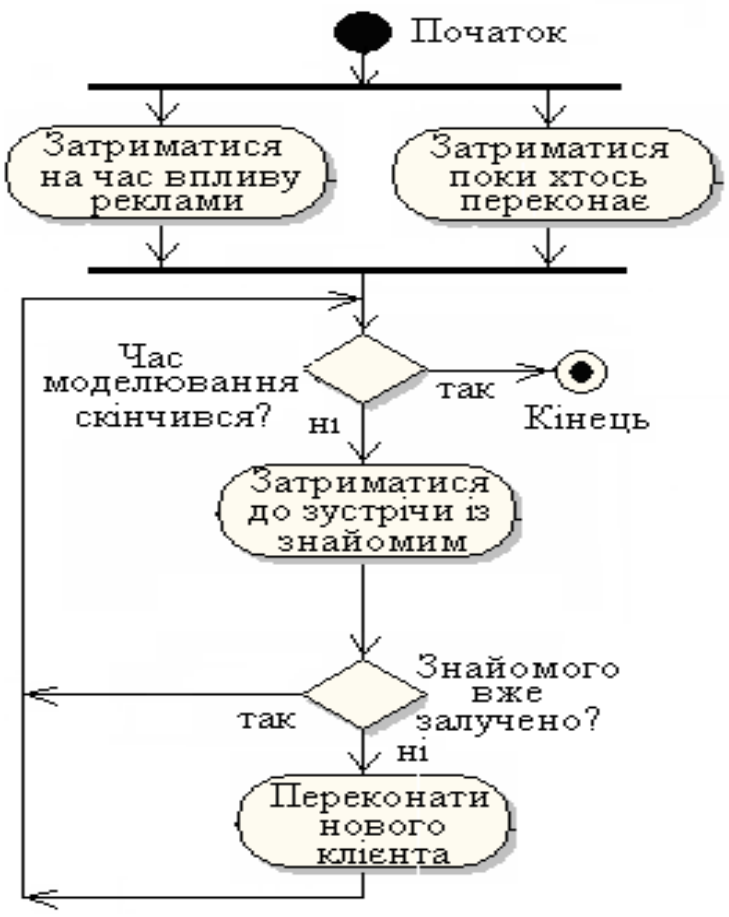

Рисунок 5.5 – Алгоритму роботи клієнта

КОНТРОЛЬНІ ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Які проблеми виникають при моделюванні паралельних процесів?
2. Як призупинити потік за допомогою об’єкту класу Semaphore?
3. Як відновити роботу потоку за допомогою об’єкту класу Semaphore?
4. За якими ознаками можна визначити приналежність об’єкту до класу Actor?
5. Який метод слід використати для зупинки виконання правил дії об’єкту на деякий час?
6. Алгоритм виконання методу HoldForTime(double).
7. Який метод слід використати для зупинки виконання правил дії об’єкту до виконання деякої умови?
8. Алгоритм виконання методу WaitForCondition(IWaitCondition).
9. Призначення та методи інтерфейсу IwaitCondition?
10. Як сформувати умову, виконання якої чекає актор?
11. Чи можна запланувати альтернативну зупинку правил дії – або до виконання умови, або на деякий час?
12. Коли розпочинається виконання правил дії актора?
13. Який метод використовується для передачі актора диспетчеру?
14. У яких списках диспетчера можуть перебувати актори?
15. У якій послідовності диспетчер переглядає списки з акторами?
16. Які події і коли формує диспетчер?
17. Як запрограмувати реакцію на подію диспетчера?
18. Що відбувається з акторами, які не дочекалися виконання умови?
19. Як потрібно обробляти виключну ситуацію DispatcherFinishEvent?
20. Як можна виведенням протоколу роботи моделі?
21. Як виводити повідомлення у протокол?

6 ТЕМА 6. ПОБУДОВА ІМІТАЦІЙНИХ МОДЕЛЕЙ СИСТЕМ МАСОВОГО ОБСЛУГОВУВАННЯ

План лекції

– Системи масового обслуговування.
– Компоненти фреймворку Simulation для побудови моделі СМО.
– Методика побудови моделі СМО.
– Приклад побудови програмної системи для моделювання СМО.

6.1 Системи масового обслуговування

Система масового обслуговування (СМО), або система з чергами (queued system) - це система, для якої характерна наявність таких характерних рис:

– випадковий потік заявок (транзакцій) на обслуговування;
– черги заявок, що чекають обслуговування;
– пристрої, що обслуговує ці заявки.

На рисунку 6.1 представлено схематичне зображення найпростішої системи масового обслуговування.

Рисунок 6.1 – Схема найпростішої СМО

Але СМО може мати и складнішу структуру. На рисунку 6.2 показано схему багатоканальної двофазної СМО.

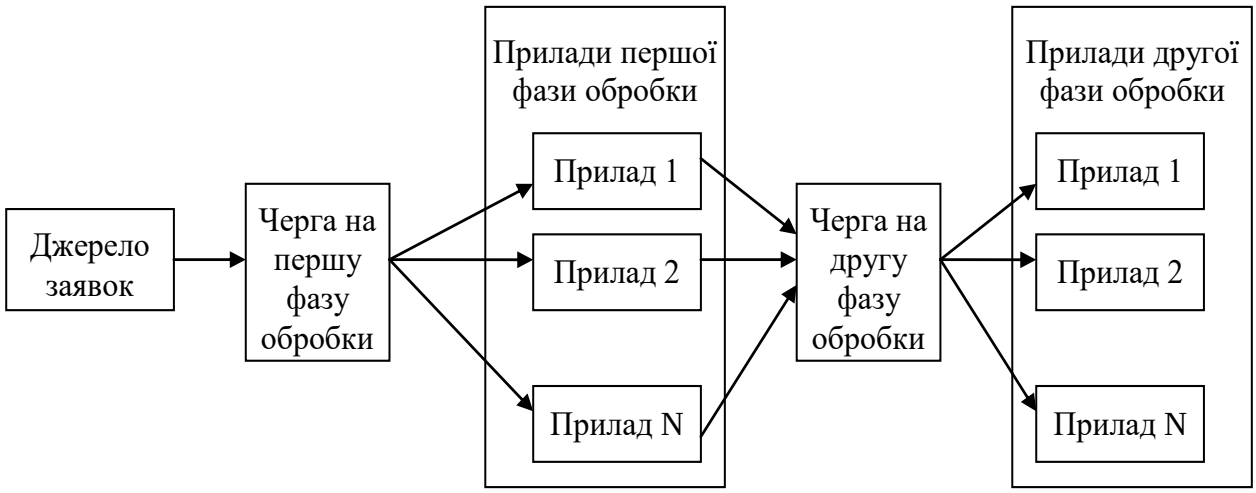

Рисунок 6.2 – Схема двофазної багатоканальної СМО

Багатоканальною ця система є з тієї причини, що обробка заявок виконується паралельно багатьма обслуговуючим пристроями. А внаслідок

того, що кожна заявка потребує обробки спочатку на одному з приладів першої групи, по потім на приладі другої групи, система є двофазною.

На рисунку 6.3 зображено СМО із каналом зворотного зв'язку. Такі СМО називають СМО із зворотними зв'язками.

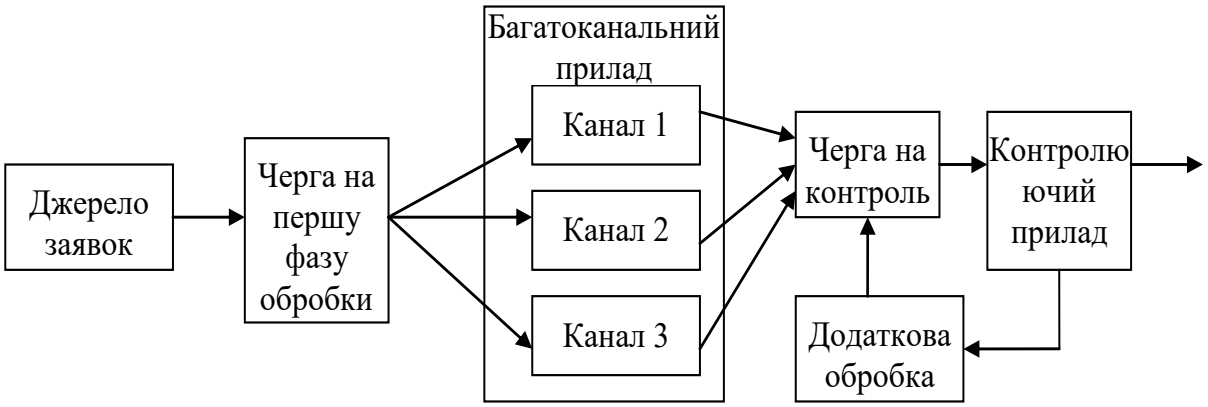

Рисунок 6.3 – Схема двофазної багатоканальної СМО із зворотними зв'язками

Існують і мережі СМО. Прикладом такої СМО може бути комп'ютерна мережа.

В усіх розглянутих вище прикладах СМО являє собою деяку структуру до складу якої входять джерела заявок, черги та обслуговуючі пристрої. При цьому джерела заявок і обслуговуючі пристрої є активними елементами, що керують переміщенням трансакцій і створюють події у часі. Джерело заявок затримується на деякий час і після цього формує заявку і додає її до черги – це і є подія. Обслуговуючий пристрій чекає появи заявки у черзі, а дочекавшись забирає її і витрачає деякий час на роботу із заявкою. Момент завершення обслуговування теж є подією. Черги ж є пасивними елементами і використовуються як місце перебування заявок між фазами обробки. Їх стан (заповнення) залежіть від взаємодії активних елементів. Поява заявки в черзі – це теж подія і вона відбувається у певний час, але цю подію створює не черга, а той об'єкт, що додає заявку до черги.

Трансакція у такій СМО теж є пасивним елементом. Вона може зберігати у собі деяку інформацію, але не керує своїм переміщенням у системі.

Однак у деяких СМО трансакції можуть бути активними, і у окремих фазах свого життєвого циклу можуть самостійно приймати рішення про переміщення по системі. Так само і активні елементи системи у деяких випадках можуть ставати трансакціями і чекати обслуговування. Так, наприклад, хворий (заявка), втомившись у черзі чекати обслуговування може покинути чергу і піти із скаргою до головного лікаря. Лікар, що веде прийом хворих і являє собою обслуговуючий пристрій СМО, може на деякий час припинити обслуговування хворих і піти на консультацію до іншого лікаря, таким чином перетворившись у заявку. Для таких СМО структурне зображення буде неповним. Поведінка таких нестандартних елементів СМО може бути представлена схемою алгоритму або діаграмою діяльності.

При моделюванні СМО досліднику відомі характеристики окремих компонент системи, а невідомим є результат їх взаємодії. У результаті дослідження найчастіше отримують таку інформацію:

– статистичні характеристики для довжини черг;

– статистичні характеристики для часу очікування в чергах;

– статистичні характеристики для часу обслуговування заявок;

– коефіцієнт завантаження обслуговуючих приладів.

Окрім того метою дослідження може бути отримання залежності наведених вище характеристик від параметрів СМО (кількості обслуговуючих пристроїв, характеристик закону обслуговування і таке інше).

Перелічені вище задачі стосуються сталого режиму роботи СМО, але інколи досліджують і перехідні процеси у СМО.

6.2 Компоненти фреймворку Simulation для побудови моделі СМО

Фреймворк Simulation надає розробнику моделі достатньо ефективні засоби побудови складових частин моделі. Клас Actor уже розглядався у попередній лабораторній роботі. На базі цього класу можна створювати класи для активних компонентів моделі, реалізувавши правила дії цих компонентів у вигляді методів rule().

Нижче розглянуто ще декілька класів, що можна використовувати для створення складових частин моделі.

6.2.1 *Клас ChooseData*

Об’єкти цього класу доцільно використовувати для введення числових параметрів компонент моделі. Цей клас успадковує клас JTextField. Поряд із усіма можливостями стандартного текстового компоненту користувач класу ChooseData отримує додаткові послуги. Вигляд компоненту представлено на рисунку 6.4.

Рисунок 6.4 – Вигляд компоненту класу ChooseData

Метод setTitle(String) дозволяє позначити призначення компоненту.

Методи getInt() та getDouble() дозволяють отримати числа відповідних типів.

Методи getIntArray() та getDoubleArray() дозволяють отримати масиви відповідних типів.

6.2.2 *Класи для накопичення статистичних даних*

Класи Histo і DiscretHisto дозволяють накопичувати інформацію про значення випадкових безперервних і дискретних величин, і подавати інформацію про ці величини у вигляді гістограм. Гістограма може бути

представлена у вигляді стовпчастої діаграми або у вигляді таблиці.

Перед використанням гістограма має бути проініціалізована. Найчастіше використовують такі методи ініціалізації:

– init(). Цей метод без параметрів можна використовувати для об'єктів обох класів.

– initFromTo(int, int). Цей метод можна використовувати для об'єктів класу DiscretHisto. Два цілих числа, що передаються в якості параметрів, визначають межі гістограми.

– initFromTo(double, double, int). Цей метод можна використовувати для об'єктів класу Histo. Два дійсних числа, що першими передаються в якості параметрів, визначають межі гістограми, а ціле число визначає кількість інтервалів.

Для занесення даних у гістограми найчастіше використовуються такі методи, що можна використовувати для об'єктів обох класів:

– add(double). Цей метод додає у гістограму число, що передається до методу в якості параметру.

– addFrequencyForValue(double, double). В якості першого параметра задається вага переданого випадкового числа, а другим передається саме випадкове число. У простих випадках вага може дорівнювати 1. Саме так реалізовано метод add(double).

Для відображення результатів обробки накопичених даних можна використовувати такі методи:

– showRelFrec(diagram). Використовується для виведення результатів у вигляді стовпчастої діаграми. До методу як параметр передається посилання на об'єкт класу Diagram, де буде відображатися діаграма. Можлива й інша модифікація методу, з більшим числом параметрів і, відповідно, більшими можливостями.

– toString(). Використовується для виведення результатів обробки накопичених даних у вигляді тексту з таблицею відносних частот.

6.2.3 *Класи для графічного відображення результатів моделювання*

Ці класи розглянуті у підрозділі 2.5.

Рисунок 6.5 – Вигляд компоненту класу Diagram

Методи setGridByX(int), setGridByY(int) використовуються для налаштування масштабної сітки на діаграмі.

Методи setHorizontalMinText(String), setHorizontalMaxText(String), setVerticalMinText(String), setVerticalMaxText(String) використовуються для налаштування діапазону діаграми по горизонтальній та вертикальній висях.

Метод setTitle(String) використовуються для налаштування заголовку діаграми.

Метод clear() очищає діаграму.

Зображення геометричних фігур на діаграмі виконуються за допомогою об'єкту класу Painter, що входить до складу об'єктів класу Diagram. З його допомогою можна зображувати лінії, прямокутники, овали. Колір ліній налаштовується. Особливість цих об'єктів полягає в тому, що після зображення лінії, вони пам'ятають своє становище на діаграмі, що спрощує зображення графіків

Об'єкти класу Painter можуть існувати і незалежно від діаграми, а мати тільки посилання на неї. Завдяки цьому, кілька об'єктів класу Painter можуть використовувати одну й ту ж діаграму, що дозволяє зображати на одній діаграмі кілька графіків.

Метод placeToXY(float, float) використовуються для переміщення пера у точку із заданими відносно діаграми координатами.

Метод drawToXY(float, float) використовуються для проведення лінії у задану точку на діаграмі.

Метод drawDependency(…) можна використовувати для виведення графіків. Як параметри в останній метод передається два масиви, які задають координати точок графіка, колір графіка і параметр логічного типу, який вказує, чи слід налаштовувати діаграму під дані графіка.

Метод drawBarsDiagram(…) цього класу можна використовувати для виведення стовпчастих діаграм. Як параметри в метод передається масив границь інтервалів по горизонтальній вісі, масив значень по вертикалі, ширина стовпчика в долях ширини інтервалу, зсув стовпчика від лівої межі проміжку і параметр логічного типу, який вказує, чи потрібна зміна налаштування діаграми під дані.

Метод drawNeedleDiagram(…) цього класу можна використовувати для відображення голчастої діаграми. Перелік параметрів цього методу майже такий самий як і у попереднього.

На діаграмах також можна відображати гістограми. Але ці методи викликаються для об’єктів-гістограм, а посилання на діаграму у ці методи передається як параметр.

6.2.4 *Клас MultiActor*

Цей клас використовується для створення груп однакових об'єктів (бригад), що моделюють багатоканальну обробку в СМО. При такому обслуговуванні група паралельно працюючих приладів працює з однією чергою. Для налаштування об’єкту цього класу потрібно визначити значення двох атрибутів:

nClone – визначає кількість копій.

original – містить посилання на зразок, з якого будуть зроблені копії.

Для цього використовуються методи setNumberOfClones(int) та setOriginal(Actor)

Клас успадковує клас Actor і у ному визначено метод rule(). Під час виконання правил дії об’єкта створюються копії оригіналу шляхом клонування і ці копії одразу записуються у стартовий список диспетчера.

Для клонування використовується метод clone() класу Actor. Тут слід зазначити, що операція клонування в Java є операцією поверхневого копіювання (shallowCopy). При такому копіюванні створюється копія області пам'яті, що містить інформацію про об'єкт. Внаслідок цього ми отримуємо копії значень для примітивних типів і типу String, а для решти класів отримуємо не копії об’єктів, а копії посилань на об'єкти. Таким чином, поля всіх клонованих об'єктів посилаються на ті самі об'єкти, що визначаються зразком.

При клонуванні акторів, що забезпечують багатоканальну обробку, такий спосіб клонування нас майже влаштовує. Всі клони будуть посилатися на ту саму чергу, на той самий генератор випадкових чисел, у всіх буде той самий диспетчер. Імена клонів можна робити різними. Єдиний недолік поверхневого копіюванні в тім, що всі клони будуть мати той самий семафор (об'єкт класу Semaphore). Для того, щоб усунути цей недолік, у класі Actor перевизначений базовий метод clone(), у якому для кожного клону створюється власний об'єкт класу Semaphore.

Цей метод слід перевизначити і у спадкоємців класу Actor, якщо вони мають посилання на об’єкти, що мають бути унікальними для кожної копії.

6.2.5 *Клас QueueForTransactions*

Клас використовується для моделювання черг у системах масового обслуговування. Для збереження об’єктів, що потрапляють до черги, клас має поле queue типу ArrayList, але робота з цим об’єктом забезпечується через методи класу QueueForTransactions.

Об'єкти цього класу можуть відображати свій стан у вигляді діаграми, використовуючи об'єкти класу paint.Painter, накопичувати інформацію про поточні розміри черги у об’єкті класу DiscretHisto, а також виводити інформацію про зміну розмірів до протоколу роботи моделі.

Для виконання цих завдань черга має мати посилання на диспетчера, під керівництвом якого працює імітаційна модель, щоб отримувати значення поточного часу.

Окрім того, для відображення стану черги на діаграмі, цій черзі слід передати посилання на об'єкт класу paint.Painter, за допомогою методу setPainter(Painter). А цей об'єкт класу paint.Painter, у свою чергу, має мати посилання на якусь діаграму.

А для накопичення статистичних даних про розмір черги їй слід передати посилання на об'єкт класу DiscretHisto за допомогою методу setDiscretHisto(DiscretHisto).

Для об’єктів даного класу використовується також поняття максимального розміру, вище якого черга заповнена бути не може. У випадку спроби додавання об'єкта до заповненої черги формується подія

QueueOverflowEvent, але об'єкт не додається. Доступ до неприйнятої події можна отримати через об’єкт класу QueueOverflowEvent, створивши слухача даної події.

Перед початком роботи моделі черга повинна бути проініціалізована. Для цього використовується метод init(). Під час ініціалізації черга очищається, а об'єкт «painter», якщо він існує, і існує діаграма, на якій буде зображуватися черга, переводиться у положення з координатами 0,0.

Зміна розмірів черги відбувається за допомогою методів addLast(Object), remove(Object), removeFirst().

Особливість методу addLast(Object) полягає в тому, що в ньому аналізується ступінь заповнення черги й, у випадку неможливості її подальшого збільшення, формується подія QueueOverflowEvent.

Особливість методів додавання й видалення об'єктів полягає в тому, що перед зміною черги та після її зміни викликається один з наступних методів - beforeAdd, beforeRemove, afterAdd, afterRemove, залежно від виконаної операції.

Методи beforeAdd, beforeRemove виводять до протоколу диспетчера інформацію про новий розмір черги, а також забезпечують відображення стану черги перед її зміною, та накопичення інформації у гістограмі, якщо визначені необхідні для цього об’єкти. Ці методи мають специфікатор доступу protected і тому можуть бути довизначені у спадкоємців класу.

6.2.6 *Клас Store*

Цей клас схожий на попередній. Різниця полягає у тому, що об’єкти цього класу накопичують значення типу double у полі size. Так само як і у попередньому класі можливо відображення поточного значення size на діаграмі і накопичення цих значень у гістограмі, тільки гістограма має бути типа Histo. Перед початком роботи об’єкт слід проініціалізувати.

6.3 Методика побудови моделі СМО

Сучасні технології проектування програмних систем рекомендують створювати програмний продукт, як поєднання декількох шарів. Для побудови застосування, що забезпечує моделювання СМО, можна рекомендувати використовувати шар подання (presentation layer), шар моделі та шар компонентів.

Схематичне зображення проекту для моделювання у вигляді взаємопов’язаних шарів представлено на рисунку 6.6.

Рисунок 6.6 - Структура застосування для моделювання СМО

6.3.1 *Шар подання*

За звичай першим шаром програмного продукту є шар подання. У нашому випадку це не щось інше, як графічний інтерфейс користувача, що є посередником між користувачем і самою моделлю. Основні завдання цього шару такі:

– отримання від користувача налаштувань компонент моделі;

– надання доступу до компонент, з налаштуваннями користувача;

– забезпечення запуску моделі;

– надання засобів для динамічної індикації процесу моделювання;

– надання засобів для відображення результатів моделювання.

У випадку використання фреймворку Simulation для побудови моделі рекомендовано у шарі подання реалізувати такі завдання:

– створення диспетчера;

– створення моделі;

– передачу моделі посилання на диспетчера;

– передачу моделі посилання на інтерфейс користувача;

– відображення результатів моделювання, отриманих від моделі.

Таким чином, шар подання реалізується візуальним класом з необхідним набором візуальних елементів.

Перед початком моделювання у класі має бути створений диспетчер та модель. Створення моделі перед кожним її запуском дещо уповільнює роботу програми, тому що заново створюються усі об’єкти, але це спрощує програмування.

Для створення моделі слід використовувати фабрику моделей, що має реалізувати інтерфейс IModelFactory. Інколи без використання фабрики можна обійтись, але це є обов’язковим, якщо у програмному додатку будуть використовуватися компоненти, що забезпечують автоматизоване проведення серій багаторівневих експериментів з моделлю або дослідження перехідних процесів.

Для запуску моделі використовується метод диспетчера start().

Відображення результатів моделювання, отриманих від моделі після завершення її роботи, можна пов’язати із подією DispatcherFinishEvent, створивши відповідного слухача цієї події. Інший варіант розв’язання цієї задачі полягає у тому, щоб після запуску моделі створити потік виведення результатів моделювання і приєднати його до потоку диспетчера за допомогою методу join().

Взагалі, для побудови інтерфейсу користувача достатньо стандартних засобів, що надає Java, але використання компонентів фреймворку Simulation спрощує цю задачу.

На етапі проектування шару подання першочергову увагу слід приділити визначенню публічних методів класу (або інтерфейсу), що будуть надавати доступ до компонентів моделі, що містять налаштування користувача. Це дозволить доручити розробку класу одному з членів команди.

6.3.2 *Шар моделі*

Модель створює і містить у собі посилання на усі об’єкти, що моделюють складові частини досліджуваної системи і засоби для накопичення статистичної інформації, а також має створювати ці об’єкти, передавати їм посилання на себе та візуальну частину, і надавати доступ до результатів моделювання у шар подання.

Нижче наведено перелік основних завдань цього шару:

– створити усі необхідні на момент старту об’єкти моделі;

– створити засоби для накопичення статистичної інформації;

– надати публічний (або пакетний) доступ до компонент моделі;

– передати об’єктам моделі посилання на модель та шар подання;

– завантажити «акторів» моделі до стартового списку диспетчера;

– надавати шару подання доступ до результатів моделювання.

Шар моделі має декілька особливостей.

Перша особливість полягає у тому, що незважаючи на те, що модель нібито весь час працює, насправді це найбільш пасивний елемент. Модель - це перш за все сховище посилань на моделі складових частин системи. У моделі має бути також метод ініціалізації, що викликається перед запуском диспетчера.

Тобто методи моделі працюють тільки на початку моделювання та після його завершення, а в процесі моделювання виконуються методи складових її частин, що утворюють третій шар програмної системи.

До складу шару моделі будемо відносити і фабрику моделей, незважаючи на те, що ця фабрика створюється і використовується у шарі подання. Відповідно до вимог інтерфейсу IModelFactory у фабриці має бути реалізований усього один метод – createModel(Dispatcher). Клас фабрики має передавати моделі окрім посилання на диспетчера ще й посилання на візуальну частину. Для досягнення цієї мети слід використати конструктори фабрики та моделі.

Можна рекомендувати таку послідовність програмної реалізації моделі:

Визначити поля класу, що відповідають усім складовим моделі (акторам, чергам, гістограмам та іншим складовим) без створення відповідних об’єктів. Слід також передбачити поля для посилань на диспетчера та візуальну частину, хоча ці компоненти не створюються у моделі. Зважаючи на те, що тільки модель відповідає за створення цих об’єктів, доступ до всіх складових не слід робити публічним.

Визначити конструктор для моделі, через параметри якого передавати посилання на диспетчера та візуальну частину. Пустий конструктор краще не визначати. Це унеможливить створення моделі без посилань на диспетчера та візуальну частину. У конструкторі, окрім визначення посилань на диспетчера та візуальну частину можна завантажити усіх акторів до стартового списку диспетчера. Для цього доцільно реалізувати метод componentsToStart, у якому передати диспетчеру усіх акторів, що мають почати роботу після старту моделі, але для звернення до акторів слід використовувати методи get…().

Створити об’єкти для складових частин моделі. Цю частину класу доцільно реалізувати за методикою візуального редактора Eclipse, використовуючи технологію відкладеного створення об’єктів. Відповідно до цієї методики об’єкт створюється під час першого звернення до нього, у методі get…(), після аналізу, чи дорівнює посилання на цей об’єкт null. Більшість цих методів має бути публічними, для того, щоб об’єкти мали можливість спілкуватися між собою через модель.

Для створення акторів доцільно використовувати конструктор з передачею імені актора та посилань на візуальну частину і модель.

У методах get…() для компонентів, що не мають конструкторів з параметрами, після створення об’єкту, йому слід передати усі необхідні посилання через методи set…(). Зокрема для черг, це можуть бути посилання на диспетчера та гістограми. Для об’єктів типу MultiActor це посилання на оригінал та визначення кількості клонів.

6.3.3 *Шар компонентів*

Цей шар складається з класів, що реалізують моделі складових частин досліджуваної системи.

Основні завдання цього шару такі:

– моделювати складові частини системи;

– налагодити через модель зв’язки з усіма необхідними компонентами;

– реалізувати правила дії активних компонент моделі;

– передавати дані про роботу до накопичувачів статистики;

– надавати дані для динамічної індикації процесу моделювання.

Особливість цього шару полягає у тому, що до його складу входять як класи, що потребують створення, так і вже існуючі класи, які надає користувачеві фреймворк Simulation та Java.

Найчастіше, під час реалізації цього шару доводиться створювати класи для активних компоненти системи, які мають успадковувати клас Actor і у цих класах має бути реалізований метод rule(). Для реалізації черги можна використовувати колекції, але краще використовувати класи черг фреймворку Simulation. Теж саме стосується і класів для накопичення статистичної інформації.

6.4 Приклад побудови програмної системи для моделювання

Розглянемо проект, реалізація якого знаходиться в пакеті qusystem.rgr.

Система, що підлягає моделюванню, складається з одного бульдозера, навантажувача та кількох самоскидів. Бульдозер порціями згрібає ґрунт у купу. На одну порцію ґрунту він витрачає випадковий час Бульдозер припиняє свою роботу, якщо розмір купи збільшується до деякого критичного розміру і відновлює роботу тільки після того, як розмір купи стає удвічі меншим за критичний розмір.

Навантажувач починає працювати коли у купі є ґрунт і є самоскид, що чекає завантаження. Він порціями насипає ґрунт з купи у кузов самоскиду. Місткість кузову самоскида декілька порцій ґрунту. Самоскиди виїжджають із автопарку и стають у чергу до навантажувача. Після завантаження самоскиди відвозять ґрунт замовнику, розвантажуються, і повертаються до навантажувача.

Потрібно створити Java застосування для імітаційного моделювання робіт з видобування ґрунту, яке дозволить:

налаштування параметрів моделі, а саме: закони розподілення для випадкових величин продуктивності навантажувача і бульдозера, та часу перебування самоскида у дорозі кількість самоскидів, місткість кузову, критичний розмір купи, тривалість моделювання;

проведення тестових запусків моделі при різних налаштуваннях з динамічною індикацією розміру купи, довжини черги самоскидів до навантажувача, кількості самоскидів у дорозі, та виведення протоколу роботи моделі під час тестових запусків;

проведення експериментів для отримання статистичних характеристик для довжини черги самоскидів до навантажувача;

6.4.1 *Аналіз системи*

Аналізуючи наведений опис системи, перш за все можна виділити таку абстракцію, як «купа ґрунту». Характеристикою цієї абстракції є її розмір. Відповідно до технічного завдання необхідно забезпечити динамічну індикацію зміни цього параметру у часі. Абстракція «купа ґрунту» у системі безпосередньо пов'язана з двома іншими абстракціями системи – «бульдозером» та «навантажувачем».

Абстракція «бульдозер» моделює робочу машину, що додає до купи порції ґрунту, збільшуючи таким чином її розмір. Одноразова порція ґрунту в реальній системі, звичайно, є випадковою величиною. Але купа ґрунту накопичує ці порції, тобто інтегрує їх, і таким чином зменшує вплив коливань розміру порції. Тому, для спрощення моделі будемо вважати, що порція ґрунту має стале значення і дорівнює одиниці. Тобто одиницею виміру кількості ґрунту у системі буде середнє значення порції ґрунту, що бульдозер додає до купи за один раз. На видобування порції ґрунту бульдозер витрачає деякий час, що є випадковою величиною. Найбільш вірогідно, що ця величина підпорядковується закону Ерланга.

Абстракція «навантажувач» моделює робочу машину, що навантажує ґрунт у самоскиди, беручи його з купи. Будемо вважати, що розмір порції, що навантажувач бере з купи за один раз дорівнює одинці. На завантаження порції ґрунту навантажувач витрачає деякий час, що є випадковою величиною. Найбільш вірогідно, що ця величина підпорядковується нормальному закону. Зрозуміло, що навантажувач може працювати коли є ґрунт у купі і самоскид, що чекає завантаження. Коли ж ці умови не виконуються, навантажувач чекає на їх виконання. Абстракція «навантажувач» у системі безпосередньо пов'язана з абстракцією «купа ґрунту» та абстракцією «черга самоскидів на завантаження».

Абстракція «самоскид» моделює авто, що перевозить ґрунт від купи до місця призначення. У системі може працювати одразу декілька самоскидів. Самоскид може везити декілька порцій ґрунту. На доставку ґрунту та повернення до навантажувача самоскид витрачає деякий час, що є випадковою величиною. Найбільш вірогідно, що ця величина підпорядковується закону Ерланга. Абстракція «самоскид» у системі може перебувати у одному з трьох станів – дорога до навантажувача, чекання на завантаження, дорога до місця вивантаження ґрунту. Для цих станів має бути передбачена динамічна індикація.

Абстракція «черга самоскидів до навантажувача» моделює чергу самоскидів, що чекають завантаження. Самоскиди стають у цю чергу, коли під'їжджають до навантажувача. Цю абстракцію використовує навантажувач, щоб визначити, чи є самоскид, що потребує завантаження. Відповідно до завдання динаміка зміни цієї черги у часі має бути відображена на візуальній частині застосування.

Окрім того, ця черга має накопичувати інформацію про свої розміри для отримання статистичних характеристик довжини черги до навантажувача. Вирішення цієї проблеми покладемо на абстракцію «накопичувач інформації про довжину черги до навантажувача»

Абстракція «самоскиди у дорозі» моделює групу самоскидів, що знаходяться у дорозі. Це віртуальна черга, що необхідна для динамічної індикації кількості самоскидів, що знаходяться у дорозі.

До складу абстракцій додамо також таку абстракція, як модель системи у цілому, що об'єднує інші абстракції і є об'єктом експериментального дослідження під час моделювання.

Окрім того нам будуть потрібні диспетчер, що забезпечує керування процесом моделювання, фабрика моделей та графічний інтерфейс користувача.

Результати першого кроку аналізу системи по визначенню ключових абстракцій, наведені у таблиці 6.1.

Таблиця 6.1 – Абстракції системи, що підлягає моделюванню

Абстракція	Перелік завдань
Купа ґрунту	Накопичувати порції ґрунту від бульдозера. Відображати на діаграмі зміни розміру купи у часі
Бульдозер	Передавати через випадкові інтервали часу порції ґрунту до купи. Стежити за розміром купи. Якщо розмір купи більше прийнятного то припиняти роботу. Відновлювати роботу коли розмір купи зменшиться у два рази.
Навантажувач	Чекати на появу ґрунту у купі і самоскида у черзі до навантажувача. Перевантажувати потрібну кількість порцій ґрунту із купи у самоскид, затрачуючи на це випадковий час.
Самоскид	Перебувати у дорозі до навантажувача, стати у чергу до навантажувача, чекати заповнення кузову, їхати на розвантаження, повертатися знов до навантажувача.. Реєструватися у віртуальній черзі самоскидів, що їдуть.
Черга самоскидів до навантажувача	Місце, де знаходяться самоскиди, що чекають на завантаження. Забезпечити динамічну індикацію змін свого розміру у часі. Накопичувати інформацію про свої розміри.

Продовження таблиці 6.1

Накопичувач інформації про розмірів черги до навантажувача	Накопичувати інформацію про про розмір черги до навантажувача. Відображувати статистичні характеристики довжини черги у графічному та текстовому вигляді.
Самоскиди у дорозі	Динамічна індикація кількості самоскидів у дорозі до навантажувача.
Модель системи	Об'єднує інші абстракції і є об'єктом експериментального дослідження під час моделювання.

6.4.2 *Реалізація шару подання*

Основою шару подання є інтерфейс користувача, який представлено на рисунках 6.7, 6.8. Інтерфейс був створений відповідно до завдань, що були визначені вище. Інтерфейс спроектовано як сукупність трьох основних панелей.

Перша панель використовується для налаштувань користувача і присутня на екрані у всіх режимах роботи.

Друга та третя панелі розташовані одна під одною і з'являються після вибору відповідного режиму роботи.

У проекті передбачено два режими роботи. Перший режим, це режим тестування моделі із динамічною індикацією зміни розмірів черг у часі.

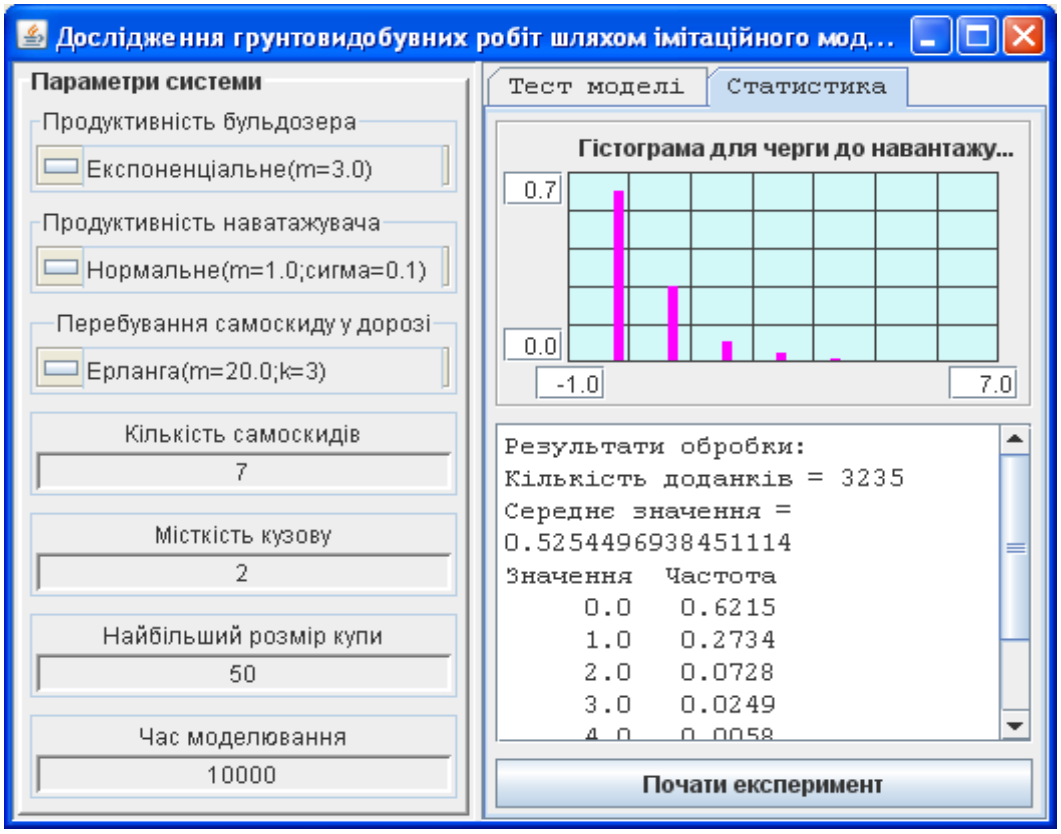

Рисунок 6.7 – Інтерфейс користувача моделі у режимі тестування

Другий режим використовується для збирання та виведення на екран статистичних даних про роботу моделі.

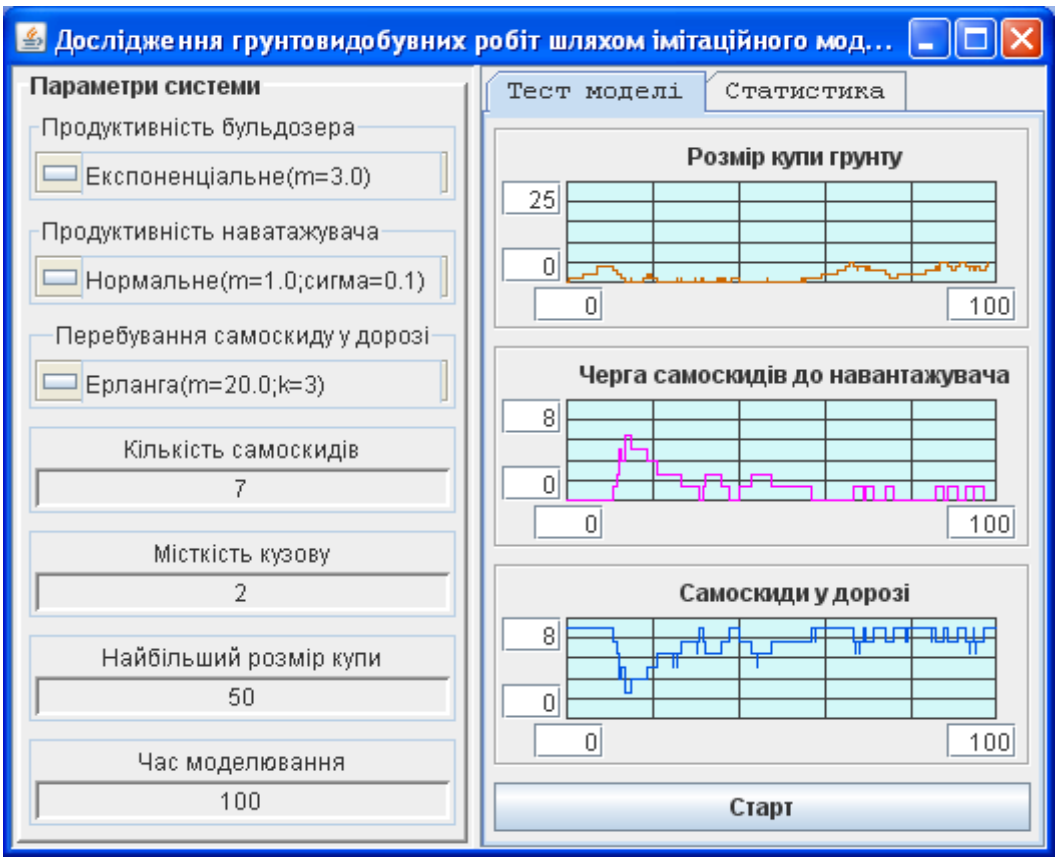

Рисунок 6.8 – Інтерфейс користувача моделі у режимі статистики

Для запуску моделі у режимі тестування використовується кнопка «Старт», з якою пов’язано виклик методу startTest(), текст якого наведено у лістингу 6.1. У методі готуються до виводу графіків діаграми, створюються фабрика моделей, диспетчер та модель. Далі модель готується до роботи у режимі тестування, після чого стартує диспетчер.

Кнопка «Старт» блокується на період моделювання, а її розблокування реалізується у окремому потоці, що затримується за допомогою метода join() до завершення потоку диспетчера.

Лістинг 6. 1 - Метод запуску процесу моделювання у режимі тестування

```
private void startTest() {
getDiagramHeepSize().clear();
getDiagramQueueToLoader().clear();
getDiagramLorryOnRoad().clear();
FactoryForRgrModel factory = new FactoryForRgrModel(RgrGUI.this);
final Dispatcher dispatcher = new Dispatcher();
RgrModel model = factory.createModel(dispatcher);
model.initForTest();
dispatcher.start();
getJButtonStart().setEnabled(false);
```

```
        //Створюємо потік, що чекає завершення роботи диспетчера,
        //а після цього відновлює  доступність кнопки "Старт"
        new Thread() {
            public void run() {
                //Затримка до кінця моделювання
                try {
                    dispatcher.getThread().join();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                getJButtonStart().setEnabled(true);
        }.start();
    }
}
```

Для запуску моделі у режимі накопичення статистичної інформації використовується кнопка «Почати експеримент» закладки «Статистика», з якою пов’язано виклик методу startStat(), текст якого наведено у лістингу 6.2. на початку методу візуальна частина готується для виведення статистичної інформації. Потім у методі створюються фабрика моделей, диспетчер та модель. Далі модель готується до роботи у режимі накопичення статистики, після чого стартує диспетчер.

У методі також створюється додатковий потік, що чекає завершення потоку диспетчера, а після того виводить на екран статистичні характеристики черги у вигляді діаграми та тексту. У цьому ж потоці розблокується кнопка «Почати експеримент».

Лістинг 6. 2 - Метод запуску процесу моделювання у режимі тестування

```
private void startStat() {
        getDiagramHistoQueueToLoader().clear();
        getJTextPaneAvto().setText("");
        FactoryForRgrModel factory = new FactoryForRgrModel(RgrGUI.this);
        final Dispatcher dispatcher = new Dispatcher();
        final RgrModel model = factory.createModel(dispatcher);
        model.initForStat();
        dispatcher.start();
        getJButtonStat().setEnabled(false);
```

```
    //Створюємо потік, що чекає завершення роботи диспетчера
    //а після цього відновлює  доступність кнопки "Почати експеримент"
    //та виводить на екран статистичні характеристики черги
    new Thread() {
       public void run() {
          try {
             dispatcher.getThread().join();
          } catch (InterruptedException e) {
             e.printStackTrace();
          }
          getJButtonStat().setEnabled(true);
          model.getHistoForQueueToLoader().showRelFrec(
                getDiagramHistoQueueToLoader());
          getJTextPaneAvto().setText(
                model.getHistoForQueueToLoader().toString());
          getJTextPaneAvto().select(0, 0);
       }
    }.start();
 }
```

Ще одна важлива функція шару подання – надання доступу до своїх компонент іншим класам. Перелік відповідних публічних методів наведено на рисунку 6.9.

Рисунок 6.9 – Публічний інтерфейс шару подання

6.4.3 *Реалізація шару моделі*

До складу цього шару входять два класи – клас фабрики моделей та клас моделі.

6.4.3.1 **Клас Factory**

Завдання об’єктів цього класу – дозволити створення моделі і передачу їй посилання на диспетчера за допомогою методу createModel(Dispatcher), що передбачений інтерфейсом IModelFactory. Окрім того, моделі має бути передано посилання на візуальну частину. Тому конструктор фабрики створено таким чином, щоб отримати відповідне посилання і запам’ятати його.

Сама модель створюється за допомогою конструктора з двома параметрами. Текст класу фабрики наведено у лістингу 4.3.

Лістинг 6. 3 - Клас фабрики моделей

```
public class Factory implements IModelFactory {
   private GUI gui;


   public Factory(GUI gui) {
      this.gui = gui;
   }


   public Model createModel(Dispatcher dispatcher) {
      Model newModel = new Model(dispatcher, gui);
      return newModel;
   }
```

6.4.3.2 **Клас Model**

Клас модель побудовано виходячи з того, що модель буде створюватися перед кожним її запуском. Це значно спрощує програмування і підвищує його надійність, тому що при цьому усі компоненти моделі теж створюються заново і не потребують ініціалізації.

Перелік публічних методів моделі, що утворюють її інтерфейс наведено на рисунку 6.10.

Для створення моделі використовується конструктор з двома параметрами, який забезпечує гарантовану передачу моделі посилань на візуальну частину і диспетчера. Конструктор наведено у лістингу 6.4.

Конструктор забезпечує також передачу акторів моделі до стартового списку диспетчера.

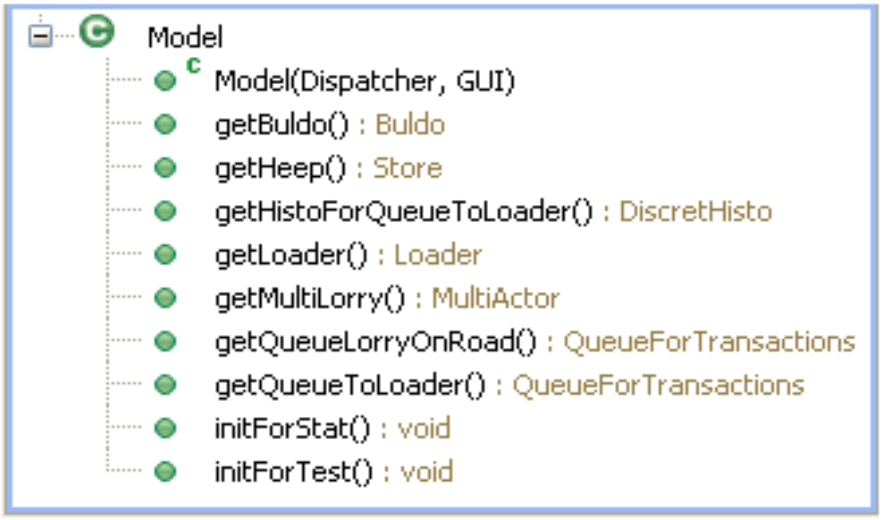

Рисунок 6.10 – Публічний інтерфейс моделі

Лістинг 6. 4 - Конструктор моделі

```
public Model(Dispatcher d, GUI g) {
    if (d == null || g == null) {
        System.out.println("Не визначено диспетчера або GUI для Model");
        System.out.println("Подальша робота неможлива");
        System.exit(0);
    }
    dispatcher = d;
    gui = g;
    //Передаємо акторів до стартового списку диспетчера
    componentsToStartList();
}
```

Такий конструктор можна вважати стандартним для виконання РГР або курсового проекту. Натомість, зміст методу componentsToStartList() залежіть від конкретного завдання. У нашому випадку цей метод має вигляд, представлений у лістингу 6.5.

Лістинг 6. 5 - Метод componentsToStartList()

```
public void componentsToStartList() {
    // Передаємо акторів диспетчеру
    dispatcher.addStartingActor(getBuldo());
    dispatcher.addStartingActor(getLoader());
    dispatcher.addStartingActor(getMultiLorry());
}
```

Слід звернути увагу на те, що для звертання до акторів використовуються методи get…(), у яких реалізовано відкладене створення об’єктів.

Як приклад методу для створення актора наведемо метод getBuldo(), лістинг 6.6. Для створення усіх акторів ми будемо використовувати однаковий підхід, що полягає у використанні конструктора з параметрами, які передають посилання на візуальну частину та на модель. Маючи ці посилання, створений об’єкт отримує доступ до інформації, що потрібна йому для функціонування.

Лістинг 6. 6 - Метод відкладеного створення об’єкту бульдозер

```
public Buldo getBuldo() {
    if (buldo == null) {
        buldo = new Buldo("Бульдозер", gui, this);
    }
    return buldo;
}
```

Об’єкти класу MultiActor створюються дещо інакше. Такому об’єкту потрібно передати посилання на зразок, що буде клонуватися, та задати кількість клонів. Наводимо тут метод створення бригади авто, лістинг 6. 7.

Лістинг 6. 7 – Метод відкладеного створення бригади самоскидів

```
public MultiActor getMultiLorry() {
    if (multiLorry == null) {   multiLorry = new MultiActor();
        multiLorry.setNameForProtocol("MultiActor для бригади самоскидів");
        multiLorry.setOriginal(new Lorry("Самоскид", gui, this));
        multiLorry.setNumberOfClones(gui.getChooseDataNLorry().getInt());
    }
    return multiLorry;
}
```

Об’єкти для черг створюються також інакше. У класі На відміну від акторів, об’єкти для черг створюються дещо інакше. У класі QueueForTransactions відсутні конструктори з параметрами. Тому посилання, що необхідні чергам слід передавати через методи set…(). Для прикладу наведемо метод getQueueToLoader(), лістинг 6. 8.

Лістинг 6. 8- Метод відкладеного створення черги до навантажувача

```
public QueueForTransactions getQueueToLoader() {
    if (queueToLoader == null) {
        queueToLoader = new QueueForTransactions();
        queueToLoader.setNameForProtocol("Черга до навантажувача");
```

```
            queueToLoader.setDispatcher(dispatcher);
            queueToLoader.setDiscretHisto(getHistoForQueueToLoader());
            queueToLoader.init();
        }
        return queueToLoader;
    }
```

Аналогічно створюються об’єкти для гістограм.

Завершується робота над класом моделі створенням методів ініціалізації моделі для можливих режимів роботи. Ці методи налаштовують модель до вимог конкретного режиму. Так у методі initForTest() діаграмам передаються посилання на об’єкти класу Painter, що забезпечує відображення цих черг на гістограмах. У методі initForStat() блокується виведення протоколу на консоль.

У інших режимах може виникнути потреба реалізації методів деяких інтерфейсів. Якщо методи ініціалізації з цих інтерфейсів передають якісь налаштування компонентам моделі, то ці налаштування доцільно перенести до шару подання звідки компоненти їх можуть отримати.

6.4.4 *Реалізація компонентів моделі*

Класи, що потребують реалізації у даному проекті – це класи для бульдозера, навантажувача та клас для самоскида. Як приклад, розглянемо реалізацію класу Buldo.

6.4.4.1 **Клас Buldo**

Об’єкти цього класу представляють абстракцію бульдозер.

Головне завдання цієї абстракції – додавати порції ґрунту до купи через випадкові інтервали часу. Якщо розмір купи збільшується до критичного розміру, бульдозер зупиняється і відновлює роботу, коли розмір купи зменшиться удвічі. Ці дії бульдозер має виконувати впродовж усього часу моделювання для будь якого режиму роботи моделі.

Для роботи бульдозеру необхідні дані, перелік яких наведено у таблиці 6.2.

Таблиця 6.2 – Поля класу Buldo

Назва поля	Клас	Призначення поля	Джерело
heap	Store	Посилання на купу ґрунту	Model
heapMaxSize	double	Критичний розмір купи	GUI
finishTime	double	Час моделювання	GUI
rnd	rnd.Randomable	Генератор випадкових чисел	GUI

Перелічені у таблиці поля ініціалізуються безпосередньо у конструкторі (лістинг 6.9), через посилання на візуальну частину та модель.

Лістинг 6. 9 - Конструктор для створення бульдозера

```
public Buldo(String name, GUI gui, Model model) {
    setNameForProtocol(name);
    heap = model.getHeep();
    heapMaxSize = gui.getChooseDataHeepMaxSize().getDouble();
    finishTime = gui.getChooseDataFinishTime().getDouble();
    rnd = gui.getRndBuldo();
}
```

Правила дії бульдозера схематично можна представити у вигляді діаграми діяльності, рисунок 6.11.

Реалізація правил дії у методі rule() наведена у лістингу 6.10.

На початку методу створіється об'єкт heapHalfSize. Це об'єкт анонімного класу, що реалізує інтерфейс IWaitCondition, і містить умову, яку бульдозер використовує для відновлення роботи після зупинки, пов'язаної із збільшенням розміру купи до критичного. Цей об'єкт доцільно створити один раз, до початку циклу. Далі у методі реалізовано цикл виконання правил дії, відповідно до рисунку 6.11.

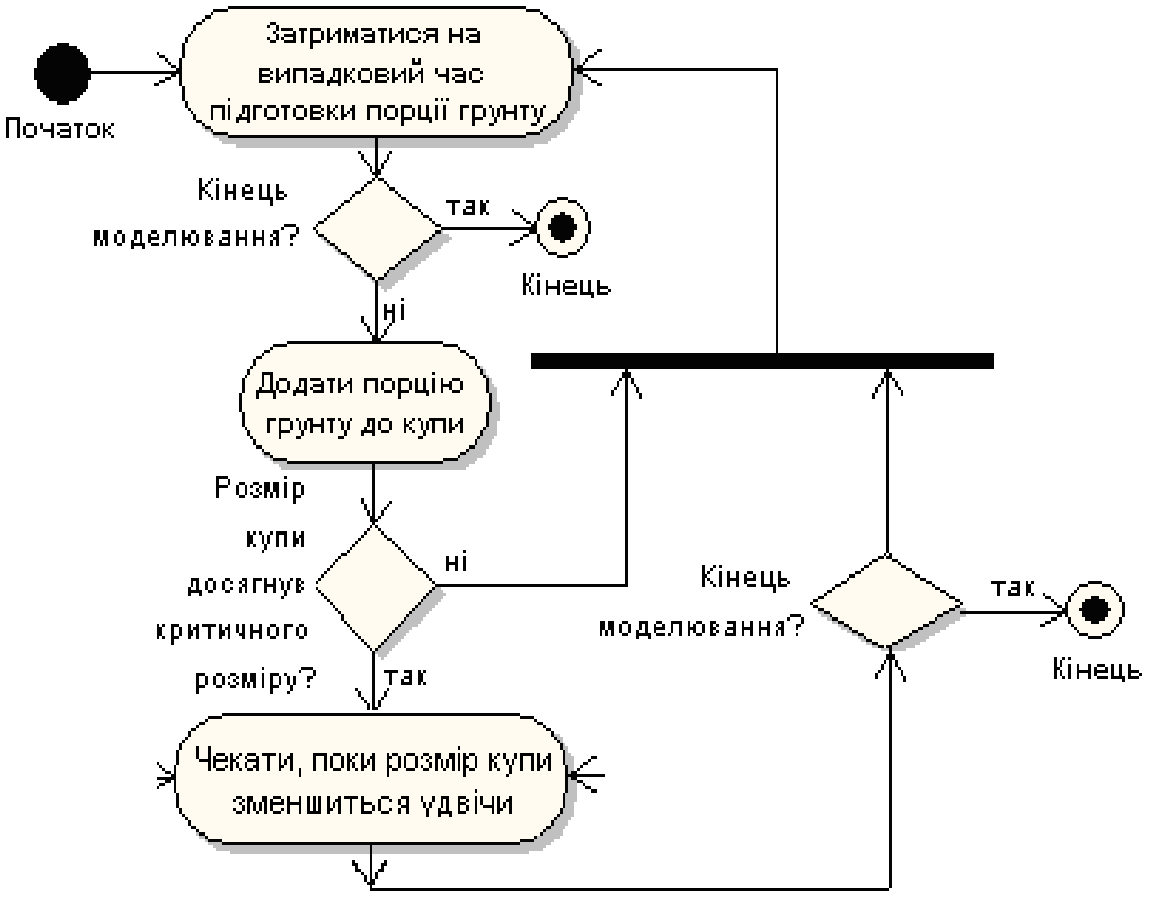

Рисунок 6.11 – Діаграма діяльності об'єкта класу Buldo

Лістинг 6. 10 - Правила дії бульдозера

```
protected void rule() {
    // Умова для методу WaitForConition()
    IWaitCondition heapHalfSize = new IWaitCondition() {
        public boolean testCondition() {
            return heap.getSize() <= heapMaxSize / 2;
        }
        public String toString() {
            return ("поки купа зменшіться удвічи");
        }
    };
    // Цикл правил дії бульдозера
    while (true) {
        // Затримка на час формування порції грунту
        holdForTime(rnd.next());
        // Можливо після цього вже настав час закінчити роботу
        if (getDispatcher().getCurrentTime() > finishTime)
            break;
        // Збільшення розміру купи на одну порцію
        getDispatcher().printToProtocol(
                "  " + getNameForProtocol() + " додає порцію грунту.");
        heap.add(1);
        // Зупинка,якщо купа досягла критичного розміру
        if (heap.getSize() >= heapMaxSize) {
            try {
                waitForCondition(heapHalfSize);
            } catch (DispatcherFinishException e) {
                return;
            }
        }
    }
}
```

КОНТРОЛЬНІ ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Які різновиди системи масового обслуговування ви знаєте?
2. Які основні ознаки СМО?
3. Які параметри СМО визначаються шляхом моделювання?
4. Які компоненти фреймворка можна використати для побудови візуальної частини програми?
5. Наведіть публічні методи об’єктів класу ChooseData.
6. Наведіть публічні методи об’єктів класу ChooseRandom.
7. Наведіть публічні методи об’єктів класу Diagram.
8. Наведіть публічні методи об’єктів класу Painter.
9. Наведіть публічні методи об’єктів класу Histo.
10. Наведіть публічні методи об’єктів класу DiscretHisto.
11. Чим відрізняються об’єкти класу QueueForTransaction від звичайних колекцій типу List.
12. У яких випадках замість класу QueueForTransaction доцільно використовувати клас Store?
13. Коли доцільно використовувати клас MultiActor?
14. Яких налаштувань потребує клас MultiActor?
15. Які завдання має вирішувати шар подання?
16. Які завдання має вирішувати шар моделі?
17. Які завдання має вирішувати шар компонентів моделі?
18. Які переваги надає використання фабрики моделей?

7 ТЕМА 6. ДОСЛІДЖЕННЯ НАЙПРОСТІШОЇ СМО

План лекції

– Знайомство з методикою аналітичного дослідження характеристик найпростішої СМО.

– Експериментальне визначення характеристик найпростішої СМО шляхом моделювання.

7.1 Аналітичне дослідження СМО

Аналітичні рішення знайдені тільки для деяких класів СМО. Як правило, це системи, у яких випадкові процеси підкоряються експоненціальному закону розподілу. Такі системи називаються марківськими. Цю назву системи одержали тому, що розрахунок їх можна проводити, використовуючи ланцюги Маркова. Цінність аналітичних рішень для імітаційного моделювання полягає в тому що, маючи теоретичні значення параметрів системи, ми можемо оцінити, наскільки точні результати будуть одержані шляхом моделювання.

7.1.1 *Ланцюг Маркова для простішої СМО*

Ланцюг Маркова для найпростішої СМО представлено на рисунку 7.1.

Станам системи відповідають вершини цього графа, номера яких відповідають кількості заявок у системі. Стан 0 - це стан, коли в системі немає заявок, і обслуговуючий прилад чекає. Імовірність того, що система перебуває в цьому стані, позначимо P0. Стан 1 - це стан, коли в системі одна заявка, і вона обслуговується. Довжина черги при цьому, також як і в попередньому стані, дорівнює 0. Імовірність цього стану позначимо P1. Стан 2 - це стан, коли в системі дві заявки, одна обслуговується а друга чекає в черзі. Довжина черги при цьому дорівнює 1. Імовірність цього стану позначимо P2, і так далі.

Ребра графа визначають можливі переходи системи з одного стану в інший. У кожний момент часу відбувається тільки одна подія. Якщо генератор додає заявку в чергу, то система переходить у наступний за номером стан. Якщо обслуговуючий прилад закінчує обробку заявки, то система переходить у попередній стан.

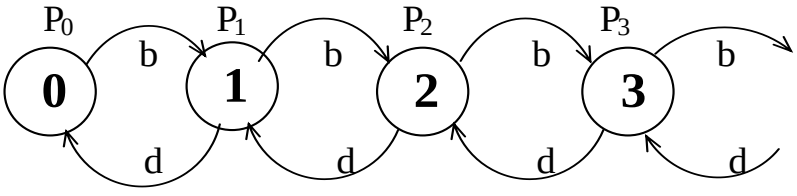

Рисунок 7.1 – Ланцюг Маркова для найпростішої СМО

Кожному з ребер відповідає деяка ймовірність відповідного переходу. Процес переходів, представлений на рисунку 7.1, у науковій літературі іноді називають процесом розмноження й загибелі. Імовірність b (birth) залежить від

інтенсивності генератора, а ймовірність d (death) - від продуктивності обслуговуючого приладу.

7.1.2 *Визначення ймовірностей станів системи в сталому режимі*

Стан ланцюга Маркова характеризується значеннями ймовірностей станів. У початковому стані системи ймовірність стану 0 дорівнює 1, а ймовірності інших станів нульові. У процесі роботи системи ці ймовірності змінюються, але згодом їх значення стабілізуються й приймають деякі постійні значення. Такий режим роботи системи називається сталим. У сталому режимі перехід з одного стану в інший відбуваються таким чином, що ймовірності станів не змінюються.

Здатність ланцюгів Маркова з часом переходити у сталий режим роботи називають «ергодичною властивістю».

Аналізуючи сталий режим у ланцюзі Маркова, можна визначити сталі значення ймовірності кожного стану системи. Для цього записуються рівняння балансу ймовірностей, засновані на такому твердженні: «якщо ймовірність стану не міняється, то ймовірність переходу системи в цей стан дорівнює ймовірності її виходу із цього стану».

Розглянемо рівняння балансу для нульового стану, виходячи із графа, представленого на рисунку 5.1.

Імовірність того, що система залишить стан 0, дорівнює $\mathbf{P_0 b}$, а ймовірність того, що система повернеться у стан 0, дорівнює $\mathbf{P_1 d}$.

Рівняння балансу ймовірностей для нульового стану запишеться так: $\mathbf{P_0 b = P_1 d}$, звідки одержимо $\mathbf{P_1 = b/d\ P_0}$.

Можна показати, що відношення **b/d** дорівнює відношенню інтенсивності надходження заявок у СМО до інтенсивності їхнього обслуговування. Це відношення в літературі зазвичай позначають грецькою буквою **ω** і називають коефіцієнтом завантаження системи. Тоді $\mathbf{P_1 = \omega P_0}$.

Розглянемо тепер перший стан.

Імовірність того, що система залишить цей стан, дорівнює $\mathbf{P_1 b + P_1 d}$, а ймовірність того, що вона в нього повернеться, буде $\mathbf{P_0 b + P_2 d}$.

Умова балансу ймовірностей буде така: $\mathbf{P_1 b + P_1 d = P_0 b + P_2 d.}$ З огляду на те, що $\mathbf{P_1 d = P_0 b}$, одержимо: $\mathbf{P_1 b = P_2 d,}$ **або** $\mathbf{P_2 = \omega P_1}$.

Підставивши в останню рівність $\mathbf{P1 = \omega P0}$, одержимо $\mathbf{P2 = \omega 2 P0}$

Записуючи умови балансу для наступних станів, одержимо для них аналогічні співвідношення. У загальному випадку можна записати

$$P_i = \omega^i \cdot P_0 \qquad (7.1)$$

Варто також враховувати, що в будь-який момент часу система повинна обов'язково перебувати в якому-небудь зі станів, отже, сума всіх ймовірностей дорівнює одиниці.

$$\sum_{i=0}^{\infty} P_i = 1 \tag{7.2}$$

З огляду на співвідношення 7.1 і 7.2, можна одержати рівняння для визначення $\mathbf{P_0}$

$$\sum_{i=0}^{\infty} \omega^i \cdot P_0 = 1 \tag{7.3}$$

Сума в лівій частині рівняння являє собою суму елементів геометричної прогресії і за умови $\boldsymbol{\omega < 1}$ дорівнює $\mathbf{P_0/(1-\omega)}$. Звідси маємо:

$$P_0 = 1 - \omega \tag{7.4}$$

З огляду на співвідношення 7.1 і 7.4, можна одержати рівняння для визначення ймовірності будь-якого стану системи в сталому режимі.

$$P_i = (1-\omega) \cdot \omega^i \tag{7.5}$$

7.1.3 *Визначення ймовірностей появи черг різної довжини.*

Якщо система перебуває в станах 0 або 1, то черга відсутня, тому що в нульовому стані в системі зовсім немає заявок, а в першому - єдина заявка яка обслуговується, і черга порожня.

Отже, імовірність появи черги нульової довжини дорівнює сумі ймовірностей першого й другого стану.

$$P_{q=0} = P_0 + P_1 = (1-\omega) + (1-\omega)\cdot\omega = 1 - \omega^2 \tag{7.6}$$

Якщо розглянути деякий і-ий стан, то для нього черга буде мати довжину і-**1**, тому ймовірність появи черги довжиною **k**, буде дорівнювати ймовірності **k+1** стану.

$$P_{q=k} = (1-\omega) \cdot \omega^{k+1} \tag{7.7}$$

Маючи формулу 7.7 можна визначити ймовірність появи черги, довжина якої не менша ніж k

$$P_{q \geq k} = \sum_{i=0}^{\infty} (1-\omega)\cdot\omega^{k+1+i} = (1-\omega)\cdot\omega^{k+1} \sum_{i=0}^{\infty} \omega^i = \omega^{k+1} \tag{7.8}$$

Із формули 7.8 можна визначити ймовірність появи черги, довжина якої менша ніж k.

$$P_{q \angle k} = 1 - \omega^{k+1} \tag{7.9}$$

7.1.4 *Визначення середньої довжини черги*

Знаючи ймовірності появи черги будь-якої довжини, ми можемо визначити середню довжину черги по наступній формулі:

$$\bar{q} = \sum_{k=1}^{\infty} k \cdot P_{q=k} \tag{7.10}$$

Підставивши в цю формулу ймовірність появи черги довжиною k, яка визначається по співвідношенню 7.7, одержимо

$$\bar{q} = \sum_{k=1}^{\infty} k(1-\omega) \cdot \omega^{k+1} = (1-\omega)\omega \sum_{k=1}^{\infty} k\omega^{k} \tag{7.11}$$

З огляду на те, що

$$\sum_{k=0}^{\infty} k \cdot \omega^{k} = \omega + 2\omega^2 + 3\omega^3 + 4\omega^3 + \ \ldots \ =$$

$$= \omega + \omega^2 + \omega^3 + \omega^4 + \ \ldots \ + \omega^2 + \omega^3 + \omega^4 + \ \ldots \ + \omega^3 + \omega^4 + \ \ldots \ =$$

$$= \omega(1 + \omega + \omega^2 + \omega^3 + \ldots) \ + \omega^2(1 + \omega + \omega^2 + \omega^3 + \ldots) \ + \omega^3(1 + \omega + \omega^2 + \omega^3 + \ldots) \ + \ldots =$$

$$= (1 + \omega + \omega^2 + \omega^3 + \ldots) \cdot (\omega + \omega^2 + \omega^3 + \ldots) =$$

$$= \omega(1 + \omega + \omega^2 + \omega^3 + \omega^4 + \ldots)^2 =$$

$$= \omega / (1-\omega)^2,$$

одержимо

$$\bar{q} = (1-\omega) \cdot \omega \cdot \frac{\omega}{(1-\omega)^2} = \frac{\omega^2}{1-\omega} \tag{7.12}$$

7.1.5 *Визначення середнього часу очікування в черзі*

Середній час очікування в черзі можна знайти, використовуючи так званий результат Літла.

Результат Літла говорить, що середню кількість заявок у черзі для марківської СМО можна знайти як добуток інтенсивності потоку на вході та середнього часу перебування заявок у черзі.

$$\bar{q} = \lambda \cdot \bar{\tau} \tag{7.13}$$

Із цього співвідношення можна знайти вираз для визначення середнього часу очікування в черзі

$$\bar{\tau} = \bar{q} / \lambda \tag{7.14}$$

Підставивши в співвідношення 7.14 вираз для середньої довжини черги 7.12, одержимо формулу 7.15 для визначення середнього часу очікування в черзі.

$$\bar{\tau} = \frac{\omega^2}{\lambda \cdot (1-\omega)} = m\frac{\omega^2}{1-\omega} \qquad (7.15)$$

7.2 Дослідження СМО шляхом моделювання

На жаль, аналітичні методи можна використовувати тільки для дослідження нескладних марківських систем.. У інших випадах дослідження доводиться проводити шляхом моделювання.

Визначимо вимоги до програми, за допомогою якої буде проводитися моделювання.

Насамперед, робота моделі має бути візуалізована, тобто дослідник повинен мати можливість спостерігати за тим, як поводить себе модель. Звичайно для візуалізації на екран виводиться інформація про поточні значення часу моделювання, розміри черг та стану об'єктів. Ця інформація може виводитися на екран або в цифровому вигляді, або у вигляді переміщень різного роду покажчиків. Обидва способи мають свої плюси й мінуси, але варто враховувати, що аналогову інформацію людина сприймає краще. Можна використовувати і мультиплікацію. Найчастіше використовується динамічна індикації розмірів черг у вигляді часових діаграм.

Крім візуалізації модель повинна видавати користувачеві статистичну інформацію про параметри, які його цікавлять.

Реальна модель повинна створюватися із урахуванням цілей моделювання. При цьому варто використовувати базові класи фреймврку і на їхній основі, використовуючи механізм наслідування, створювати підкласи, які забезпечують вирішення поставлених задач.

Нижче розглядається програмне застосування, що дозволить експериментально визначити характеристики найпростішої СМО.

7.2.1 *Опис проекту*

У проекті досліджується проста марківська система масового обслуговування. До складу моделі входять генератор заявок, черга й обслуговуючий прилад. Система проектувалася виходячи з того, що вона повинна забезпечити:

– виведення діаграми зміни довжини черги у часі;

– накопичення, обробку й видачу (у вигляді гістограми й тексту) статистичної інформації про довжину черги;

– накопичення, обробку й видачу (у вигляді гістограми й тексту) статистичної інформації про час очікування заявок у черзі;

– забезпечувати налаштування параметрів моделі - тривалість моделювання, максимальний розмір черги, статистичні характеристики потоку заявок і часу обслуговування.

Проект було реалізовано з використанням технології проектування, що була розглянута у попередньому розділі..

7.2.2 *Клас testTheory.GUI*

Клас testTheory.GUI реалізує перший шар програми і визначає візуальну частину застосування, що оформлена, як набір сторінок (об'єкти класу javax.swing.JTabbedPane).

Основна сторінка інтерфейсу користувача, яку видно на рисунку 7.2, забезпечує налаштування, запуск моделі й динамічну індикацію довжини черги на обслуговування.

Запуск моделі відбувається при натисканні кнопки «Старт».

Після натискання кнопки створюються диспетчер і фабрика моделей, далі створюється модель і їй передається посилання на візуальну частину та диспетчера. Під час створення моделі усі її актори заносяться до стартового списку диспетчера. Модель запускається викликом методу start диспетчера.

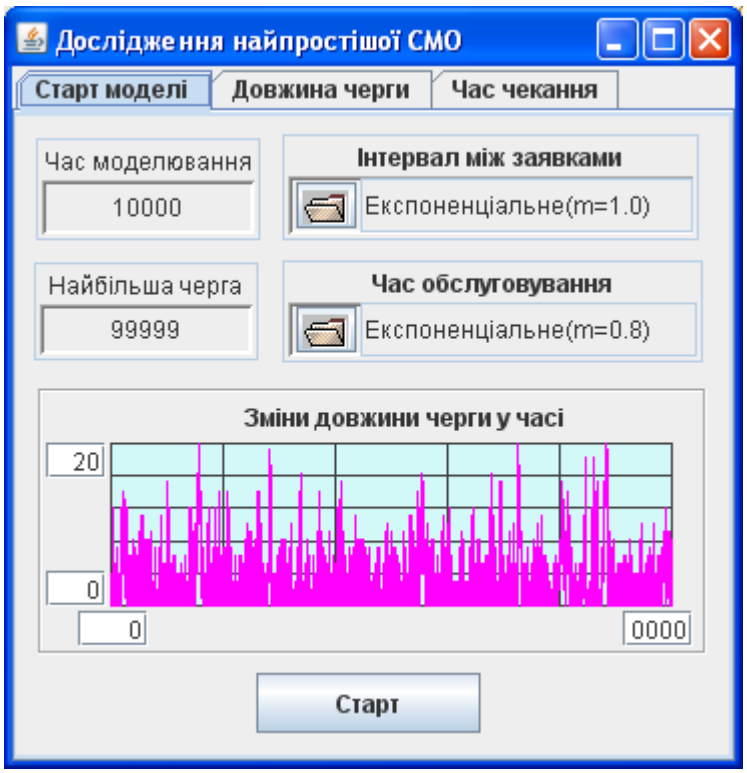

Рисунок 7.2 – Вигляд основної сторінки проекту

Після запуску моделі виконання методу, пов’язаного з кнопкою «Старт», затримується до завершення роботи диспетчера. Далі метод відновлює свою роботу і забезпечує виведення статистичної інформації, накопиченої у процесі моделювання, на наступні сторінки візуальної частини.

Відповідний метод наведено нижче, у лістингу 7.2.

Лістинг 7.2 – Метод, пов’язаний з кнопкою «Старт»

```
private void start(){
    getDiagramShow().clear();
    Dispatcher dispatcher=new Dispatcher();
```

```
        Factory factory=new Factory(this);
        Model model=factory.createModel(dispatcher);
        getJButtonStart().setEnabled(false);
        dispatcher.addDispatcherFinishListener( new DispatcherFinishListener(){
        public void onDispatcherFinish(){
           getJButtonStart().setEnabled(false);
           getJTextAreaQueue1().setText(model.getDiscretHisto().toString());
           model.getDiscretHisto().showRelFrec(getDiagramQueue());
           getJTextAreaTime1().setText(model.getHisto().toString());
           model.getHisto().showRelFrec(getDiagramTime());
        });
        dispatcher.start();
}
```

Сторінки «Довжина черги» та «Час чекання» основної форми використовуються для виводу статистичної інформації про довжину черги й час очікування в черзі. Кожна зі сторінок пов'язана зі своєю гістограмою і при відкритті сторінки дані з гістограми відображаються у вигляді тексту й стовпчастої діаграми. Вид сторінок представлений на рисунку 7.3.

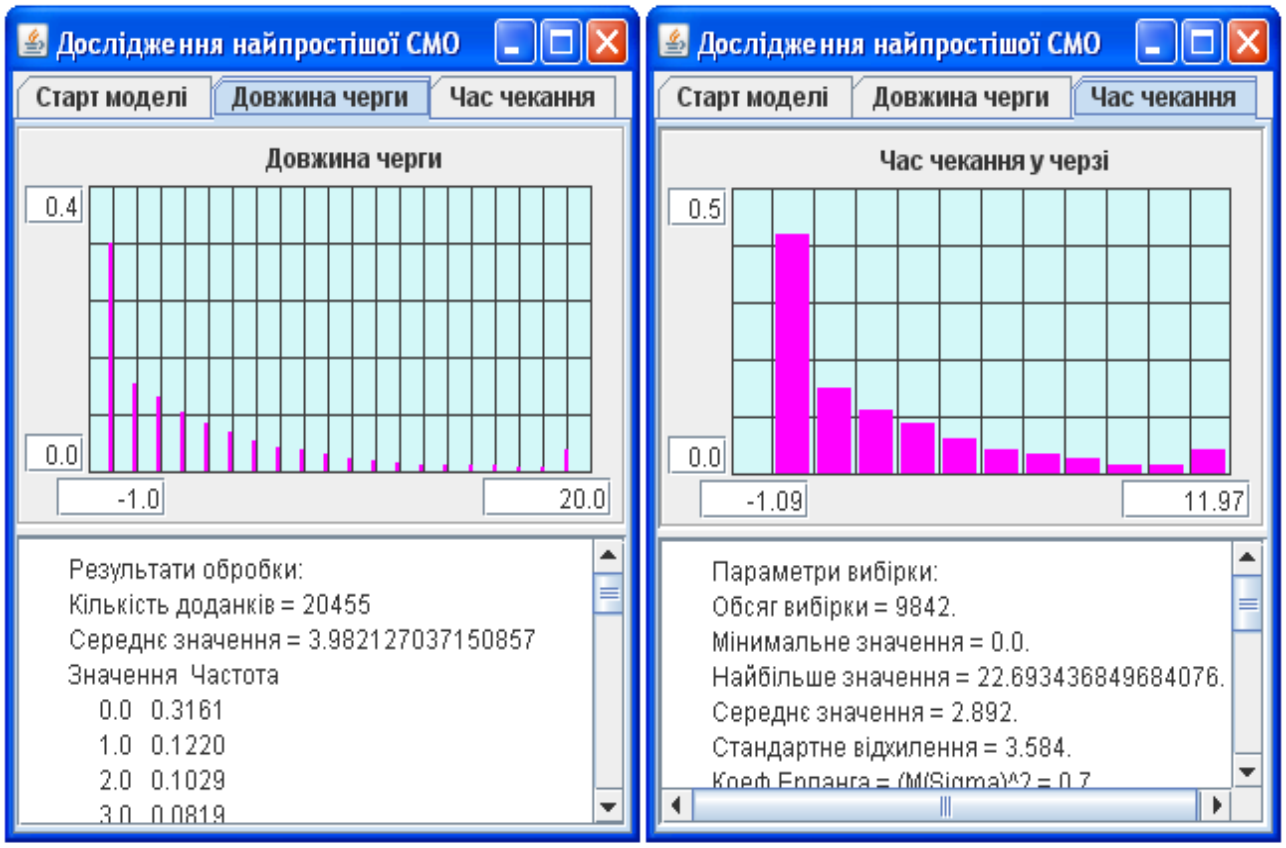

Рисунок 7.3 – Сторінки проекту із статистичною інформацією

Взаємодія візуальної частини з іншими об’єктами реалізується через публічні методи класу, представлені на рисунку 7.4.

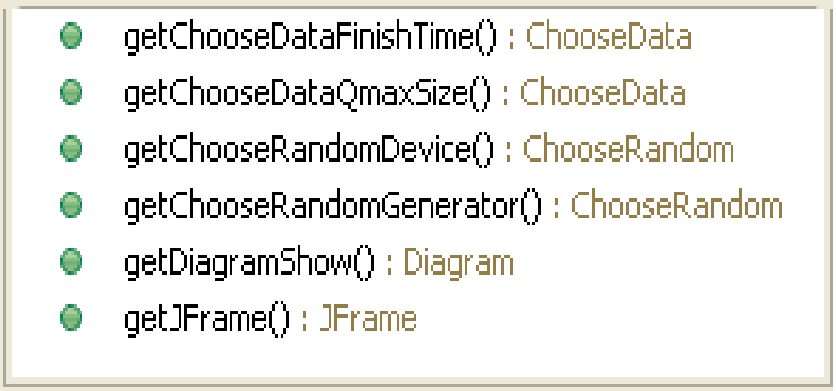

Рисунок 7.4 – Публічні методи шару подання

7.2.3 *Модель досліджуваної системи*

Модель досліджуваної системи об’єднує усі компоненти моделі, що забезпечують її роботу та накопичення статистичної інформації. Модель складається з наступних об'єктів:

– генератор заявок - об'єкт transactGenerator, екземпляр створеного у проекті класу TransactGenerator;

– черга заявок на обслуговування - об'єкт queue, екземпляр класу QueueForTransactions;

– обслуговуючий прилад - об'єкт device, екземпляр створеного у проекті класу Device.

– гістограма для збору інформації про довжину черги - об'єкт discretHisto, екземпляр класу stat.DiscretHisto;.

– гістограма для збору інформації про час очікування в черзі - об'єкт histo, екземпляр класу stat.Histo;.

Перелік публічних методів моделі, що утворюють її інтерфейс наведено на рисунку 7.5.

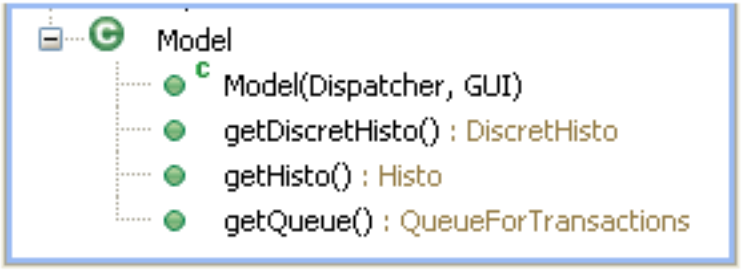

Рисунок 7.5 – Публічний інтерфейс моделі

Структура моделі така сама, як і структура моделі розглянутої раніше.

Для створення моделі використовується фабрика, що реалізує інтерфейс IModelFactory. Завдання фабрики – створення моделі і передача їй посилань на диспетчера та візуальну частину. Диспетчер передається за допомогою методу createModel(Dispatcher), що передбачений інтерфейсом IModelFactory. А для передачі посилання на візуальну частину використовується конструктор

фабрики, який створено таким чином, щоб отримати відповідне посилання і запам’ятати його. Сама модель створюється за допомогою конструктора з двома параметрами, посиланнями на диспетчера та графічний інтерфейс користувача

Конструктор забезпечує також передачу акторів моделі до стартового списку диспетчера, використовуючи приватний метод componentsToStartList(). залежіть від конкретного завдання. Для звертання до акторів використовуються методи get…(), у яких реалізовано відкладене створення відповідних об’єктів

Для створення усіх акторів використовується однаковий підхід, що полягає у використанні конструктора з параметрами, які передають посилання на візуальну частину та на модель. Маючи ці посилання, створений об’єкт отримує доступ до інформації, що потрібна йому для функціонування.

На відміну від акторів, об’єкти для черга створюються дещо інакше. У класі QueueForTransactions відсутні конструктори з параметрами. Тому посилання, що необхідні чергам слід передавати через методи set…().

Аналогічно створюються об’єкти для гістограм.

7.2.4 *Класи акторів*

7.2.4.1 **Клас theorStudy.TransactGenerator**

Цей клас призначений для створення об'єктів, які генерують заявки. Об’єкт цього класу затримується на випадковий час, потім створює заявку і ставить її у чергу на обслуговування. Заявка створюється у вигляді числа, значення якого дорівнює поточному значенню часу моделювання, тобто часу появи заявки. Це дозволить обслуговуючому приладу визначити, як довго заявка чекала на обслуговування.

Діаграма діяльності об’єктів класу представлена на рисунку 7.6.

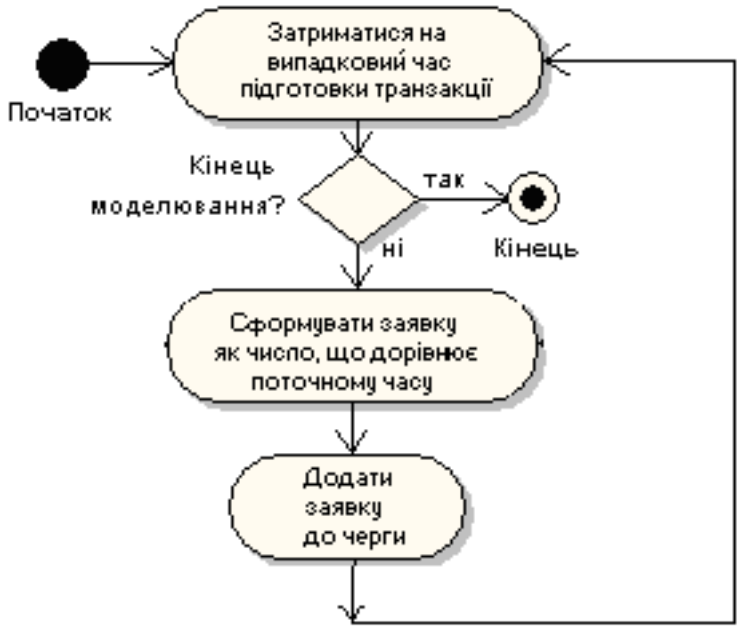

Рисунок 7.6 – Діаграма діяльності генератора заявок

Текст класу teorStudy.TransactGenerator наведено нижче, у лістингу 7.5.

Лістинг 7.5. – Клас генератора заявок

```
public class TransactGenerator extends Actor {
   // Тривалість роботи генератора
   private double finishTime;
   // Черга заявок
   private QueueForTransactions queue;
   // Генератор часу, що витрачається на створення транзакції
   private Randomable rnd;
   //Коструктор, у якому ініціалізуються усі поля класу
   //через доступ до моделі та візуальної частини
   public TransactGenerator(String name, GUI gui, Model model) {
      setNameForProtocol(name);
      finishTime = gui.getChooseDataFinishTime().getDouble();
      rnd = gui.getChooseRandomGenerator();
      queue = model.getQueue();
   }
   public void rule() {
      while (true) {
         holdForTime(rnd.next());
         if (getDispatcher().getCurrentTime() > finishTime) {
            break;
         }
         getDispatcher().printToProtocol(
               "  " + getNameForProtocol() + " створює транзакцію.");
         queue.addLast( getDispatcher().getCurrentTime());
      }
   }
}
```

7.2.4.2 **Клас theorStudy.Devices**

Даний клас моделює роботу обслуговуючого приладу системи. Об’єкт цього класу чекає, поки у черзі з’явиться заявка, вилучає її з черги і після цього затримується на деякий час, імітуючи процес обробки. Окрім того, цей обєкт

забезпечує обчислення й реєстрацію у гістограмі часу очікування у черзі для кожної заявки.

Діаграма діяльності об’єкту класа представлена на рисунку 7.7.

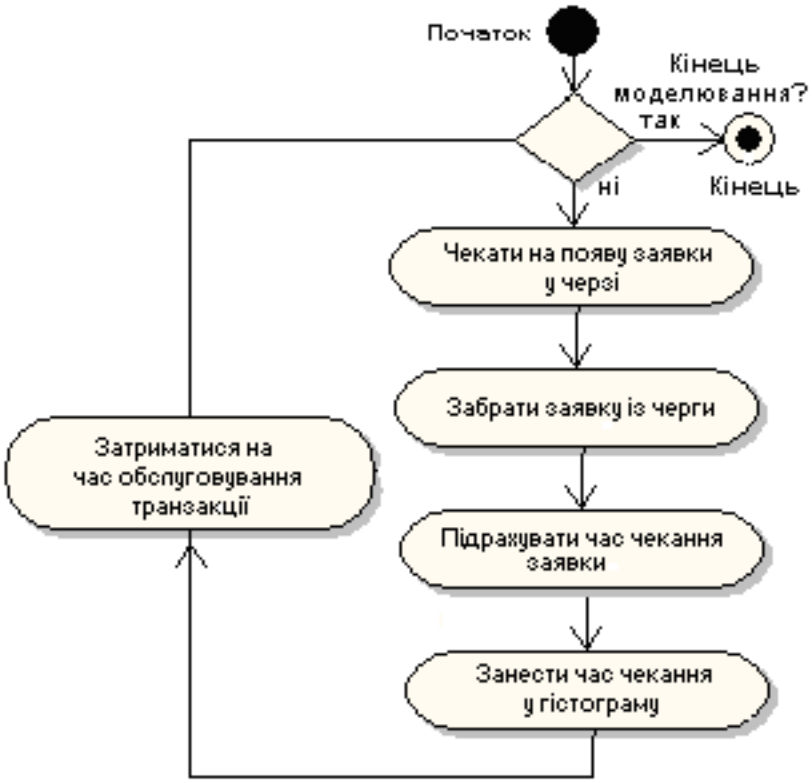

Рисунок 7.7 – Діаграма діяльності обслуговуючого пристрою

Текст класу teorStudy.Device наведено нижче, у лістингу 7.6.

Лістинг 7.6. – Клас обслуговуючого пристрою

```
public class Device extends Actor {
   // Тривалість роботи обслуговуючого пристрою
   private double finishTime;
   // Черга заявок
   private QueueForTransactions queue;
   // Генератор часу, що витрачається на обслуговування транзакції
   private Randomable rnd;
   // Гістограма для накопичення часу чекання у черзі
   private Histo histo = null;
   // Коструктор, у якому ініціалізуються усі поля класу
   // через доступ до моделі та візуальної частини
   public Device(String name, GUI gui, Model model) {
      setNameForProtocol(name);
```

```
        finishTime = gui.getChooseDataFinishTime().getDouble();
        rnd = gui.getChooseRandomDevice();
        queue = model.getQueue();
        histo= model.getHisto();
    }
    public void rule() {
        // Створюємо умову, виконання якої буде чекати актор
        IWaitCondition queueSize = new IWaitCondition() {
            public boolean testCondition() {
                return queue.size() > 0;
            }
            public String toString() {
                return "поки у " + queue.getNameForProtocol()
                        + " з'явиться транзакція";
            }
        };
        // цикл виконання правил дії
        while (getDispatcher().getCurrentTime() <= finishTime) {
            try {
                waitForCondition(queueSize);
            } catch (DispatcherFinishException e) {
                return;
            }
            double bornTime = (Double) (queue.removeFirst());
            double waitTime = getDispatcher().getCurrentTime() - bornTime;
            histo.add(waitTime);
            holdForTime(rnd.next());
        }
    }
}
```

7.2.5 *Статистична інформація про довжину черги*

Для збору статистичної інформації про довжину черги в даному проекті використовується об’єкт queue класу QueueForTransactions, який моделює чергу

в системі. Цей об’єкт забезпечує і динамічну індикацію зміни своїх розмірів на діаграмі. Для накопичення статистичної інформації об’єктові queue було передано посилання на об'єкт discretHisto класу DiscretHisto. Об’єкт queue перед кожною зміною розміру черги передає до об'єкту discretHisto свій розмір та час, на протязі якого цей розмір не змінювався.

Програмно це реалізовано через методи beforeAdd() і beforeRemove(), які викликаються перед зміною розміру черги. Ці методи викликають приватний метод accum(), що передає дані у гістограму через метод addFrequencyForValue(double frequency, double value), frequency – це різниця між поточним часом і моментом попередньої зміни черги, а value - це довжина черги.

Для зберігання значення моменту часу, пов'язаного з попередньою зміною черги, використовується приватна змінна екземпляра lastTime. Очищення черги й ініціалізація змінної lastTime відбувається в методі init().

Текст методу accum() наведений у лістингу 7.7.

Лістинг 7.7 - Метод накопичення інформації про довжину черги

```
private void accum() {
   if (discretHisto != null) {
      discretHisto.addFrequencyForValue(
         dispatcher.getCurrentTime() - lastTime,
         size());
      lastTime = dispatcher.getCurrentTime();
   }
}
```

У класі DiscretHisto ці дані накопичуються спочатку у колекції. Якщо розмір колекції перевищує 500, або процес накопичення закінчився, у класі створюється два масиви. Один для унікальних значень параметру value, другий для сум відповідних значень frequency. Таким чином дані value опиняються у елементах масиву arrayForValue, а відповідніі інтервали часу накопичуються в елементах масиву arrayForAbsolutFrequency.

Якщо після цього значення value перевищує максимальне із значень, заданих у масиві arrayForValue, то переданий інтервал накопичується в останньому елементі масиву. Для пошуку середньої довжини черги в класі DiscretHisto введені змінні totalInterval і totalIntegral. У першій з них накопичується загальна сума відрізків часу які заносяться в гістограму. Наприкінці моделювання тут буде величина, що дорівнює всьому періоду накопичення інформації. У другий змінній накопичується інтеграл від довжини черги за часом, значення якого визначається шляхом множення поточної довжини черги на ширину тимчасового інтервалу між поточним часом і часом попередньої події. Наприкінці моделювання, щоб оцінити ймовірність появи кожного із значень масиву arrayForValue кожен з елементів

масиву arrayForAbsolutFrequency ділиться на значення змінної totalInterval. А розділивши значення змінної totalIntegral на значення змінної totalInterval, одержуємо середню довжину черги.

7.2.6 *Статистична інформація про час очікування в черзі*

У рішенні цієї задачі бере участь декілька компонентів моделі. Як ми бачили раніше, заявка, що створюється об'єктом класу theorStudy.TransactGenerator, являє собою час її створення. Отже, від генератора заявок, через чергу, до обслуговуючого приладу, передається час появи заявки, яка обслуговується.

У класі theorStudy.Device, який використовується для створення об'єкта, що моделює роботу обслуговуючого приладу, забезпечується обчислення часу очікування в черзі. Час очікування визначається як різниця між поточним часом і значенням змінної, що вилучена з черги. Ця інформація передається об'єкту класу Histo за допомогою методу add(double). У цьому об’єкті дані спочатку просто накопичуються у колекції. Коли їх кількість перевищує 500, ці дані аналізуються, визначається мінімальне та максимальне значення для даних. Після цього визначається кількість інтервалів і гістограма ініціалізується. Завершується цей етап перенесенням даних з колекції до гістограми. Надалі, нові дані заносяться безпосередньо до гістограми.

КОНТРОЛЬНІ ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Які системи масового обслуговування иожна досліджувати аналітичними методами?
2. Що означає номер стану у ланцюзі Маркова.
3. У яких станах ланцюга маркова для СМО з одним обслуговуючим приладом черга дорівнює 0?
4. Що характерно для сталого режиму ланцюга маркова?
5. Запішить рівняння балансу ймовірностей для стану з номером n>0.
6. Чому дорівнює сума ймовірностей усіх станів ланцюга Маркова?
7. Що таке коефіцієнт завантаженння СМО з одним обслуговуючим пристроєм?
8. Що буде відбуватися у СМО, якщо коефіцієнт завантаження більше одиниці?
9. Запишіть формулу для ймовірності перебування системи у стані з номером n.
10. Як знайти ймовірність відсутності черги у найпростішій СМО?
11. Як знайти ймовірність появи черги довжиною k у найпростішій СМО?
12. Як знайти середню довжину черги у найпростішій СМО?
13. Що стверджує результат Літла?
14. Як знайти середній час перебування у черзі для найпростішій СМО?
15. Як в моделі накопичується інформація про час перебування у черзі?
16. Як в моделі накопичується інформація про довжину черги?

8 ТЕМА 8. ОДНОФАКТОРНІ ЕКСПЕРИМЕНТИ З ІМІТАЦІЙНИМИ МОДЕЛЯМИ

План лекції

– Одно факторні експерименти на одному рівні.
– Довірчий інтервал.
– Планування багаторівневих одно факторних експериментів.
– Дисперсійний аналіз результатів експериментів.
– Регресійний аналіз результатів експериментів.
– Компоненти фреймворку Simulation для побудови моделі СМО.

8.1 Короткі теоретичні відомості про планування однофакторних експериментів та обробку їх результатів

Зазвичай при моделюванні проводиться не один, а багато експериментів. Це пов'язано, в першу чергу, з тим, що результат модельного експерименту величина випадкова і тому потребує оцінки точності, а точність отриманих даних можна оцінити лише в тому випадку, якщо ми маємо у розпорядженні вибірку випадкових величин. Крім того, моделювання часто проводиться з метою знайти залежність деякої характеристики досліджуваної системи від факторів, які впливають на неї. Для цього доводиться проводити серії експериментів при різних значеннях факторів. Чим більше експериментів в одній серії і чим більше різних значень факторів, які аналізуються, тим точніше дослідник може оцінити залежність, яка його цікавить. З іншого боку, чим більше експериментів, тим більше витрат на їх проведення. Тому при моделюванні доводиться вирішувати задачу планування експериментів для того, щоб досягти бажаного результату з мінімальними затратами.

Планування експерименту передбачає вирішення таких задач:

– визначення значень факторів, для яких проводяться серії експериментів;

– визначення кількості експериментів в одній серії;

Під час обробки отриманих результатів вирішуються такі завдання:

– оцінка точності отриманих результатів;

– оцінка однорідності дисперсій;

– оцінка значимості впливу фактору;

– вибір функції регресії для отриманих результатів і визначення параметрів цієї функції;

– оцінка адекватності отриманої функції регресії результатам експерименту.

8.1.1 *Однофакторний експеримент на одному рівні*

Експерименти на одному рівні проводять з метою оцінки значення функції відгуку для даного значення фактору. Невідоме значення функції

відгуку визначається за результатами серії із n експериментів шляхом знаходження середнього значення цих результатів.

Отримане таким чином значення функції відгуку називають крапковою оцінкою, від слова крапка, тобто графічно це значення відображається крапкою.

У випадку, якщо експериментів проведено дуже багато, то отримане середнє буде представляти значення функції відгуку досить точно. Це пов'язано з тим, що дисперсія середнього у n раз менше дисперсії початкових випадкових величин, де n - кількість повторних експериментів..

При вибірках малого об'єму крапкова оцінка може значно відрізнятись від дійсного значення оцінюваної величини. З цієї причини при невеликих вибірках прийнято користуватися інтервальними оцінками. У цьому випадку значення функції відгуку визначається довірчим інтервалом, який задає діапазон, з якого може приймати значення функція відгуку, «від… до…».

Для того щоб правильно спланувати кількість експериментів одно факторного експерименту, потрібно знати, як це значення впливає на ширину довірчого інтервалу.

8.1.1.1 **Довірчий інтервал**

Довірчий інтервал визначається двома числами – кінцями інтервалу. Границі інтервалу обчислюються таким чином, щоб цей інтервал охоплював дійсне значення оцінюваної величини із заданою імовірністю.

Імовірність, з якою довірчий інтервал охоплює істинне значення величини, називають довірчою імовірністю, або надійністю.

Ширину довірчого інтервалу можна знайти, якщо відомий закон розподілення для випадкової величини середнього значення. Виходячи з того, що середнє значення визначається на підставі суми випадкових величин, що незначно відрізняються одна від одної, ця величина підпорядковується нормальному закону (теорема Ляпунова).

Дисперсію середнього значення можна знайти, обрахувавши дисперсію результатів експериментів $D_{експ}$.

$$D_{ср} = D_{експ} / n \tag{6.1}$$

Для нормального розподілення відомо, що з імовірністю γ випадкова величина попадає в діапазон

$$m \pm t_{\gamma} \cdot \sigma \tag{6.2}$$

де m – математичне очікування випадкової величини відгуку;

σ – середньоквадратичне відхилення;

γ – довірча імовірність;

t_{γ} – число, для якого подвійне значення функції Лапласа буде дорівнювати γ.

Для імовірності γ = 0.95, t_{γ} = 1.96. Для γ = 0.99, t_{γ} = 2.57. Для інших ймовірностей це число можна знайти в таблицях функції Лапласа. За звичай приймають γ = 0.95.

Таким чином, якщо знайдено середнє значення функції відгуку $y_{ср}$, і відома дисперсія результатів експерименту, то можна стверджувати, що з імовірністю γ істинне значення y знаходиться у діапазоні

$$y_{ср} - t_{\gamma} \cdot \sqrt{D_{эксп}/n} \;\; .. \;\; y_{ср} + t_{\gamma} \cdot \sqrt{D_{эксп}/n} \qquad (8.3)$$

Границі цього діапазону і задають довірчий інтервал для y.

Формула для обчислення половини ширини довірчого інтервалу набуває такого вигляду

$$\Delta_{y_{ср}} = t_{\gamma} \cdot \sqrt{D_{эксп}/n} \qquad (8.4)$$

Із цієї формули випливає, що ширина довірчого інтервалу збільшується пропорційно квадратному кореню із дисперсії експерименту і звужується пропорційно квадратному кореню із кількості експериментів. Отже, якщо ми хочемо зменшити ширину інтервалу вдвічі, потрібно провести у 4 рази більше експериментів.

Слід мати на увазі, що величину t_{γ} можна знаходити у відповідності з функцією Лапласа тільки при об'ємах вибірки більших 30. Це пов'язано з тим, що оцінка дисперсії експерименту $D_{эксп}$ по вибірці малого розміру буде неточною. Тому, при малих вибірках, для визначення t_{γ} потрібно використовувати критичні значення розподілення Ст'юдента для n-1 числа степенів свободи, рівні значимості, який дорівнює 1-γ і двосторонній критичній області. Це призводить до розширення довірчого інтервалу. Так при 95% надійності і об'ємі вибірки 30 коефіцієнт t_{γ} дорівнює 2.05, а при об'ємі вибірки 10 стає рівним 2.23. Тобто при малих об'ємах вибірки, ширина довірчого інтервалу при зменшенні числа експериментів буде збільшуватись швидше, ніж корінь квадратний із об'єму вибірки.

Таким чином, число експериментів в серії може бути вибрано виходячи з потрібної точності отриманих результатів із урахуванням дисперсії експерименту.

8.1.2 *Однофакторний багаторівневий експеримент*

8.1.2.1 Планування багаторівневого експерименту

Експерименти на декількох рівнях фактора проводять з метою визначення залежності функції відгуку від значення фактора. Перша проблема, яку при цьому необхідно вирішувати, це визначення кількості рівнів фактора, на яких буде проводитися експеримент.

При вирішенні цієї проблеми потрібно керуватись такими міркуваннями:

– з точки зору оцінки точності результатів, краще провести багато експериментів на невеликій кількості рівнів, ніж небагато експериментів на багатьох рівнях;

– кількість рівнів повинна бути не менше, ніж число невідомих коефіцієнтів функції регресії;

– перевірку на адекватність можна провести тільки в тому випадку, якщо число рівнів фактору більше, ніж число коефіцієнтів функції регресії, які необхідно визначити;

– бажано, щоб число експериментів на рівнях було однаковим.

Враховуючи ці міркування, можна рекомендувати вибирати число рівнів на 1 більше, ніж число невідомих коефіцієнтів функції регресії. Якщо ж припущень про вигляд цієї функції нема, слід починати проведення експериментів на трьох рівнях, а потім, у випадку необхідності, поступово збільшувати їх кількість.

Після вибору кількості рівнів слід вибирати значення фактора для кожного рівня. При вирішенні цієї проблеми можна керуватись наступним:

– експерименти обов'язково проводяться на краях області, що нас цікавить;

– значення фактора, що залишились, бажано рівномірно розподіляти в області, яка досліджується.

8.1.2.2 **Обробка експериментальних даних**

Після проведення експериментів слід визначити середнє значення функції відгуку y_{jcp} і дисперсії випадкової величини D_{yj} для кожного рівня.

Отримані середні значення функції відгуку для рівнів можуть бути використані в подальшому для отримання функції регресії, а значення дисперсії – для обчислення середньої дисперсії експерименту.

8.1.2.3 **Перевірка однорідності дисперсії**

Дисперсія функції відгуку на різних рівнях повинна бути приблизно однаковою. Лише в цьому випадку можна буде довіряти статистичній оцінці значимості впливу фактора на функцію відгуку і оцінці адекватності моделі.

Для того щоб оцінити, чи викликана різниця в отриманих значеннях дисперсії лише випадковими факторами, можна використовувати критерій Кочрена. Він використовується лише в тому випадку, коли число експериментів на кожному рівні однакове. В якості оцінки в цьому критерії використовується відношення максимальної дисперсії до суми всіх дисперсій. Очевидно, що це відношення не може бути більше одиниці і менше 1/p, де p – кількість рівнів. В останньому випадку всі дисперсії однакові.

Отримане значення відношення порівнюють із критичним значенням, яке можна знайти в статистичних таблицях розподілення Кочрена по кількості степенів свободи для оцінюваних дисперсій, яке дорівнює n-1 (n – кількість дослідів на рівні), кількості порівнюваних дисперсій і прийнятому рівневі значимості. Наприклад, при рівні значимості 0.05, кількості рівнів 4 і кількості експериментів на рівні 10 отримуємо критичне значення 0.5.

Якщо розраховане значення відношення менше критичного, дисперсії можна вважати однорідними.

Якщо дисперсії однорідні, то в якості оцінки точності експериментів, які проводяться, приймають їх середнє значення.

$$D_{эксп} = \sum_{j=1}^{p} D_{y_j} \Big/ p \qquad (8.5)$$

Якщо дисперсії неоднорідні, експериментатор ризикує загіпнотизувати себе числовими оцінками, які насправді помилкові. Тому потрібно спробувати зробити дисперсії однорідними. Для цього можна використати деяке перетворення функції відгуку. Досить часто допомагає логарифмічне перетворення, тобто замість рівняння регресії виду y=f1(x), використовують рівняння ln(y)=f2(x).

8.1.2.4 **Перевірка значимості фактора**

Визначивши середнє значення функції відгуку для значень фактору, які аналізуються, і знаючи дисперсію експерименту, можна оцінити, чи впливає фактор на функцію відгуку. Якщо відповідь на це питання не очевидна, проводять статистичну оцінку значимості фактора за допомогою дисперсійного аналізу.

Основна ідея дисперсійного аналізу полягає у порівнянні відхилень, які викликані змінами фактору, з відхиленнями, які викликані похибками експерименту.

Для оцінки відхилень, викликаних змінами фактору, використовується дисперсія фактору ($D_ф$), яка представляє собою дисперсію випадкових величин відхилень середніх значень функції відгуку від загального середнього.

$$D_{ф} = \frac{\sum_{i=1}^{p} (y_i^{ср} - \bar{y})^2}{p-1} \qquad (8.6)$$

де $y_{iср}$ – середнє значення функції відгуку на і-му рівні;

$\bar{y}$ – загальне середнє функції відгуку на всіх рівнях.

Число степенів свободи для дисперсії, отриманої за цією формулою, дорівнює p-1.

Вплив фактора має значення в тому випадку, якщо дисперсія фактора набагато більше дисперсії середніх значень функції відгуку, яка може бути знайдена із дисперсії експерименту $D_{эксп}$, по формулі 8.7

$$D_{y^{ср}} = \frac{D_{эксп}}{n} \qquad (8.7)$$

де n – кількість експериментів на одному рівні.

Кількість степенів свободи дисперсії, отриманої за цією формулою, така ж, як і у дисперсії експерименту, і дорівнює p*(n-1).

Для порівняння дисперсій можна використовувати критерій Фішера.

Вплив фактора суттєвий, якщо відношення дисперсії фактора до дисперсії середніх значень функції відгуку більше критичного значення критерію Фішера. Очевидно, що таку перевірку слід проводити лише в тому випадку, якщо дисперсія фактора перевищує дисперсію середніх значень. Якщо ж

дисперсія фактора менше дисперсії середніх значень, то впливу фактору на відгук нема.

8.1.2.5 **Функція регресії**

Якщо відомо, що вплив фактора є, можна спробувати представити залежність, яка нас цікавить, у вигляді деякої функції, близької до експериментальних даних. Графік цієї функції називають лінією регресії.

Вибір функції регресії досить нелегка справа і залежить від мистецтва експериментатора та його досвіду. З одного боку функція регресії не повинна бути складною – це зробить її використання більш трудомістким, з іншого боку вона повинна добре узгоджуватись з експериментальними даними.

Зручніше всього функцію регресії представляти у вигляді лінійної комбінації елементарних функцій, які залежать від величини фактора. В цьому випадку функція регресії буде мати вигляд 8.8.

$$y(x) = a_1\varphi_1(x) + a_2\varphi_2(x) + \ldots + a_n\varphi_n(x) \qquad (8.8)$$

У наведеній залежності невідомими є коефіцієнти a_1, a_2 ,… a_n . Від вибору значень цих коефіцієнтів залежить, який вплив на лінію регресії мають відповідні їм функції φ(x). Наприклад, якщо передбачається, що залежність, яка нас цікавить, описується рівнянням y = ax + b, тоді φ1(w) = w, a φ2(w) =1, і задачею дослідника буде знайти найбільш підходящі значення a1= a і a2= b.

8.1.2.6 **Метод найменших квадратів**

Коли прийнято рішення про те, які складові будуть входити до складу функції регресії, постає задача визначення значень коефіцієнтів цієї функції. Значення коефіцієнтів повинні бути знайдені такими, щоб лінія регресії пройшла як можна ближче до експериментальних точок. Існують різні методи пошуку коефіцієнтів функції регресії.

У методі найменших квадратів в якості критерію близькості лінії регресії до результатів експерименту приймають суму квадратів відхилень експериментальних точок від відповідних значень лінії регресії. Найкращими значеннями коефіцієнтів вважаються такі, при яких сума квадратів відхилень буде мінімальною.

$$\sum_{i=1}^{n}\left(y_i^{регр} - y_i^{эксп}\right)^2 \to \min \qquad (8.9)$$

Для того, щоб знайти оптимальне значення параметрів, потрібно знайти часткові похідні від функції 8.9 по всім невідомим коефіцієнтам і прирівняти їх до 0. Після цього залишається розв’язати отриману систему лінійних рівнянь відносно невідомих значень параметрів.

В простішому випадку, при одному невідомому коефіцієнті, цільова функція 8.9 прийме вигляд 8.10.

$$\sum_{i=1}^{n}\left(a_1 \cdot \varphi_1(x_i) - y_i^{эксп}\right)^2 \to \min \qquad (8.10)$$

Після диференціювання по невідомому коефіцієнту a1, отримаємо рівняння 8.11.

$$a_1 \sum_{i=1}^{p} \left(\varphi_1^2(x_i) \right) = \sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot y_i^{эксп} \right) \tag{8.11}$$

Із цього рівняння неважко знайти вираз 8.12 для визначення коефіцієнта, який ми хочемо знайти.

$$a_1 = \sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot y_i^{эксп} \right) \Big/ \sum_{i=1}^{p} \left(\varphi_1^2(x_i) \right) \tag{8.12}$$

У випадку двох невідомих коефіцієнтів цільова функція 8.9 прийме вигляд 8.13.

$$\sum_{i=1}^{n} \left(a_1 \cdot \varphi_1(x_i) + a_2 \cdot \varphi_2(x_i) - y_i^{эксп} \right)^2 \to \min \tag{8.13}$$

Після диференціювання по невідомим коефіцієнтам a1 і a2, отримаємо систему рівнянь 8.14.

$$\begin{aligned} a_1 \sum_{i=1}^{p} \left(\varphi_1^2(x_i) \right) + a_2 \sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot \varphi_2(x_i) \right) &= \sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot y_i^{эксп} \right) \\ a_1 \sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot \varphi_2(x_i) \right) + a_2 \sum_{i=1}^{p} \left(\varphi_2^2(x_i) \right) &= \sum_{i=1}^{p} \left(\varphi_2(x_i) \cdot y_i^{эксп} \right) \end{aligned} \tag{8.14}$$

Розв’язавши цю систему рівнянь, отримаємо

$$\begin{aligned} a_1 &= \left(\sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot y_i \right) \sum_{i=1}^{p} \left(\varphi_2^2(x_i) \right) - \sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot \varphi_2(x_i) \right) \sum_{i=1}^{p} \left(\varphi_2(x_i) \cdot y_i \right) \right) \Big/ D \\ a_2 &= \left(\sum_{i=1}^{p} \left(\varphi_2(x_i) \cdot y_i \right) \sum_{i=1}^{p} \left(\varphi_1^2(x_i) \right) - \sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot \varphi_2(x_i) \right) \sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot y_i \right) \right) \Big/ D \end{aligned} \tag{8.15}$$

де

$$D = \sum_{i=1}^{p} \left(\varphi_1^2(x_i) \right) \sum_{i=1}^{p} \left(\varphi_2^2(x_i) \right) - \sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot \varphi_2(x_i) \right)^2$$

Ці формули наведені тут тому, що вони реалізовані у класі RegresLinear, але не для того, щоб їх запам’ятовувати.

8.1.2.7 **Перевірка адекватності лінії регресії**

Параметри функції регресії визначаються виходячи із середніх значень функції відгуку для значень фактора, які аналізуються. Проте лінія регресії, як правило, проходить між точками, які відповідають цим середнім значенням. Відхилення лінії регресії від точок, які використовувались для її отримання, можуть бути любими. Великі відхилення свідчать про те, що функція регресії вибрана невдало і тому лінія регресії неадекватна експериментальним даним.

Для оцінки адекватності лінії регресії використовують дисперсію випадкових величин відхилень середніх значень функції відгуку від лінії регресії, яку можна називати дисперсією неадекватності (Dнад).

$$D_{над} = \frac{\sum_{i=1}^{p} (y_i^{ср} - y_i^{ргр})^2}{p-k} \qquad (8.16)$$

де $y_{iср}$ – середнє значення функції відгуку на і-му рівні;

$y_{iргр}$ – значення функції на лінії регресії для і-го рівня;

p – число рівнів зміни фактора;

k – число коефіцієнтів функції регресії, які були знайдені за значеннями $y_{ср}$.

Кількість степенів свободи для дисперсії, отриманої за цією формулою дорівнює p-k.

Лінію регресії вважають неадекватною в тому випадку, коли дисперсія неадекватності набагато більше дисперсії середніх значень функції відгуку, яка може біти знайдена із дисперсії експерименту $D_{эксп}$, за формулою 8.7.

Число степенів свободи дисперсії, отриманої за цією формулою, таке ж, як і у дисперсії експерименту, і дорівнює p*(n-1).

Для порівняння дисперсій можна використовувати критерій Фішера.

Лінія регресії може вважатися неадекватною, якщо відношення дисперсії неадекватності до дисперсії середніх значень функції відгуку більше критичного значення критерію Фішера. Очевидно, що таку перевірку слід проводити лише в тому випадку, коли дисперсія неадекватності перевищує дисперсію середніх значень. Якщо ж дисперсія неадекватності менше дисперсії середніх значень, то лінію регресії можна вважати адекватно.

8.2 Засоби для автоматизації проведення одно факторних експериментів та обробки їх результатів

8.2.1 *Компонент Experimentcontrol*

В пакеті widgets.experiments міститься клас ExperimentControl, який надає розробнику імітаційних моделей візуальний компонент для автоматизації проведення однофакторних експериментів і накопичення результатів проведених експериментів.

Компонент передбачає технологію, відповідно до якої модель створюється, ініціалізується та запускається самим компонентом.

Створюється модель за допомогою фабрики об’єктів, що реалізує інтерфейс IModelFactory. Єдиний метод цього інтерфейсу createModel(Dispatcher) забезпечує створення моделі і передачу їй посилання на диспетчера. Зв'язок компонента ExperimentControl з фабрикою моделей налаштовується через метод setFactory(IModelFactory).

Отримавши модель, компонент ExperimentControl приводить її до типу widgets.experiments.IExperimentable. Це інтерфейс, через який компонент

ExperimentControl буде працювати моделлю. Перелік методів інтерфейсу наведено у лістингу 8.1.

Лістинг 8.1 – Інтерфейс IExperimentable

```
public interface IExperimentable {
    public void initForExperiment(double factor);
    public double getResultOfExperiment();
}
```

Метод initForExperiment(double) має забезпечити підготовку моделі до однократного запуску. В якості параметра у метод передається значення фактору, вплив якого вивчається.

Метод getResultOfExperiment() використовується для отримання результату експерименту після його закінчення.

Для запуску моделі компонент буде використовувати метод диспетчера start().

Візуальна композиція компонента представлена на рисунку 8.1.

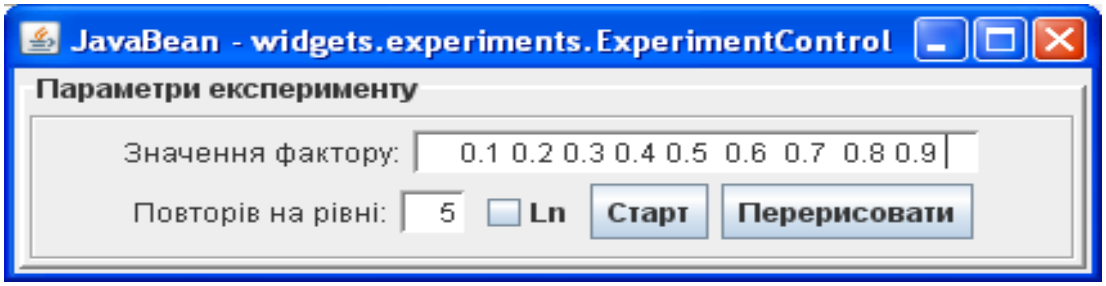

Рисунок 8.1 – Компонент для проведення експериментів

Компонент можна використовувати як елемент інтерфейсу користувача у застосуваннях для імітаційного моделювання.

Значення фактору, для яких потрібно провести експерименти, задаються у вигляді рядка символів в полі «Значення фактору:». Числові значення факторів повинні бути розділені стандартними розділовими знаками.

Кількість експериментів, що мають бути проведені для кожного значення фактору, задається у полі «Повторів на рівні:».

Необхідною умовою для роботи компонента є передача йому посилання на фабрику моделей що реалізує інтерфейс IModelFactory.

Кнопка «Старт» викликає метод buttonStartClick(), який забезпечує проведення заданої кількості експериментів на кожному рівні. Якщо до компоненту підключити об’єкт класу Diagram, то під час проведення експериментів на діаграмі будуть відображатися точки, які відповідають отриманим результатам, що забезпечує динамічну індикацію ходу експерименту.

На час проведення експерименту кнопка «Старт» блокується.

Після закінчення експериментів результати можна отримати за допомогою методу getResultMatrix().

Метод getFactorsArray() повертає масив значень фактору.

Результати експериментів можуть бути прологарифмовані. Для цього використовується перемикач «Ln».

Розглянемо, як компонент організує проведення експериментів з моделлю.

Перш за все компонент, за допомогою фабрики, створює модель.

Далі з інтерфейсу користувача зчитуються вихідні дані експерименту і готуються масиви для результатів експерименту.

Після завершення підготовчих операцій починаються експерименти з моделлю. Для цього в методі організовано подвійний цикл, що забезпечує періодичні запуски моделі за допомогою методу диспетчера start(). Перед кожним стартом диспетчера модель готується до запуск. Після кожного запуску моделі виконання методу призупиняється за допомогою методу join(), що викликається для потоку диспетчера.

В процесі виконання методу в змінній resultMatrix накопичуються результати експериментів. Коли експерименти будуть проведені, в цій змінній буде міститися матриця результатів. В змінній resultLnMatrix будуть міститися прологарифмовані значення функції відгуку.

Відновлення роботи методу відбувається після того як модель у черговий раз завершіть роботу. З текстом відповідного методу buttonStartClick() можна ознайомитися у вихідних кодах фреймворку.

8.2.2 *Компонент класу RegresAnaliser*

В пакеті widjets.regres міститься клас RegresAnaliser, який надає розробникам імітаційних моделей візуальний компонент для регресійного та дисперсійного аналізу результатів експериментів. Компонент можна використовувати при розробці інтерфейсу користувача у застосуваннях для імітаційного моделювання.

Передбачається, що даний компонент буде працювати з компонентами, які реалізують інтерфейс IRegresable. Зокрема, таким компонентом є компонент класу ExperimentControl. Зв'язок компонента з об’єктами, які реалізують інтерфейс IRegresable налаштовується через метод setIRegresable (IRegresable).

Опис інтерфейсу IRegresable представлений у лістингу 8.2. Методи цього інтерфейсу забезпечують доступ до експериментальних даних.

Лістинг 8.2 – Інтерфейс RegresExperimentable

```
public interface IRegresable {
   public double[] getFactorsArray();
   public double[][] getResultMatrix();
}
```

Для отримання результатів аналізу у графічному вигляді компоненту слід передати посилання на діаграму.

Компонент може працювати або у режимі одноріневого експерименту, або у режимі багаторівневого експерименту.

У першому випадку компонент видає інформацію про довірчий інтервал для результатів експерименту і графічно відображає його розміри на фоні експериментальних даних.

У режимі багаторівневого експерименту, компонент дозволяє вибрати функцію регресії і знайти її параметри і отримати графічне відображення. Окрім того надається інформація про результати перевірки на однорідність дисперсій, вплив фактору та адекватність функції регресії. Приклади використання компоненту зображені на рисунках 8.2, 8.3.

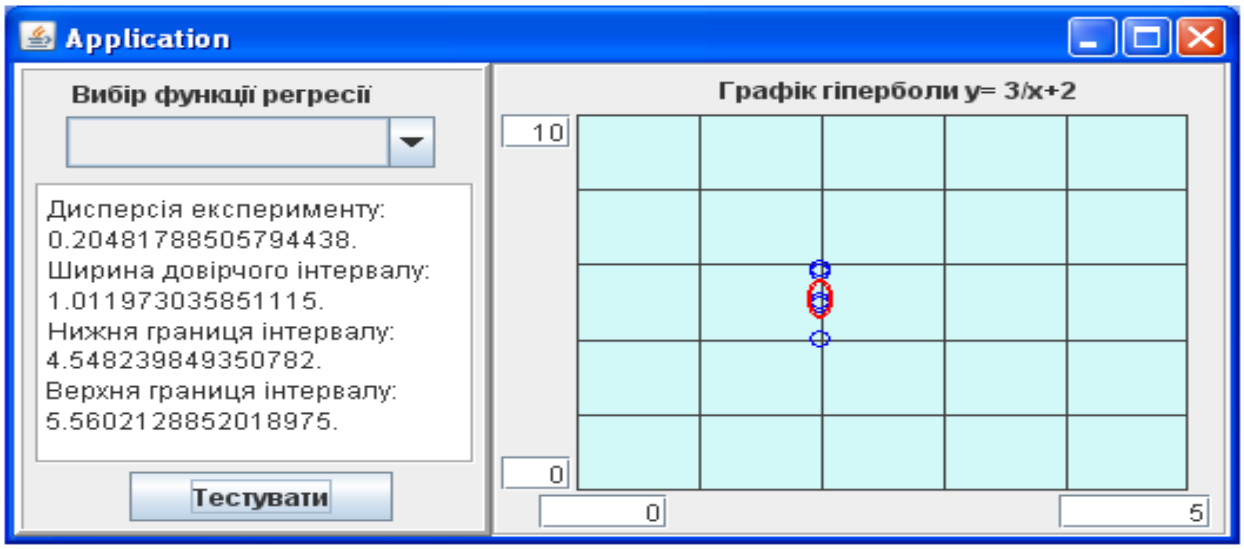

Рисунок 8.2 – Приклад використання RegresAnaliser для аналізу результатів одноріневого експерименту

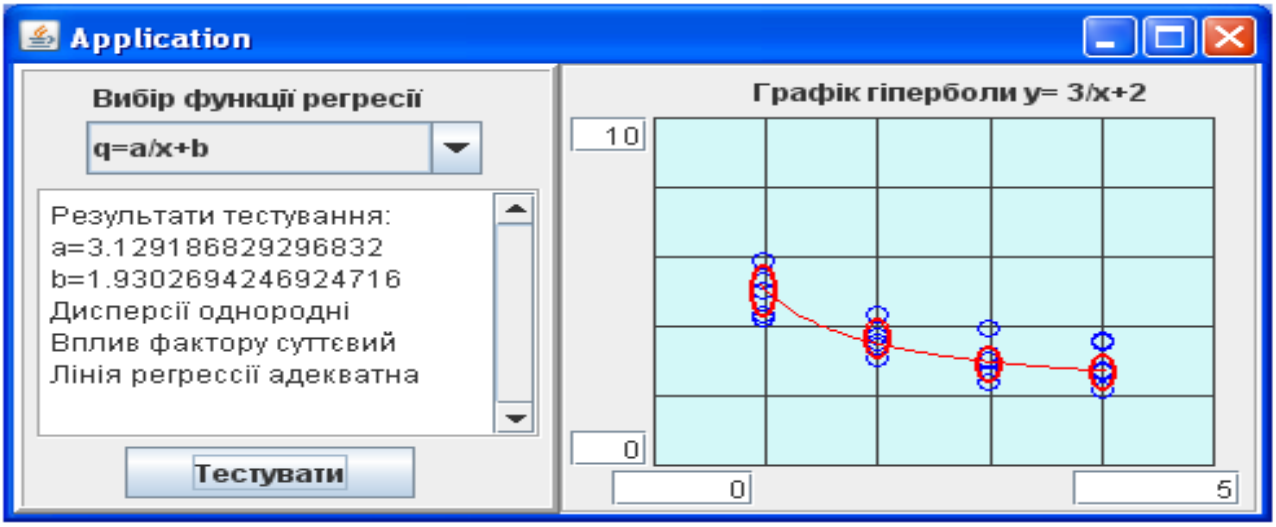

Рисунок 8.3 – Приклад використання RegresAnaliser для аналізу результатів багаторівневого експерименту

Компонент RegresAnaliser являє собою тільки візуальну оболонку. За обробку та аналіз даних відповідає ієрархія класів з базовим абстрактним класом RegresTesters, які також містяться в пакеті widjets.regres. Саме вони забезпечують дисперсійний та регресійний аналіз отриманих результатів.

8.2.2.1 **Клас RegresTesters**

Ієрархія класів, що забезпечують дисперсійний та регресійний аналіз результатів моделювання та публічні методи класу RegresTesters показані на рисунку 8.4.

Обробка даних ініціюється викликом методу testMatrix класу RegresTesters. Через параметри цього методу на обробку надходять матриця результатів експериментів matrix та масив значень факторів factorArray.

Виклик методу призводить до розрахунку статистичних характеристик отриманих даних. Отримані характеристики запам’ятовуються у полях класу RegresTesters, які перераховані нижче.

p – кількість рівнів зміни фактора.

n – кількість експериментів на рівні.

avrgArray – масив середніх значень відгуку для кожного рівня.

dArray – масив дисперсій відгуку на кожному рівні.

doverArray – масив довірчих інтервалів на рівнях.

avrg – загальне середнє.

dExpr – дисперсія експерименту.

dFactor – дисперсія фактору.

dAdequat – дисперсія адекватності.

Доступ до цих даних можливий у підкласах, а також реалізується через методи get…() цього класу.

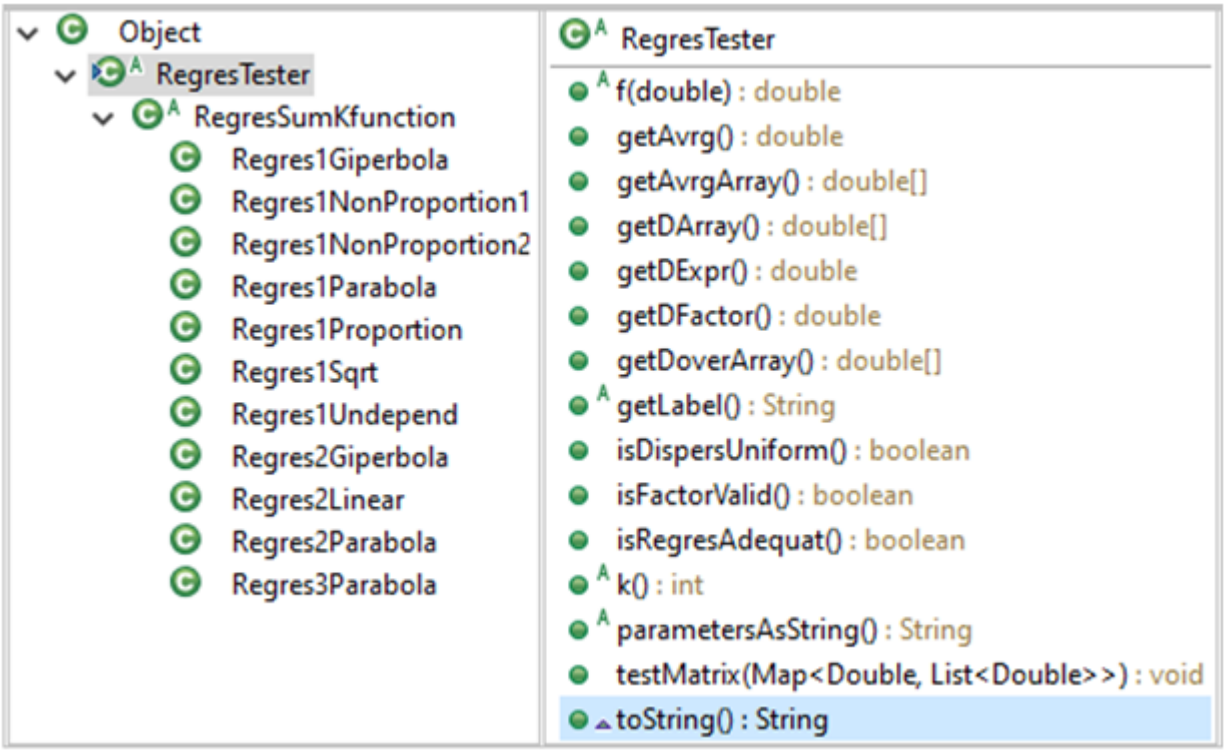

Рисунок 8.4 – Спадкоємці та методи класу RegresTesters

Метод testMatrix для проведення дисперсійного та регресійного аналізу використовує абстрактні методи f() та k(), які мають бути реалізовані у підкласах.

Метод f(double) використовується для розрахунку значень функції регресії.

Клас RegresTesters має абстрактного спадкоємця RegresSumKfunction.

8.2.2.2 **Клас RegresSumKfunction**

Клас RegresSumKfunction забезпечує отримання рівняння регресії у вигляді f=a1*φ1(x)+a2*φ2(x)+…+aN*φN(x), де φ1(x), φ2(x), …, φN(x) – довільні визначені функції від значень фактора, а невідомі коефіцієнти a1, a2, …, aN визначають вигляд лінії регресії.

Протокол класу RegresSumKfunction наведено на рисунку 8.5.

Абстрактний метод functionList() повертає перелік функцій φ1(x), φ2(x), …, φN(x).

Метод calculateParameters() використовується для розрахунку коефіцієнтів функції регресії a1, a2, …, aN.

Наявність переліку функцій та коефіцієнтів функції регресії дозволяє реалізувати методи f(), k() та parametersAsString().

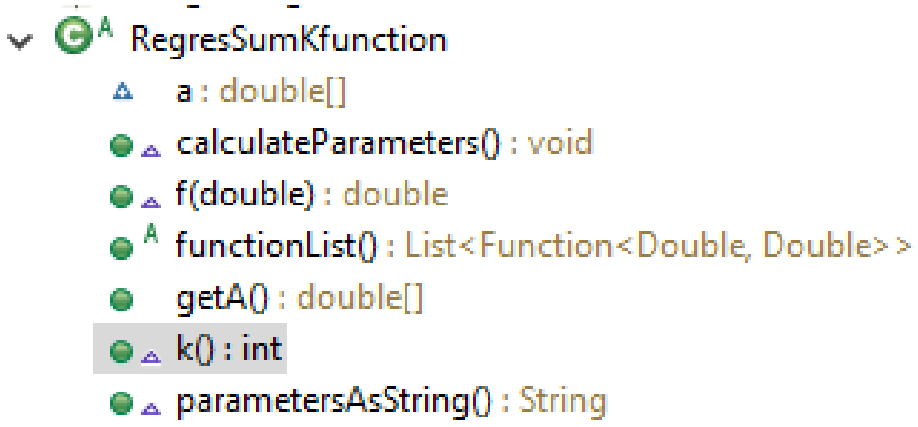

Рисунок 8.5 – Протокол класу RegresSumKfunction

8.2.2.3 **Спадкоємці класу RegresSumKfunction**

Клас RegresSumKfunction має декілька спадкоємців, у яких реалізовано певна кількість і конкретний вигляд функцій φ1(x), φ2(x), …, φN(x).

Клас Regres1Undepend реалізує функцію φ1(x) у вигляді φ1(x) = 1. В результаті виходить, що клас RegresUndepend реалізує функцію регресії виду f=a. Це випадок, коли відгук не залежить від фактору.

Клас Regres1Proportion використовується для створення об’єктів, які забезпечують отримання рівняння регресії у вигляді f=a*x. Для цього клас реалізує функцію φ1(x) у вигляді φ1(x) = x.

Клас Regres1NonProportion1 використовується для створення об’єктів, які забезпечують отримання рівняння регресії у вигляді f=a*x/(1-x). Для цього клас функцію φ1(x) у вигляді φ1(x) = x/(1-x).

Клас Regres1NonProportion2 реалізує функцію φ1(x) у вигляді φ1(x) = x*x/(1-x). В результаті забезпечується отримання рівняння регресії у вигляді f=a*x*x/(1-x).

Клас Regres1Giperbola реалізує функцію φ1(x) у вигляді φ1(x) = 1/x. В результаті забезпечується отримання рівняння регресії у вигляді f=a/x.

Клас Regres1Parabola реалізує функцію φ1(x) у вигляді φ1(x) = x*x. В результаті забезпечується отримання рівняння регресії у вигляді f=a*x*x.

Клас Regres1Sqrt реалізує функцію φ1(x) у вигляді φ1(x) = Math.sqrt(x). В результаті забезпечується отримання рівняння регресії у вигляді f=a*sqrt(x).

Клас Regres2Linear реалізує визначення параметрів функції регресії виду f=a1*x+a2. В цьому реалізовано функцію φ1(x) у вигляді φ1(x) = x, а функцію φ2(x) у вигляді φ2(x) = 1.

Клас Regres2Giperbola реалізує визначення параметрів функції регресії виду f=a1/x+a2. Для цього клас реалізує функцію φ1(x) у вигляді φ1(x) = 1/x, а функцію φ2(x) у вигляді φ2(x) = 1.

Клас Regres2Parabola реалізує функції φ1(x) = x*x, φ2(x) = x. В результаті забезпечується отримання рівняння регресії у вигляді f=a1*x*x + a2*x.

Клас Regres3Parabola реалізує функції φ1(x) = x*x, φ2(x) = x, φ3(x) = 1. В результаті забезпечується отримання рівняння регресії у вигляді f=a1*x*x+ a2*x+a3.

Клас Regres4Parabola реалізує функції φ1(x) = x*x*x, φ2(x) = x*x, φ3(x) = x, φ4(x) = 1. В результаті забезпечується отримання рівняння регресії у вигляді f=a1*x*x*x+ a2*x*x+a3*x+a4.

8.2.2.4 **Розширення переліку функцій регресії компоненту RegresAnaliser**

Щоб додати до компоненту RegresAnaliser ще якусь функцію регресії, необхідно створити клас, що успадковує клас RegresTesters, і реалізувати у ньому абстрактні методи суперкласу. Завдання спрощується, якщо нова функція регресії має вигляд, що передбачений класом RegresSumKfunction. У цьому випадку достатньо успадкувати цей класів та реалізувати у ньому методи functionList() та getLabel().

Після створення класу, що відповідає за нову функцію регресії, слід створити об’єкт цього класу і передати його компоненту RegresAnaliser за допомогою методу addFunction(RegresTesters).

Можна також скористатися анонімним класом. Нижче наведено приклад реалізації функції регресії y=a1*x^2+a2*x+a3.

```
em.getRegresAnaliser().addFunction(new RegresSumKfunction() {
  @Override
  public List<Function<Double, Double>> functionList() {
    List<Function<Double, Double>> list = new LinkedList<>();
    list.add((x)-> x*x);
    list.add((x)-> x);
    list.add((x)-> 1.0);
```

```
            return list;
          }
          public String getLabel() {
            return "q=a1*x*x+a2*x +a3";
          }
        });
```

КОНТРОЛЬНІ ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Як можна представити результати однофакторного експерименту на одному рівні?
2. Як визначити границі довірчого інтервалу.
3. Як правильно вибрати кількість рівнів для багатофакторного експерименту?
4. Як перевірити дисперсії на однорідність?
5. Як перевірити гіпотезу про вплив фактору на відгук?
6. У чому суть методу найменших квадратів?
7. Як перевірити адекватність функції регресії?
8. Призначення інтерфейсу Iexperimentable.
9. Як працює компонент ExperimentControl?
10. Призначення інтерфейсу Iregresable.
11. Охарактеризуйте ієрархію класів RegresTesters.
12. Призначення компоненту RegresAnaliser.
13. Як розширити перелік функцій регресії компоненту RegresAnaliser?
14. Які методи потрібно реалізувати у класах, що успадковують класи Regres1 та Regres2

9 ТЕМА 9. ДОСЛІДЖЕННЯ ПЕРЕХІДНИХ ПРОЦЕСІВ У СМО

План лекції

– Методика дослідження перехідних процесів у СМО шляхом імітаційного моделювання.

– Ще раз про функцію регресії та метод нйменших квадратів.

– Методами пошуку екстремумів функцій однієї змінної.

– Метод координатного спуску.

– Компонент для проведення експерименту по аналізу переходного процесу.

– Компонент для автоматизованого пошуку параметрів переходного процесу.

9.1 Перехідні процеси у СМО

СМО відносяться до динамічних систем і, отже, при вивченні цих систем слід розрізняти сталий режим роботи (для реальних СМО він зазвичай рано чи пізно настає) і перехідний процес. Так, наприклад, черга в простішій СМО спочатку дорівнює 0, а через деякий час починає змінюватись навколо деякого середнього значення. Перші заявки у системі чекають на обслуговування недовго, навіть при великому завантаженні системи, а для заявок, які з'явилися пізніше, середній час очікування може бути досить великим.

Досліджувати перехідні процеси аналітично, як правило, не вдається через складність диференціальних рівнянь, що описують поведінку системи в перехідному режимі. Проте, ці режими є інтересними для практики. Перш за все, вони цікаві для систем, які працюють недовго, і не встигають досягти сталого режиму. Але навіть при вивченні сталих режимів, інформація про тривалість перехідного процесу корисна, оскільки дозволяє визначити момент часу, з якого слід почати збирати інформацію про сталий режим. Таким чином, проблема побудови моделей, що дозволяють оцінити характер перехідних процесів шляхом моделювання, є досить актуальною.

9.1.1 *Методика отримання експериментальних даних про перехідний процес*

Під час спостереження за роботою однієї СМО важко встановити закономірності, характерні для перехідних процесів. Це пов'язано із сильними флуктуаціями змінної, що нас цікавить, наприклад, довжини черги. Але якщо одночасно запустити багато паралельно працюючих моделей, і спостерігати за змінами усередненого по всіх моделях значення параметру, то вплив флуктуацій буде ослаблено, і будуть виявлені закономірності характерні для перехідного режиму.

Маючи засоби для моделювання псевдопаралельних процесів, вирішити задачу виявлення перехідного процесу достатньо легко. Для цього можна одночасно моделювати роботу декількох однакових систем і проводити

усереднення змінної, яка нас цікавить. Чим більше таких систем буде паралельно працювати, тим менш помітними будуть флуктуації, і тим виразніше будуть виявлятися закономірності перехідного процесу. Надійність результатів, одержаних при цьому, залежить тільки від потужності комп'ютера і часу, виділеного на проведення експериментів.

Таким чином можна запропонувати наступну методику оцінки параметрів перехідного процесу для середньої довжини черги. Створюється велика кількість однакових моделей, які одночасно стартують і паралельно працюють у віртуальному модельному часі, єдиному для всіх моделей. Кожна з моделей на невеликому відрізку часу (інтервал накопичення або дискретизації) збирає поточну інформацію про довжину черги і наприкінці цього відрізку часу визначає середнє значення черги на цьому інтервалі, реалізуючі таким чином усереднення у часі (рисунок 9.1).

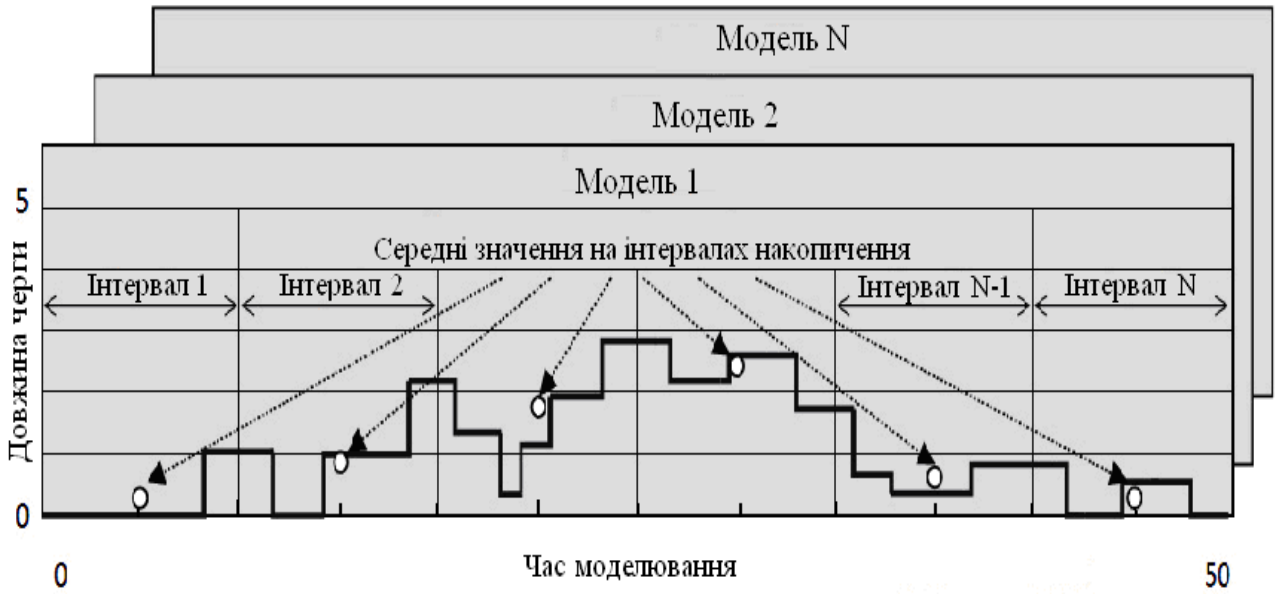

Рисунок 9.1– Усереднювання значень довжини черги за часом у межах однієї реалізації

Послідовність середніх значень, отриманих у ході експерименту, не дає уявлення про характер перехідного процесу, але дозволяє провести дискретизацію процесу. Усереднення ж результатів, отриманих на кожному інтервалу накопичення, по всіх реалізаціях дозволяє отримати послідовність середніх значень довжини черги для кожного інтервалу, що дасть уявлення про характер перехідного процесу (рисунок 9.2).

Інтервал накопичення може бути як-завгодно малим, але в його суттєвому зменшенні нема потреби, якщо у перехідному процесі нема високочастотних складових.

Для отриманої послідовності можна знайти функцію регресії і таким чином отримати аналітичний опис перехідного процесу.

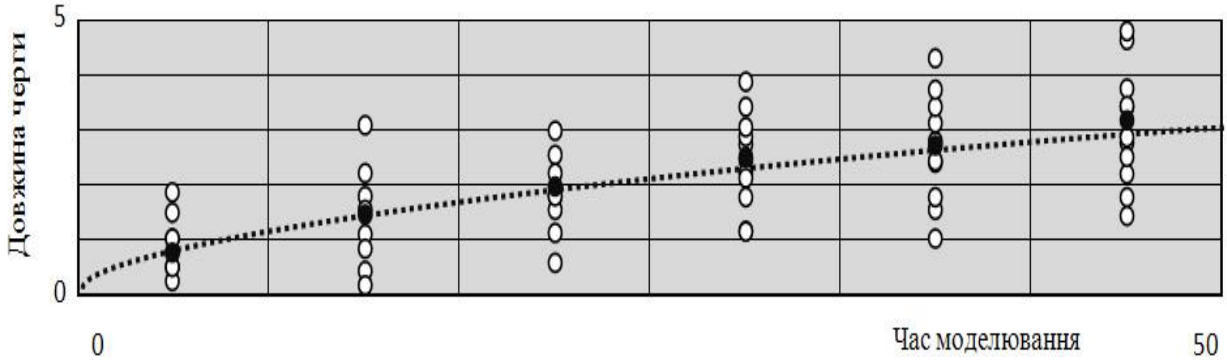

Рисунок 9.2 – Результати усереднювання середньої довжини черги по реалізаціях

В прикладі, який розглядався, зміна характеристики СМО (довжина черги) була пов’язана із часом. Проте таке можливо не завжди.

Наприклад, якщо досліджувати перехідний процес для часу очікування в черзі, то його потрібно пов'язувати не з часом, а з номером заявки. Тоді, в результаті усереднення по реалізаціях, буде отримана залежність середнього часу очікування в черзі від номера заявки.

9.2 Обробка експериментальних даних про перехідний процес

9.2.1 *Функція регресії*

Для даних, представлених на рисунку 9.2 і характерних для найпростішої СМО, можна припустити, що залежність, яка нас цікавить, має експоненціальний характер. Така лінія регресії зображена на рисунку 9.2. Відповідна аналітична залежність має вигляд формули 9.1.

$$q(t) = \left(1 - e^{-\frac{t}{T}}\right) \cdot q_{уст} \qquad (9.1)$$

де $q_{уст}$ – величина, до якої прямує значення середньої довжини черги з плином часу. В залежності від вибору цього параметру рівень, який установився , буде вище, або нижче;

Т – параметр, який називають «сталою часу». Чим більше його значення, тим повільніше росте значення функції.

Теоретично, сталий режим для такого процесу наступає лише в нескінченності, але на практиці звичайно вважають, що перехідний процес закінчується за час рівний 3Т. За цей час значення функції досягає величини рівної 95% від сталого значення

9.2.2 *Цільова функція*

Вибір функції регресії – це лише половина справи. Потрібно ще визначити значення її параметрів. В прикладі, що розглядався вище, таких параметрів два – постійна часу і стале значення. Від вибору значень цих параметрів залежить, як близько до експериментальних точок пройде лінія

регресії.

Для прийняття рішення про вибір значень параметрів, потрібно мати критерій оцінки близькості лінії регресії до експериментальних точок. Формальне описання такого критерію називають цільовою функцією. Значення цільової функції залежить від вибору параметрів і повинне збільшуватися, якщо лінія регресії віддаляється від експериментальних точок, і зменшуватися при її наближенні. Найкращими значеннями параметрів будуть такі, при яких цільова функція має мінімальне значення.

У якості цільової функції можна прийняти суму абсолютних значень ординат відхилень експериментальних точок від відповідних значень лінії регресії.

$$\sum_{i=1}^{n}\left|q_i^{регр}-q_i^{експ}\right|\to \min \qquad (9.2)$$

Недолік цього полягає в тому, що при аналітичному визначенні значень параметрів, що забезпечують мінімум цільової функції, доведеться знаходити часткові похідні по цих параметрах, а модуль робить функцію такою, що не можна диференціювати. З цієї причини найчастіше цільова функція представляється у вигляді суми квадратів відхилень

$$\sum_{i=1}^{n}\left(q_i^{регр}-q_i^{експ}\right)^2\to \min \qquad (9.3)$$

9.2.3 *Метод найменших квадратів*

Метод найменших квадратів припускає представлення цільової функції у вигляді суми квадратів відхилень. Для того, щоб знайти оптимальні значення параметрів, потрібно взяти похідні від цільової функції по цих параметрах і прирівняти їх до 0. Після цього залишається розв’язати отриману систему рівнянь щодо невідомих значень параметрів. В розглянутому вище прикладі цільова функція виглядатиме так

$$\sum_{i=1}^{n}\left(q_{уст}\left(1-e^{-\frac{t_i}{T}}\right)-q_i\right)^2\to \min \qquad (9.4)$$

Взявши похідні від цільової функції по $q_{уст}$ і T, та прирівнявши їх до нуля 0, отримаємо систему рівнянь 9.5 для визначення цих параметрів.

На жаль, вирішити аналітично цю систему рівнянь не вдається. Але знайти оптимальні значення $q_{уст}$ і T можна за допомогою чисельних методів пошуку екстремуму. В цьому випадку цільова функція не обов'язково повинна бути такою, що диференціюється і можна мінімізувати як суму квадратів відхилень, так і суму модулів відхилень.

$$
\begin{aligned}
& q_{уст}\sum_{i=1}^{n}\left(1-e^{-\frac{t_i}{T}}\right)^2-\sum_{i=1}^{n}q_i\left(1-e^{-\frac{t_i}{T}}\right)=0 \\
& q_{уст}\sum_{i=1}^{n}t_i e^{-\frac{t_i}{T}}\left(1-e^{-\frac{t_i}{T}}\right)-\sum_{i=1}^{n}t_i q_i e^{-\frac{t_i}{T}}=0
\end{aligned}
\qquad (9.5)
$$

9.3 Методи пошуку екстремуму для функції однієї змінної

9.3.1 *Пошук екстремуму за допомогою чисел Фібоначчі*

Планування експериментів для пошуку оптимального значення цільової функції, яка залежить від однієї змінної, полягає в наступному.

Перш за все, задається точність, з якою потрібно визначити невідомий параметр. Цю величину називатимемо кроком пошуку.

Потім визначаються границі області пошуку, всередині якої знаходиться екстремум, причому екстремум повинен бути тільки один. Такі функції називають унімодальними.

9.3.1.1 **Визначення границь області пошуку**

Для визначення границь області пошуку слід максимально використовувати всю наявну апріорну інформацію про досліджувану систему. В тому випадку, якщо границі області пошуку відразу визначити не вдається, зазвичай використовують наступний алгоритм (передбачається, що ми шукаємо мінімум цільової функції).

Вибирається будь-яке значення параметра і для нього обчислюється значення цільової функції.

Значення параметра збільшується на величину кроку, обчислюється значення цільової функції в новій точці, і порівнюється з її значенням в початковій точці.

Якщо нове значення менше, значить напрямок пошуку вибрано правильно і перший етап завершено.

Якщо нове значення більше попереднього, то знак кроку зміни змінної потрібно змінити на протилежний і повторити попередні дії.

Якщо ж і в цьому випадку значення цільової функції більше, ніж в початковій точці, то це означає, що мінімум знаходиться в початковій точці і його шукати вже не потрібно.

В результаті виконання першого етапу, стає відомим напрямок пошуку, який задається знаком кроку зміни параметра.

Далі починається пошук другої границі області пошуку.

В розглянутому нижче алгоритмі пошуку, відстань від початкової точки до точки, де проводиться експеримент, поступово збільшується з кожним новим кроком. Для формування цих відстаней зручно використовувати числа

Фібоначчі.

Особливість чисел Фібоначчі полягає в тому, що кожне наступне, починаючи з 3-го, дорівнює сумі двох попередніх:

1 1 2 3 5 8 13 21 34 55 89 . . .

Пошук починають з числа 3. В результаті, значення цільової функції будуть відомі для точок 0,1 і 3, оскільки точка 0 відповідає початковій, а в точці 1 проводився експеримент для визначення напрямку пошуку.

На наступному кроці точка 1 стає початковою і перетворюється в 0. Точка 3 приймає значення 2 (2=3-1), а новий експеримент проводять в точці 5.

Якщо функція продовжує зменшуватися, то далі 2 перетворюється на 0, 5 на 3 і новий експеримент проводять в точці 8.

Наступний експеримент буде проводитись у точці, віддаленій на 13 кроків від початкової, причому, слід враховувати, що абсолютна координата початкової точки постійно зміщується, тобто в процесі пошуку правої границі ми постійно змінюємо і ліву границю.

Так продовжується до тих пір, поки значення цільової функції не почне зростати, рисунок 9.3.

Зверніть увагу, послідовність розмірів інтервалів між точками, де проводилися експерименти, відповідає послідовності чисел Фібоначчі

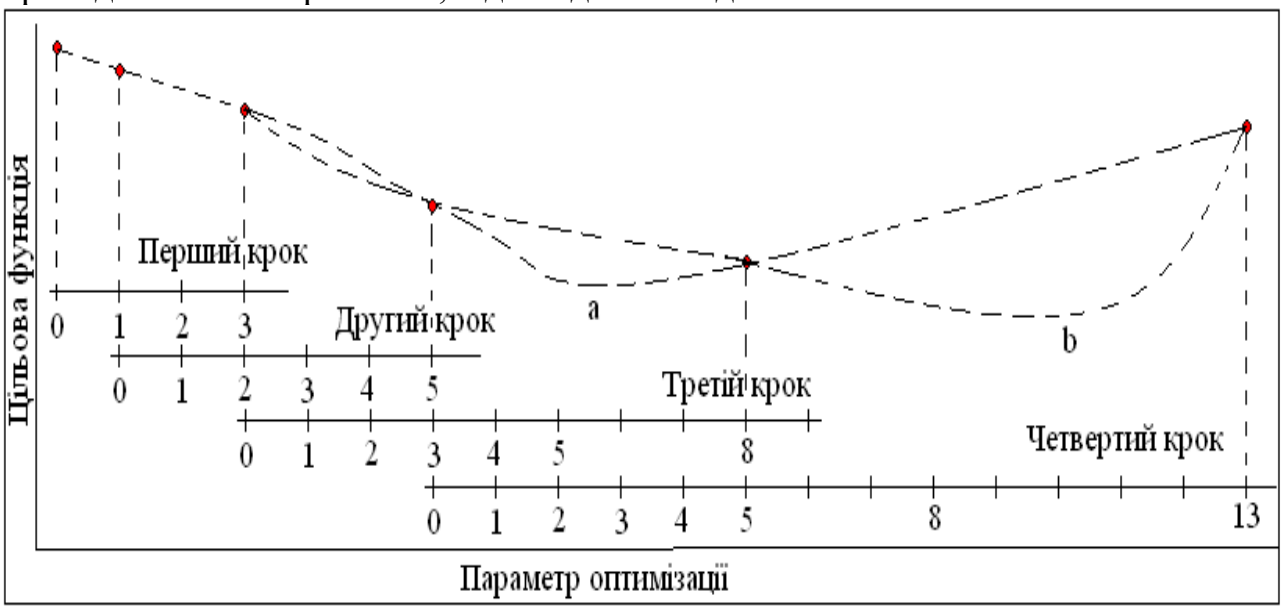

Рисунок 9.3 – Результати визначення меж області пошуку

На рисунку 9.3 зображена ситуація, коли в точці 13 цільова функція почала зростати. В результаті, на краях інтервалу значення функції стали більше, ніж посередині. Проте мінімум може бути розташований як у лівій частині інтервалу – варіант кривої “a” на рисунку 9.3, так і в правій – варіант кривої “b” на тому ж рисунку.

9.3.1.2 **Пошук екстремуму у середині області**

Після визначення границь області пошуку, можна починати пошук точки

екстремуму всередині цієї області. Пошук полягає в поступовому звуженні знайденої області пошуку. Тут також дуже зручно використовувати числа Фібоначчі. Алгоритм фактично є продовженням попереднього і полягає в наступному.

Усередині області пошуку проводять додаткові експерименти так, щоб значення цільової функції були відомі в обох точках, які відповідають числам Фібоначчі, з яких складається число, що визначає праву границю пошуку. Для рисунка 9.3 це точки 5 і 8. Як правило, для однієї з цих точок результат вже визначений, залишається протестувати іншу точку. В результаті, область пошуку розбивається на 3 інтервали, ширина кожного з яких пропорційна деякому числу Фібоначчі.

Після цього порівнюють значення функції відгуку для двох точок всередині області пошуку, і приймають рішення про звуження цієї області, приймаючи в якості нової границі області пошуку точку з більшим значенням функції відгуку. Якщо значення цільової функції для точок всередині області пошуку однакові, то відкидається і лівий і правий інтервали. Ширина нової області пошуку у будь-якому випадку знову виходить рівною числу Фібоначчі.

У випадку, якщо змінюється ліва границя області пошуку, слід відкоригувати значення чисел Фібоначчі для границь області і точок всередині неї. Всі ці числа повинні бути зменшені на старе значення лівої границі. В результаті, лівій границі знову відповідатиме число 0.

Якщо ширина хоча б одного з інтервалів більше одиниці, процес повторюється.

В іншому випадку обирається точка з мінімальним значенням цільової функції, яка і буде точкою оптимуму.

Розглянемо приклад пошуку у середині області, як продовження попереднього. Нехай функція відгуку має вигляд показаний на рисунку 9.4, варіант b.

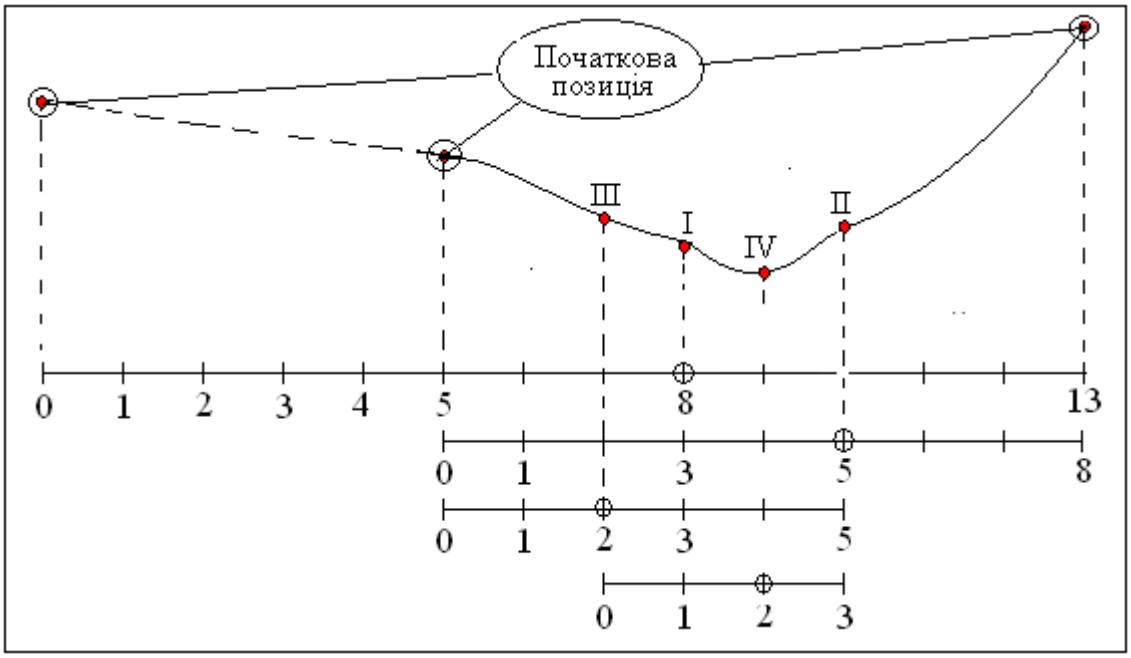

Рисунок 9.4 – Пошук в межах області

Після визначення меж області пошуку ми маємо результати експериментів у точках 0, 5 та 13. Ширина області дорівнює числу Фібоначчі 13, що складається з 5 та 8, тому наступні експерименти слід проводити в точках 5 та 8. Але в точці 5 експеримент вже проведено. Залишається провести експеримент в точці 8, результат якого має позначку I.

Цей результат менший ніж у точці 5, тому ліву границю пошуку переносимо у точку 5 и змінюємо систему координат.

Ширина нової області дорівнює числу 8, що складається з 3 та 5. У точці 3 (до зміни координат це була точка 8) експеримент вже проведено. Залишається провести експеримент в точці 5, результат якого має позначку II.

Цей результат більший, ніж у точці 3, тому змінюємо праву границю і переносимо її до точки 5. Систему координат при цьому змінювати не потрібно.

Ширина звуженої області дорівнює числу 5, що складається з 2 та 3. У точці 3 експеримент вже проведено. Залишається провести експеримент в точці 2, результат якого має позначку III.

Цей результат більший, ніж у точці 3, тому змінюємо ліву границю і переносимо її до точки 3 і знову змінюємо систему координат.

Ширина нової області дорівнює числу 3 і в усіх точках області, окрім точки 2, експерименти вже проведено. Залишається провести останній експеримент і визначитися з мінімумом. На рисунку результат експерименту має позначку IV і тут відгук має мінімальне значення, але могло статися й так, що відгук у цій точці мав значення більше ніж у точці 2, тоді б саме це значення параметру було б оптимальним.

Як бачимо, основна перевага методу Фібоначчі полягає в тому, що на кожному кроці використовуються дані попереднього кроку, що скорочує число експериментів.

9.3.2 *Метод золотого перетину*

Цей метод відрізняється від попереднього лише тим, що розмір області пошуку не пов’язаний з числом Фібоначчі, але співвідношення між інтервалами, на які ділиться область пошуку, повинне зберігатися постійним. На рисунку 9.5 представлено розділення області пошуку при застосуванні методу золотого перетину.

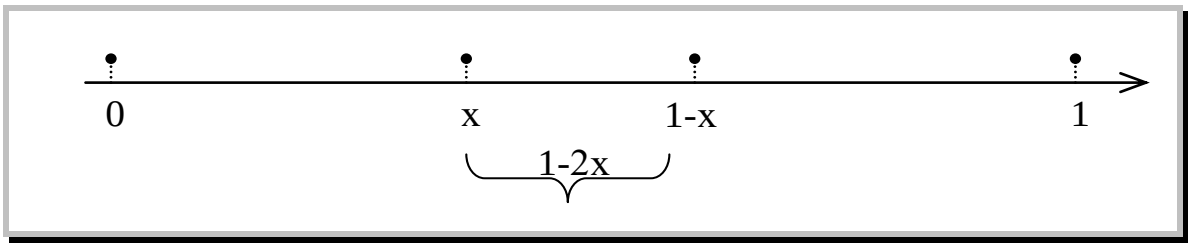

Рисунок 9.5 – Розділення області пошуку при застосуванні методу золотого перетину

Ширина всієї області на рисунку представлена одиницею. Точки всередині області, в яких потрібно провести експерименти, віддалені від

границь області на відстань x. Після оцінки результатів експериментів в цих точках, один із крайніх інтервалів області буде відкинуто і точки, в яких потрібно буде проводити експерименти на наступному кроці пошуку, будуть вже віддалені від границі інтервалу на відстань 1-2x. Але відношення довжини цього інтервалу до ширини нової області пошуку повинно залишатися незмінним. Тобто

$$\frac{1-2x}{1-x} = \frac{x}{1} \tag{9.6}$$

Із цього відношення отримуємо квадратне рівняння для пошуку x.

$$x^2 - 3x + 1 = 0 \tag{9.7}$$

Розв'язавши це рівняння, отримуємо

$$x \approx 0.382,\ 1\text{-}x \approx 0.618,\ 1\text{-}2x \approx 0{,}236 \tag{9.8}$$

9.3.3 *Метод дихотомії*

Цей метод один із найвідоміших, але не найкращий, оскільки він вимагає проведення більшої кількості експериментів в порівнянні із попередніми методами. Суть методу полягає в наступному, рисунок 9.6.

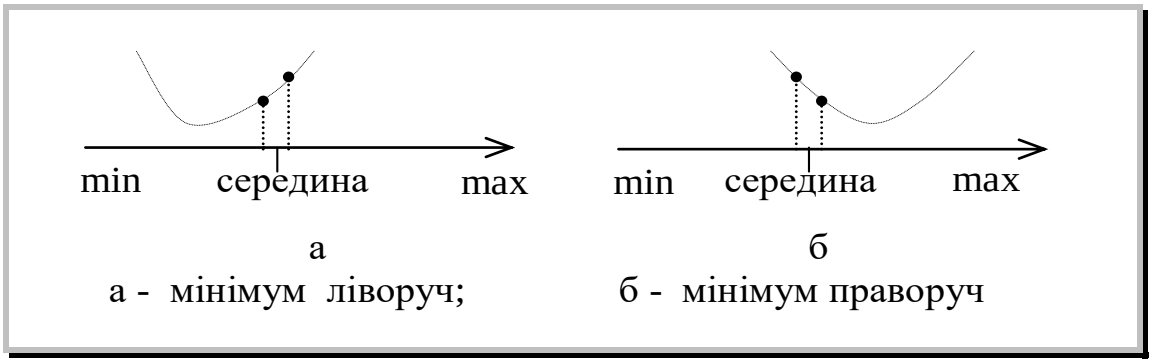

Рисунок 9.6 – Пошук екстремуму методом дихотомії

Область пошуку ділиться навпіл і з двох сторін від середньої точки, на невеликій відстані, яка відповідає необхідній точності визначення значення параметра, проводиться два експерименти. На підставі результатів цих дослідів робиться висновок про те, де знаходиться екстремум - в лівій або правій половині області пошуку, що дозволяє звузити область пошуку в 2 рази.

В середині нової області пошуку знову проводять два експерименти і так доти поки ширина області пошуку не стане менше заданої точності.

9.4 Пошук екстремуму функції кількох змінних

Існує декілька способів вирішення цієї задачі. Найбільш простим із них вважається метод координатного спуску. Суть його полягає в наступному.

Обирається початкова точка для пошуку і, починаючи від цієї точки, знаходиться мінімум цільової функції, шляхом зміни лише однієї змінної. Вся решта змінних не міняється.

Після визначення мінімуму по вибраній координаті починається пошук мінімуму по наступній координаті і так далі. Після того, як значення цільової функції перестануть змінюватися при переході від однієї змінної до іншої, пошук закінчується.

9.5 Засоби для дослідження перехідних процесів

Класи та інтерфейси, які можуть бути використані під час дослідження перехідних процесів у різних моделях, в тому числі й під час курсового проектування, знаходяться у пакеті widgets.trans бібліотеки Simulation.

9.5.1 *Інтерфейс ITransProcesable*

Цей інтерфейс, лістингу 9.1, визначає протокол спілкування з моделлю, у якої досліджується перехідний процес.

Лістинг 9.1 - Методи інтерфейсу ITransProcesable

```
public interface ITransProcesable {
   public void initForTrans(double finishTime);
   public void resetTransAccum();
   public double getTransResult();
}
```

Метод initForTrans(double finishTime) використовується монітором для ініціалізації компонентів моделі. Як параметр до методу передається тривалість моделювання.

Метод resetAccum() повинен забезпечувати ініціалізацію накопичувача інформації в моделі

Метод getTransResult() забезпечує повернення результату накопичення інформації.

9.5.2 *Клас TransProcessQueues*

Цей клас успадковує усі можливості класу QueueForTransaction і моделює черги, які мають здатність накопичувати інтеграл від свого розміру за часом і повертати середнє значення черги на будь-якому проміжку часу. Таким чином використання такого об’єкту в моделі у якості черги дозволяє проводити усереднення розміру черги у часи і використовувати ці дані для побудови перехідного процесу, а також реалізувати у моделі інтерфейс ITransProcesable.

На рисунку 9.7 наведена діаграма даного класу.

Змінна sum використовується для накопичення інтеграла від довжини черги на заданому проміжку часу. Змінна accumTime використовується для накопичення довжини інтервалу інтегрування. Змінна lastTime використовується для зберігання моменту часу, коли відбулася остання зміна черги.

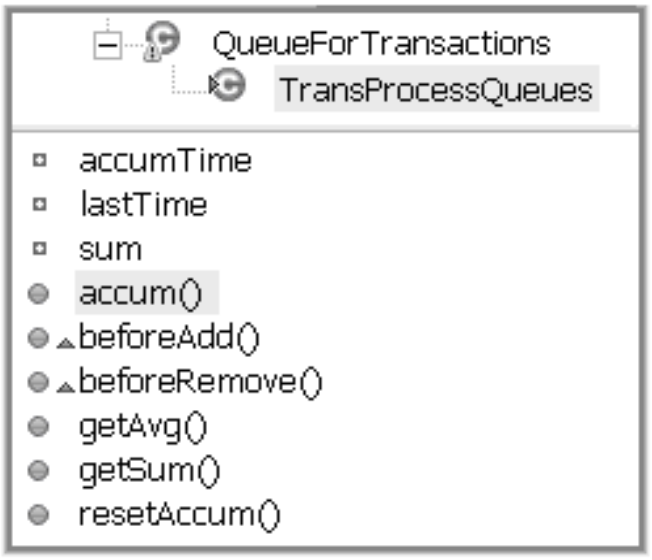

Рисунок 9.7 - Діаграма класу TransProcessQueues

Змінні об'єктів ініціалізуються в методі resetAccum, текст якого наведений у лістингу 9.2.

Лістинг 9.2 - Текст методу resetAccum ()класу TransProcessQueues

```
public void resetAccum() {
   lastTime = getDispatcher().getCurrentTime();
   sum = 0;
   accumTime = 0;
}
```

Для накопичення інтегралу використовується метод accum(), що викликається з методів beforeAdd() і beforeRemove() перед зміною черги. У методі accum різниця між поточним значенням часу й значенням змінної lastTime множиться на розмір черги й додається до значення змінної sum. Після цього значення змінної lastTime обновляється. Текст методу наведений у лістингу 9.3.

Лістинг 9.3 - Текст методу accum () класу TransProcessQueues

```
public void accum() {
   double dt = getDispatcher().getCurrentTime() - lastTime;
   sum += dt * this.size();
   accumTime += dt;
   lastTime = getDispatcher().getCurrentTime();
}
```

Метод getAvg() повертає середнє значення черги на інтервалі накопичення. Текст методу наведений у лістингу 9.4.

Лістинг 9.4 - Текст методу getAvg() класу TransProcessQueues

```
public double getAvg() {
   accum();
   return sum / accumTime;
}
```

При звертанні до методу відбувається обробка останнього інтервалу накопичення й визначається середнє значення черги шляхом ділення накопиченого інтеграла на довжину інтервалу накопичення.

9.5.3 *Компонент TransMonitorView*

Цей компонент використовується для проведення експериментів по виявленню закономірностей перехідного процесу в досліджуваній моделі, що реалізує інтерфейс ITransProcesable.

Візуальна частина компоненту, рисунок 9.8, забезпечує його налаштування, а також запуск процесу моделювання.

По натисканню кнопки “Старт” починається процес моделювання.

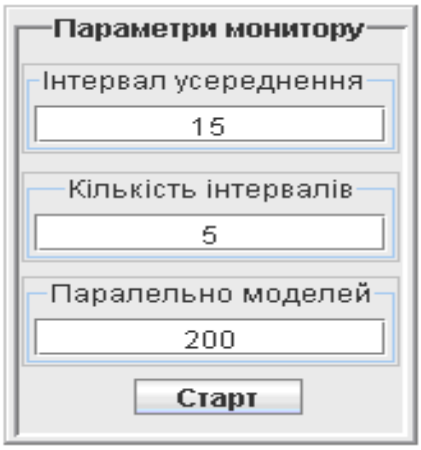

Рисунок 9.8 - Візуальна композиція класу TransMonitorView

Поле “Інтервал усереднення” визначає довжину інтервалу накопичення інформації.

Поле “Кількість інтервалів” визначає кількість інтервалів накопичення.

У полі “Паралельно моделей” задається кількість паралельно працюючих систем.

Крім візуальної частини до складу компонента входять поля, які містять посилання на такі об'єкти:

– monitor – монітор, об’єкт класу TransMonitor, що наслідує клас Actor, і в ньому визначений метод rule(), де описаны правила дії, що забезпечують проведення експерименту. Обєкт створюється у компоненті.

– factory – посилання на фабрику моделей, що реалізує інтерфейс IModelFactory, за допомогою якої будуть створюватися моделі.

– diagram – посилання на компонент класу Diagram, що забезпечує відображення результатів моделювання і регресійного аналізу отриманих

даних.

– dispatcher – посилання на об’єкт класу Dispatcher, що забезпечує взаємодію у часі монітора та моделей.

Для запуску процесу моделювання використовується кнопка «Старт», з якою пов'язаний метод startMonitoring(), лістинг 9.5. Завдання цього методу налаштувати діаграму, передати монітору параметри експерименту та посилання на фабрику моделей, а після цього передати монітор диспетчеру та запустити процес моделювання, викликавши метод диспетчера start().

Лістинг 9.5 - Текст методу для запуску процесу моделювання

```
private void startMonitoring() {
    getJButtonStart().setEnabled(false);
    diagram.setHorizontalMaxText(String.valueOf((int) getFinishTime()));
    diagram.setGridByX(getNIntervals());
    diagram.clear();

    getMonitor().setNParallel(getNParallel());
    getMonitor().setNIntervals(getNIntervals());
    getMonitor().setInterval(getInterval());
    getMonitor().setFactory(getFactory());
    getMonitor().setDiagram(diagram);

    getDispatcher().addStartingActor(getMonitor());
    getDispatcher().start();
}
```

9.5.3.1 **Клас TransMonitor**

Об’єкт цього класу використовується для проведення експериментів по виявленню закономірностей перехідного процесу в досліджуваній моделі. Об’єкт, за допомогою фабрики, створює необхідну кількість копій моделі і викликає, через задані проміжки часу, отримує від моделей статистичну інформацію про досліджуваний параметр. Отримані дані накопичуються, а потім можуть бути передані на обробку. Крім того, ці дані в процесі роботи компонента можуть візуалізуватися.

Клас TransMonitor наслідує клас Actor, і в ньому визначений метод rule(), який описує правила дії його об'єктів, що забезпечують реалізацію функцій по проведенню експерименту.

Текст методу rule() наводиться в лістингу 9.6.

Лістинг 9.6 - Правила дії об’єкту monitor

```
public void rule() {
    // Масив для моделей
    ITransProcesable[] modelArray = new ITransProcesable[nParallel];
    for (int i = 0; i < nParallel; i++) {
        // Створюємо моделі і готуємо їх до старту
        modelArray[i] = (ITransProcesable) (getFactory()
                .createModel(getDispatcher()));
        modelArray[i].initForTrans(interval * nIntervals);
    }
    // Готуємося до моніторингу моделей
    float average;
    intervalAverageArray = new double[nIntervals];
    // Цикл по інтервалам усереднення
    for (int i = 0; i < nIntervals; i++) {
        // Ініціалізація накопичувачів інформації
        for (int j = 0; j < nParallel; j++) modelArray[j].resetTransAccum();
        this.holdForTime(interval); // Затримка на довжину інтервалу
        // Усереднення накопичених даних
        average = 0;
        for (int j = 0; j < nParallel; j++) average += modelArray[j].getTransResult();
        average /= nParallel;
        // Запис та індикація результату усереднення
        intervalAverageArray[i] = average;
        getDispatcher().printToProtocol(
                " " + getNameForProtocol() + ":середнє по інтервалу"+ average);
        if (diagram != null)
            diagram.getPainter().drawOvalAtXY(
                    (float) (interval * (i + 0.5)), (float) average, 3, 3);
    }
}
```

У методі rule() спочатку створюється масив, розмірність якого дорівнює кількості паралельно працюючих моделей. У масив заносяться посилання на

моделі, створені за допомогою фабрики. Копії моделей ініціалізуються.

Після створення та ініціалізації моделей монітор починає головний цикл роботи. У цьому циклі монітор ініціалізує накопичувачі інформації в моделях шляхом виклику методу resetAccum(), а потім зупиняється на проміжок часу, який дорівнює довжині інтервалу накопичення інформації.

Після зупинки монітора диспетчер запускає всі моделі, і вони працюють, накопичуючи інформацію протягом інтервалу накопичення.

Потім відновляються правила дії монітора. Він одержує інформацію від кожної з моделей, використовуючи метод getTransResult(), і визначає середнє значення цих результатів. Отримане середнє значення монітор заносить у черговий елемент масиву результатів intervalAverageArray, а також відображає на діаграмі точку, яка відповідає отриманому результату.

9.5.4 *Компонент ParmFinderView*

Цей компонент забезпечує пошук параметрів рівняння регресії для перехідного процесу в досліджуваній моделі. Композиція класу представлена на рисунку 9.9.

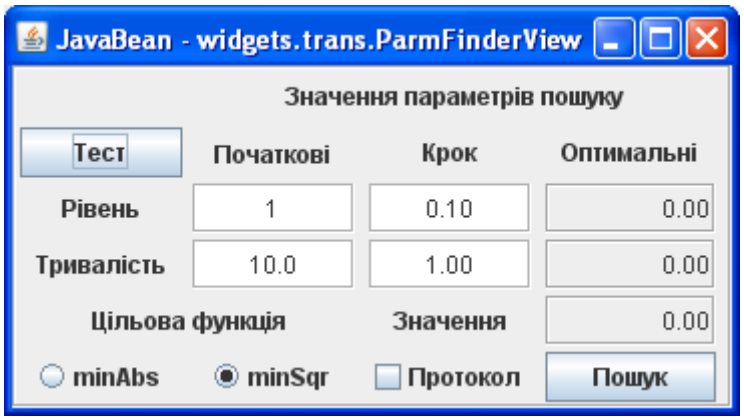

Рисунок 9.9 - Візуальна композиція класу ParmFinderView

Візуальна частина композиції використовується для налаштування й виводу результатів пошуку у вигляді чисел.

Пошук параметрів рівняння регресії може виконуватися як в автоматичному, так і в ручному режимі.

При ручному пошуку користувач оцінює по експериментальних точках значення сталого рівня й час перехідного процесу, вводить ці значення в колонку «Стартовые значения» і натискає кнопку «Тест». Уведені значення передаються компоненту ParamFinder, який відображає на діаграмі лінію регресії, що відповідає введеним параметрам, і розраховує значення цільової функції, що є показником відхилення експериментальних точок від лінії регресії.

В якості цільової функції можна вибирати або суму абсолютних відхилень, або суму квадратів відхилень експериментальних точок від лінії регресії. Для перемикання використовується група радіо кнопок «minAbs» - «minSqr».

Якщо тестувати різні варіанти параметрів лінії регресії можна підібрати

більше менш підходящі значення.

При автоматичному пошуку задаються стартові значення параметрів лінії регресії й крок їхньої зміни при пошуку. При натисканні кнопки «Пошук» ці значення передаються компоненту ParmFinder, який методом координатного спуску робить пошук оптимальних значень параметрів, після чого лінія регресії й значення цільової функції виводяться для знайдених значень параметрів.

За бажанням можна отримати протокол пошуку.

9.5.4.1 **Клас TransParmFinder**

Цей клас забезпечує реалізацію алгоритму координатного спуску при пошуку оптимальних значень параметрів лінії регресії. Для пошук екстремуму по кожній з координат використовується метод чисел Фібоначчі.

Об'єкти розглянутого класу знаходять параметри лінії регресії, рівняння якої визначається співвідношенням 9.9

$$Y(t)=\left(1-e^{-\frac{3t}{tau}}\right)\cdot level \tag{9.9}$$

Ця формула ідентична формулі 9.1. Різниця полягає лише в тому, що замість постійної часу T, тут використовується тривалість перехідного процесу tau=3T.

Розташування лінії регресії щодо експериментальних точок залежить від значень параметрів залежності 9.9 - тривалості перехідного процесу tau, і сталого значення черги, що представлено у формулі змінної level. Оптимальні значення саме цих параметрів і знаходять об’єкти класу TransParmFinder.

9.6 Опис програмного комплексу для лабораторної роботи

Програмний комплекс для даної лабораторної роботи створювався у вигляді пакету testTrans виходячи з того, що він повинен забезпечувати:

– налаштування параметрів моделі – кількість паралельно працюючих систем, довжину інтервалу усереднення за часом, число інтервалів усереднення, параметри вхідного потоку і процесу обслуговування;

– накопичення, обробку і видачу в графічному вигляді середніх значень черги по інтервалам накопичення;

– забезпечувати вибір цільової функції – сума модулів відхилень або сума квадратів відхилень;

– можливість автоматичного пошуку параметрів рівняння регресії;

– можливість пошуку параметрів рівняння регресії вручну;

– індикацію значення цільової функції та відповідної лінії регресії;

– формування та вивід протоколу пошуку.

Класи, які безпосередньо пов’язані з виконанням лабораторної роботи знаходяться в пакеті testTrans проекту AllSimLab.

Клас TransGUI реалізує шар подання програми. До складу шару подання,

окрім компонентів, що забезпечують налаштування параметрів моделі, входять компонент TransMonitorView та компонент ParmFinderView.

Компонент TransMonitorView забезпечує настройку параметрів та проведення експерименту по визначенню переходного процесу.

Компонент ParmFinderView забезпечує пошук параметрів рівняння регресії для перехідного процесу в системі. Початкові дані об'єкт одержує від монітора, а результати обробки виводить на діаграму і в протокол.

9.6.1 *Клас TransGUI*

Клас TransGUI реалізує шар подання програми. До складу шару подання, окрім компонентів, що забезпечують налаштування параметрів моделі, входять компонент TransMonitorView та компонент ParmFinderView.

Компонент TransMonitorView забезпечує настройку параметрів та проведення експерименту по визначенню переходного процесу.

Компонент ParmFinderView забезпечує пошук параметрів рівняння регресії для перехідного процесу в системі. Початкові дані об'єкт одержує від монітора, а результати обробки виводить на діаграму і в протокол.

Візуальна композиція класу представлена на рисунку 9.10.

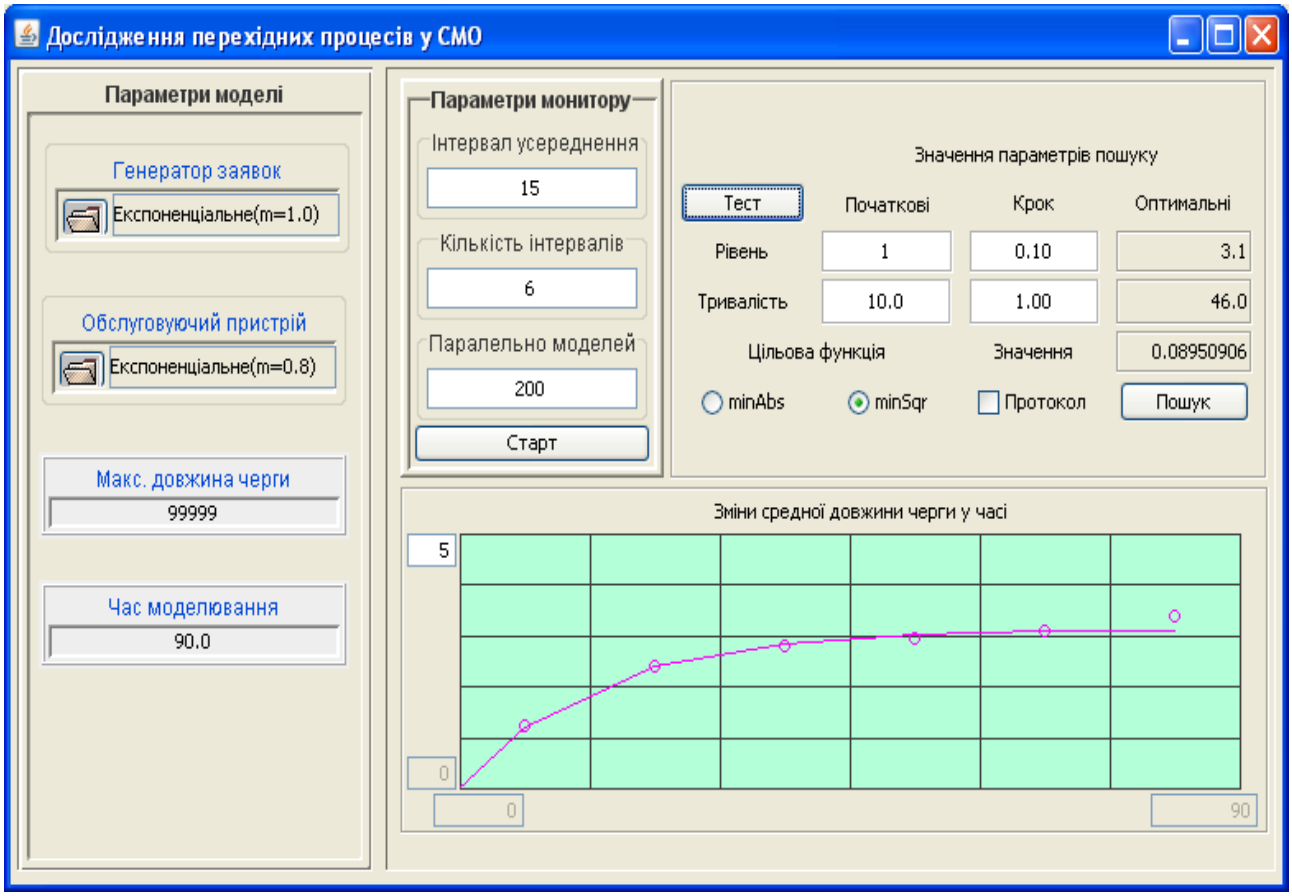

Рисунок 9.10 - Візуальна композиція проекту для виконання лабораторної роботи

Особливість програмної реалізації класу полягає у тому, що тут створюється фабрика моделей і посилання на неї передається монітору. А компоненту ParmFinderView передається посилання на монітор. Окрім того

цим компонентам передається посилання на діаграму. Таким чином компоненти пов’язуються між собою.

Процес моделювання ініціюється монітором, який за допомогою фабрики створює необхідну кількість моделей, після цього періодично отримує від моделей інформацію про середню довжину черги, накопичує ці дані і виводить на діаграму.

Процес пошуку параметрів лінії регресії починається після закінчення роботи диспетчера, як реакція на подію DispatcherFinishEvent.

9.6.2 *Реалізація шару моделі*

9.6.2.1 **Клас TransModelFactory**

Клас фабрики практично нічим не відрізняється від класів фабрик, що були розглянуті у попередніх лабораторних застосуваннях. Єдина особливість полягає у тому, що класи моделі та візуальної частини, що передається до моделі, інші.

9.6.2.2 **Клас TransModel**

Клас TransModel задає структуру моделі і забезпечує створення та ініціалізацію її компонентів. Особливість моделі полягає у тому, що в якості черги використовується об’єкт класу TransProcessQueue, що дозволяє реалізувати у класі моделі інтерфейс ITransProcessable. В лістингу 9.7 наведено реалізації методів інтерфейсу IExperimentable в даному класі.

Лістинг 9.7 – Реалізація інтерфейсу ITransProcessable у класі TransModel

```
public void initForTrans(double finishTime) {
   gui.getChooseDataFinishTime().setText(String.valueOf(finishTime));
}
public double getTransResult() {
   return queue.getAvg();
}
public void resetTransAccum() {
   getQueue().resetAccum();
}
```

9.6.3 *Реалізація шару компонентів моделі*

Для реалізації цього шару потрібно створити класи:

– TransactGenerator, що моделює генератор трансакцій;

– Device, що моделює обслуговуючий прилад.

Подібні класи вже створювалися раніше. Класи для решти компонентів, що входять до сладу моделі реалізовані у фреймворці.

КОНТРОЛЬНІ ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. У чому полягає методика отримання параметрів переходного процесу для довжини черги у СМО?
2. У чому полягає особливість визначення функції регресії для перехідного процесу експоненційної форми?
3. Що таке цільова функція?
4. У чому принципова різниці між методом найменших квадратів та методом найменших модулів?
5. Які чисельні методи можна використовувати для пошуку екстремуму?
6. Назвіть числа Фібоначчі від 1 до 55.
7. Як найти границі області , де находиться екстремум, за допомогою чисел Фібоначчі?
8. Як знайти екстремум функції у середині деякої області за допомогою чисел Фібоначчі?
9. Як знайти екстремум функції у середині деякої області за допомогою методу дихотомії?
10. Як знайти екстремум функції у середині деякої області за допомогою методу золотого перетину.
11. Призначення та методи інтерфейсу Itransprocesable.
12. Особливості класу TransProcessQueue/
13. Призначення класів TransMonitorView та TransMonitor?
14. Яким чином проводиться у моделі усереднення довжини черги за часом ?
15. Яким чином проводиться у моделі усереднення довжини черги за реалізаціями ?
16. У чому полягає метод координатного спуску для пошуку екстремума?
17. Призначення класів ParmFinderView та ParmFinder?

10 ТЕМА 10. БАГАТОФАКТОРНІ ЕКСПЕРИМЕНТИ З МОДЕЛЯМИ

План лекції

– Проблеми, пов’язані з проведення багатофакторних експериментів, та межі їх застосування.
– Багатофакторні експерименти для пошуку екстремуму функції відгуку.
– Матриця планування експерименту на двох рівнях.
– Дробові багатофакторні експерименти.
– Компонент для автоматизованого пошуку екстремуму градієнтним методом.

10.1 Багатофакторні експерименти та межі їх застосування

Багатофакторні експерименти на одному рівні практично нічим не відрізняються від одно факторних екперіментів. Кількість експериментів і методика визначення довірчого інтервалу не залежіть від кількості факторів.

Але багатофакторні експерименти на багатьох рівнях, що проводяться для визначення залежності відгуку від фактору, потребують дуже великої кількості експериментів, формула 10.1

$$n_{експ} = r \cdot p^f \quad , \tag{10.1}$$

де r – кількість повторів на рівні;
p – кількісьб рівнів;
f – кількість факторів.

До того ж візуально відстежувати отримані залежності дуже важо і, відповідно, важко вибирати функції регресії. Внаслідок цього такі експерименти ставлять не часто.

З іншого боку, багатофакторні експерименти для пошуку екстремумів функцій відгуку дуже популярні. Такі експерименти потребують менш трудомісткі, а результати очевидні.

10.2 Багатофакторні експерименти для пошуку оптимального значення відгуку

Планування експериментів при пошуку оптимального значення відгуку, що залежить від багатьох чинників, полягає в наступному.

Перш за все, в просторі факторів визначаються межі області пошуку, всередині якої знаходиться екстремум. При цьому максимально використовують всю наявну апріорну інформацію про досліджувану систему. Межі повинні бути вибрані так, щоб усередині області пошуку був лише один екстремум.

Після визначення меж області, вибирається базова точка (значення факторів) в околиці якої слід провести експерименти по оцінці впливу факторів. Часто точка береться навмання, наприклад, посередині області пошуку .

Далі складається матриця планування повного або дробового факторного експерименту на двох рівнях і проводиться відповідна цим планам серія експериментів.

Отриману в результаті цих експериментів інформацію використовують для обчислення коефіцієнті рівняння регресії і по них визначають напрям найбільшого збільшення (зменшення) функції відгуку у вигляді вектора градієнту.

В напрямку, заданому цим вектором, проводять однофакторний експеримент, визначаючи оптимальне значення функції відгуку на лінії напрямку.

Точка, отримана таким чином, береться в якості базової точки, і для неї знову планується серія експериментів на двох рівнях.

Якщо значення функції відгуку в усіх точках плану вийшли гірше, ніж в базовій точці, значить, оптимум досягнутий. В іншому випадку проводиться нова серія експериментів по визначенню напрямку найбільшої зміни функції відгуку.

10.3 Планування багатофакторних експериментів

Багатофакторні експерименти завжди громіздкі. Навіть якщо багатофакторний експеримент проводити тільки на двох рівнях, то для того щоб перебрати можливі комбінації факторів - потрібно провести 2^n експериментів, тому число експериментів зі зростанням числа факторів різко зростає.

Експерименти, в яких аналізуються всі можливі комбінації факторів, називаються повними. Для скорочення числа експериментів часто аналізуються лише деякі комбінації факторів. Такі експерименти називають дробовими багатофакторними експериментами. Дробові експерименти дають менше інформації, однак вони вимагають і менше витрат для їх проведення.

10.3.1 *Планування повного багатофакторного експерименту на двох рівнях*

План для даного виду експерименту передбачає повний перебір всіх комбінацій факторів на всіх рівнях. Наприклад, при проведенні повного трифакторного експерименту на двох рівнях, буде потрібно 8 експериментів. В результаті проведення експерименту можна отримати рівняння регресії 10.2 з вісьмома коефіцієнтами.

$$y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 + a_{12} x_1 x_2 + a_{23} x_2 x_3 + a_{13} x_1 x_3 + a_{123} x_1 x_2 x_3 \quad (10.2)$$

Для зручності обробки результатів експериментів на двох рівнях фактори нормують таким чином, щоб вони приймали значення тільки +1 та -1. При цьому використовуються співвідношення 10.3.

$$x_0 = \frac{x_{\max} + x_{\min}}{2}; \quad \Delta x = \frac{x_{\max} - x_{\min}}{2}; \quad x = \frac{x - x_0}{\Delta x} \tag{10.3}$$

При такій заміні координат, у разі, коли x = xmax, отримаємо x = 1. Якщо x = xmin, то x = -1

Внаслідок такої заміни план проведення експериментів може бути представлений у вигляді матриці, кожен елемент якої буде або +1, або -1. Для спрощення написання матриці, в ній проставляються тільки знаки одиниць.

У таблиці 10.1 представлена матриця планування повного трифакторного експерименту на двох рівнях.

Таблиця 7.1 - Матриця планування повного трифакторного експерименту на двох рівнях.

Номер експерименту	x_1	x_2	x_3	$y_{экс.}$
1	-	-	-	
2	-	-	+	
3	-	+	-	
4	-	+	+	
5	+	-	-	
6	+	-	+	
7	+	+	-	
8	+	+	+	

Якщо функція $y_{екс}$ схильна до флуктуацій, то на кожному рівні слід провести серію експериментів (бажано однакового розміру), а в таблицю занести середні значення.

10.3.2 *Властивості матриці планування експериментів*

Матриця планування експерименту повинна мати властивості нормування, симетричності і ортогональності.

Властивість нормування вимагає, щоб елементи кожного стовпця приймали значення тільки +1 або -1. Цю умову прийнято записувати у вигляді співвідношення 10.4.

$$\sum_{i=1}^{n} x_i^{\,2} = n \tag{10.4}$$

де - n число рядків матриці планування.

Властивість симетричності вимагає, щоб для кожного стовпця кількість позитивних одиниць дорівнювала кількості негативних. Цю умову прийнято записувати у вигляді співвідношення 10.5.

$$\sum_{i=1}^{n} x_i = 0 \tag{10.5}$$

Властивість ортогональності по стовпцях вимагає виконання співвідношення 10.6.

$$\sum_{i=1}^{n} x_i^k x_i^\ell = 0 \tag{10.6}$$

де k і l – номери двох будь-яких стовпчиків у матриці, і $k \neq l$.

10.3.3 *Знаходження коефіцієнтів рівняння регресії*

Після проведення експериментів у відповідності з матрицею планування та визначення $y_{екс}$ для кожного рядка матриці можна знайти коефіцієнти рівняння регресії 10.2. Для цього застосуємо метод найменших квадратів.

Для визначення коефіцієнтів необхідно знайти похідні від функції 10.7 по невідомим коефіцієнтам a, прирівняти ці похідні нулю і розв'язати отриману систему рівнянь.

$$\sum_{i=1}^{n} (y_{екс}^i - (a_0 + a_1 x_1^i + a_2 x_2^i + a_3 x_3^i + a_{12} x_1^i x_2^i + a_{23} x_2^i x_3^i + a_{13} x_1^i x_3^i + a_{123} x_1^i x_2^i x_3^i))^2) \tag{10.7}$$

Рішення отриманої системи рівнянь не складає труднощів, оскільки завдяки властивостям матриці планування експериментів кожне рівняння цієї системи перетворюється у вираз для визначення коефіцієнта, за яким знаходилася похідна.

Як наслідок отримуємо вирази 10.8.

$$\begin{aligned}
&a_0 = \frac{\sum_{i=1}^{n} y_{екс}^i}{n};\\
&a_1 = \frac{\sum_{i=1}^{n} (y_{екс}^i \cdot x_1^i)}{n}; a_2 = \frac{\sum_{i=1}^{n} (y_{екс}^i \cdot x_2^i)}{n}; a_3 = \frac{\sum_{i=1}^{n} (y_{екс}^i \cdot x_3^i)}{n};\\
&a_{12} = \frac{\sum_{i=1}^{n} (y_{екс}^i \cdot x_1^i \cdot x_2^i)}{n}; a_{13} = \frac{\sum_{i=1}^{n} (y_{екс}^i \cdot x_1^i \cdot x_3^i)}{n}; a_{32} = \frac{\sum_{i=1}^{n} (y_{екс}^i \cdot x_3^i \cdot x_2^i)}{n};\\
&a_{123} = \frac{\sum_{i=1}^{n} (y_{екс}^i \cdot x_1^i \cdot x_2^i \cdot x_3^i)}{n};
\end{aligned} \tag{10.8}$$

Для розрахунку коефіцієнтів рівняння регресії за формулами 10.8, зручно використовувати розширену матрицю планування. Її можна отримати з матриці, представленої в табліці 10.1, шляхом додавання стовпців для добутків значень факторів. Така матриця представлена в таблиці 10.2.

Таблица 10.2 – Расширенная матрица планирования полного трехфакторного эксперимента на двух уровнях

Номер експеримента	x_1	x_2	x_3	x_1x_2	x_1x_3	x_2x_3	$x_1x_2x_3$	$y_{экс.}$
1	-	-	-	+	+	+	-	
2	-	-	+	+	-	-	+	
3	-	+	-	-	+	-	+	
4	-	+	+	-	-	+	-	
5	+	-	-	-	-	+	+	
6	+	-	+	-	+	-	-	
7	+	+	-	+	-	-	-	
8	+	+	+	+	+	+	+	

Слід зазначити, що додаткові стовпці розширеної матриці мають ті ж властивості, що й основні.

Додаткові стовпці матриці спрощують обчислення коефіцієнтів, що визначають спільний вплив факторів на функцію відгуку, оскільки вони визначають знаки, з якими повинні підсумовуватися значень останнього стовпця при визначенні відповідних коефіцієнтів рівняння регресії.

10.3.4 *Дробові багатофакторні експерименти*

Повний багатофакторний експеримент дозволяє оцінити не тільки ступінь впливу окремих факторів, але також і їх взаємодії. Інформація про спільний вплив факторів корисна, але під час пошуку градієнта вона не потрібна, оскільки вектор градієнта визначається в околиці робочої точки з нульовими координатами, і тому всі складові похідних від рівняння регресії по кожній із змінних, які відповідають парним або потрійним складовим обернуться в 0. Тобто коефіцієнти, які визначаються виразами 10.8, у яких кількість індексів більш 1 для пошуку градієнта не використовуються. У рівнянні регресії 10.9 дрібним шрифтом показані складові, якими можна знехтувати.

$$y = y_0 + a_1x_1 + a_2x_2 + a_3x_3 + a_{1,2}x_1x_2 + a_{2,3}x_2x_3 + a_{1,3}x_1x_3 + a_{1,2,3}x_1x_2x_3 \quad (10.9)$$

Таким чином, кількість коефіцієнтів рівняння регресії стає менше. Внаслідок цього кількість рядків у матриці коефіцієнтів можна скоротити. План скороченого експерименту називається дробової реплікою. Але довільно число експериментів скорочувати не рекомендується. Для того щоб властивості матриць планування в дробових репліках зберігалися, число експериментів можна скорочувати в число раз, що дорівнює ступені двійки. Як результат отримуємо, наприклад, 1/2 або 1/4 або 1/8 репліку.

Використання дробових реплік особливо ефективно у випадку великого кількості факторів. Наприклад, якщо число факторів 7, то план повного факторного експерименту передбачає проведення 27 = 128 експериментів. Однак для оцінки впливу кожного з факторів досить знайти y0 і 7 коефіцієнтів,

тобто потрібно всього 8 рівнянь і, отже, 8 рядків матриці планування, що є 1/16 реплікою. При цьому число експериментів скоротиться в 16 разів.

10.3.5 *Матриця планування для дробових реплік*

Найпростіший спосіб побудови матриці планування для дробової репліки - це використання генеруючих співвідношень.

Порядок побудови матриці наступний.

Визначаємо необхідну кількість рівнянь для обчисленя коефіцієнтів рівняння регресії. Наприклад, при кількості факторів 5 таких рівнянні потрібно 6.

Далі визначається найближче, але не менше, число рівне ступені двійки. Для випадку п'яти факторів таким числом буде 8.

Наступний крок - це побудова матриці плану повного факторного експерименту, в якій число рядків дорівнює отриманму числу. У нашому випадку це буде повний план для трьох факторів, тому 2^3 = 8.

Отримана матриця є частиною створюваної і визначає стовпці плану для кількох перших факторів. У розглянутому прикладі це будуть стовпці для перших трьох факторів – x1, x2, x3.

Стовпці для наступних факторів отримують як добуток початкових стовпців, причому, рекомендується в першу чергу використовувати добутки з максимально можливим числом співмножників. Ці добутки, на основі яких отримують нові стовпчики - називають генеруючими співвідношеннями. Так, у розглянутому прикладі, потрійний добуток перших факторів дозволить знайти стовпець для четвертого фактора. Генеруюче співвідношення буде таким x4 = x1x2x3. Для п'ятого фактора генеруюче співвідношення може бути будь-яким з трьох наступних: x5 = x1x2, x5 = x1x3, x5 = x2x3. Взявши, наприклад, перше з них, отримаємо матрицю планування для 1/4 репліки 5-и факторного експерименту у вигляді таблиці 10.3.

Таблиця 10.3 – Матриця планування для 1/4 репліки 5-и факторного експерименту

Номер експерименту	x_1	x_2	x_3	$x_4=x_1x_2x_3$	$x_5=x_1x_2$	$y_{экс.}$
1	-	-	-	-	+	
2	-	-	+	+	+	
3	-	+	-	+	-	
4	-	+	+	-	-	
5	+	-	-	+	-	
6	+	-	+	-	-	
7	+	+	-	-	+	
8	+	+	+	+	+	

Другий спосіб отримання матриці планування для дробового факторного експерименту полягає у використанні визначального контрасту. Визначальним контрастом називається добуток лівої та правої частин генеруючого

співвідношення. Значення визначальних контрастів прирівнюють +1 або -1.

У нашому випадку, для +1, визначальними контрастами будуть такі співвідношення: x4x1x2x3 = 1 і x5 x1x2 = 1

Для того, щоб скористатися визначальним контрастом, потрібно побудувати матрицю повного факторного експерименту і вибрати в ній ті рядки, для яких виконуються умови, що задаються визначальними контрастами.

10.4 Компонент «Оптимізатор»

Завдання цього компонента полягає в тому, щоб шляхом проведення експериментів з моделлю, знайти оптимальні значення трьох факторів, що забезпечують мінімум цільової функції. Компонент реалізує алгоритм пошуку екстремуму описаний в підрозділі 10.2.

Компонент реализовано як комбінацію двох класів.

Інтерфейс користувача для компонента реалізує клас OptimizerView. Візуальна композиція цього класу показана на рисунку 10.1.

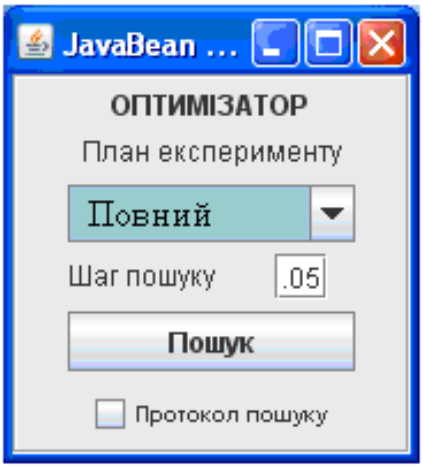

Рисунок 10.1 – Візуальна композиція компонента «Оптимізатор»

Список «План експерименту» забезпечує вибір плану проведення експериментів для визначення напрямку пошуку оптимальних значень фактору. Елемент управління містить три варіанти плану - «Повний», «Дробовий +1», «Дробовий -1», що відповідає повному плану для трифакторного експерименту на двох рівнях і двом половинним реплікам.

Текстовий елемент «Крок пошуку» задає мінімальний крок руху в напрямку градієнта. Кроки зміни налаштувань регулятора прямо пропорційні цьому кроку, але залежать також від компонент вектора градієнта і поточного числа Фібоначчі.

Елемент «Протокол пошуку» дозволяє, у випадку необхідності, отримати на консолі протокол пошуку екстремуму.

Крім панелі налаштування, компонент містить об'єкт класу Optimizer, який забезпечує пошук оптимальних налаштувань.

При натисканні кнопки «Налаштувати» викликається метод, який налаштовує об'єкт класу Optimizer відповідно до візуальної частини, після чого починається процес автоматичного пошуку оптимальних значень фактору.

Клас Optimizer вирішує задачу автоматичного пошуку оптимальних значень факторів градієнтнім методом, що використовує числа Фібоначчі для пошуку екстремуму в напрямку градієнта.

КОНТРОЛЬНІ ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. У чому полягає особливість багатофакторних експериментів на багатьох рівнях?
2. Багатофакторні експерименти для пошуку екстремуму функції відгуку.
3. Матриця планування експерименту на двох рівнях.
4. Властивості матриці планування експериментів
5. Методика визначення коефіцієнтів лінії регресії.
6. Дробові багатофакторні експерименти.
7. Що таке генеруючи співвідношення і як їх використовувати для побудови матриці дробового багатофакторного експерименту?.
8. Що таке визначальний контраст і як його використовувати для побудови матриці дробового багатофакторного експерименту?
9. Компонент для автоматизованого пошуку екстремуму градієнтним методом.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Томашевський В.М. Моделювання систем. – К.: Видавнича група BHV, 2005. -352 с.:іл.
2. Стеценко, І.В. Моделювання систем: навч. посіб. [Електронний ресурс, текст] / І.В. Стеценко ; М-во освіти і науки України, Черкас. держ. технол. ун-т. – Черкаси : ЧДТУ, 2010. – 399 с. ISBN 978-966-402-073-9
3. Shannon R. Systems simulation The art and science, NY Mc Grow Hill, 1978. – 424 с.
4. Великодній С.С. Імітаційне моделювання. Навчальний посібник. Одеська державна академія холоду, 2011. -190с.
5. Simulation Modeling and Analysis (Third Edition), Law and Kelton, McGraw Hill, 2000.– 749 с.
6. Теорія систем масового обслуговування : навч. посібник / А. Л. Литвинов ; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. – Харків : ХНУМГ ім. О. М. Бекетова, 2018. – 141 с.
7. Теорія ймовірностей та математична статистика: навчальний посібник / О. І. Огірко,. Н. В. Галайко. – Львів: ЛьвДУВС, 2017. – 292 с.
8. Конспект лекцій з курсу «Планування і обробка результатів експерименту» Харків. нац. унт міськ. госп-ва ім. О. М. Бекетова ; уклад. : Л. А. Назаренко. – Харків : ХНУМГ ім. О. М. Бекетова, 2018. – 163 с. https://sds.kname.edu.ua/images/doc/nazarenko/%D0%9F%D1%96%D0%9E%D0%A0%D0%95_%D0%9A%D0%9B_2018-1-90.pdf