Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Кафедра математики факультету інформатики

Кваліфікаційна робота

освітній ступінь - бакалавр

на тему: «**ОПТИМАЛЬНЕ КЕРУВАННЯ СИСТЕМАМИ МАСОВОГО ОБСЛУГОВУВАННЯ (OPTIMAL CONTROL FOR QUEUING SYSTEMS)**»

Виконала: студентка 4-го року навчання,

Освітньої програми «Прикладна математика», 113

Мостова Марія Сергіївна

Керівник Крюкова Г. В.

кандидат фіз.-мат. наук, доцент

Рецензент

_____

(прізвище та ініціали)

Кваліфікаційна робота захищена з оцінкою

_____

Секретар ЕК

_____

«____» _____

20____ р.

Київ – 2024

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра математики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри математики,

_____

„____”_____2023 р

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студенту факультету інформатики 4 курсу Мостовій Марії Сергіївні

ТЕМА: Оптимальне керування системами масового обслуговування (Optimal control for queuing systems)

Зміст ТЧ до кваліфікаційної роботи:

Зміст

Анотація

Вступ

Теоретична частина

Практична частина

Висновки

Список використаних джерел

Додатки

Дата видачі „___” _____ 2023 р. Керівник _____(підпис)

Завдання отримав _____(підпис)

Календарний план виконання роботи

Тема: Оптимальне керування системами масового обслуговування (Optimal control for queuing systems)

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проєкту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на бакалаврську роботу	27.10.2023	
2.	Опрацювання матеріалів	31.01.2024	
3.	Створення програми	07.05.2024	
4.	Написання теоретичної частини	14.05.2024	
5.	Коригування роботи	27.05.2024	

Студент: Мостова М. С.

Керівник: Крюкова Г. В.

“_____” _____

ЗМІСТ

ANNOTATION

This paper studies how to find an optimal strategy for certain type of queuing systems. It focuses on queuing systems with limited queue which are based on Markov chains with discrete time. The paper covers the definition of the mean total processing cost of the system and an optimality equation. The goal is to find the way to navigate incoming requests which would minimize the mean total processing cost of the system.

An algorithm to compute an optimal strategy for systems with specific parameters is implemented with Python. The libraries Itertools, Numpy, SciPy are used. An example of the work is demonstrated with certain set of parameters.

Keywords: Markov process, queuing system, optimal strategy, optimal policy, optimality equation, Bellman equation, mean total processing cost.

ВСТУП

У сучасному цифровому світі, де кількість інформації та задач для обробки зростає з неймовірною швидкістю, правильне керування системами масового обслуговування є надзвичайно важливим завданням для забезпечення безперервної та ефективної роботи різноманітних сервісів, виробничих процесів тощо. Завдяки правильному керуванню можна досягти підвищення продуктивності систем. Коректне управління потоком запитів дозволяє раціонально розподіляти задачі між обслуговуючими пристроями, тим самим уникаючи перевантаження системи. Таким чином, розробка алгоритмів оптимізації керування системами є актуальним питанням для забезпечення ефективності роботи систем.

Мета цієї роботи – розглянути системи масового обслуговування на основі ланцюгів Маркова, визначити середні витрати системи та рівняння оптимальності, а також реалізувати алгоритм знаходження оптимальної стратегії керування системою і визначити можливі способи покращення алгоритму.

Об’єктом дослідження є системи масового обслуговування, в основі яких лежить ланцюг Маркова, з дискретним часом та обмеженою чергою, де вхідний потік запитів розподілений за законом Пуассона, а час обслуговування розподілений експоненційно.

Предметом дослідження є алгоритм знаходження оптимальної стратегії керування, який би забезпечив мінімізацію середніх витрат системи.

Робота складається з двох розділів.

Перший розділ містить у собі теоретичну інформацію, необхідну для розуміння алгоритму. Там наведено такі базові визначення, як випадковий процес, ланцюг Маркова, система масового обслуговування. Також описано постановку задачі, де детально визначено тип системи та її параметри,

множину її станів і рішень та керування нею. Визначено середні витрати системи, які необхідно мінімізувати, а також виведено рівняння оптимальності.

У другому розділі наведено практичну реалізацію алгоритму. В цьому розділі описано покрокову роботу алгоритму. Наведено опис розробленої програми, яка реалізує даний алгоритм, визначено спосіб введення параметрів для задання системи та детально описано методи. В кінці показано роботу алгоритму пошуку оптимальної стратегії на прикладі з заданими параметрами, в результаті чого отримано середні витрати системи та оптимальну стратегію для даної системи.

РОЗДІЛ 1

ТЕОРЕТИЧНА ІНФОРМАЦІЯ

1.1 Основні визначення

Для кращого розуміння подальшого матеріалу переглянемо базові означення, розпочавши з випадкових процесів.

Означення 1.1.1. *Множина випадкових величин* $\xi = (\xi^t : t \in T)$*, тобто:*

$$\xi = (\xi^t : (\Omega, F, P_r) \to (X, \Sigma), t \in T),$$

де (Ω, F, P_r) *– імовірнісний простір,* (X, Σ) *– вимірний простір,* $T \neq \emptyset$*, називається випадковим процесом*. [7, 11 р.]

Випадкові процеси можна класифікувати по-різному, зокрема зважаючи на множину моментів часу.

Якщо множиною моментів часу є деякий інтервал на дійсній прямій, то випадковий процес називатимемо *безперервним за часом*. Якщо ж множина моментів часу процесу є зліченною або скінченною, то процес називається *дискретним*.

Розглянемо поняття процесу Маркова. Для цього визначимо наслідок n-го випробування через ξ_n, а стан процесу через x_n, де $n \geq 0$.

Означення 1.1.2. *Випадковий процес називається марковським процесом, якщо його наступний стан залежить лише від стану, в якому він знаходиться в момент часу, і не залежить від попередніх станів. Математично це можна записати таким чином:*

$$P\left(\xi_{n+1} = x_j \,/\, \xi_0 = x_{i_0}, \dots, \xi_{n-1} = x_{i_{n-1}}, \xi_n = x_{i_n}\right) = P\left(\xi_{n+1} = x_j \,/\, \xi_n = x_{i_n}\right) [2, 630\ p.]$$

Властивість, описана вище, називається *властивістю Маркова*. Вона означає, що для прогнозування поведінки системи в майбутньому до уваги береться тільки теперішній її стан, а не всі попередні. Отже іншими словами, випадковий процес називається марковським, якщо він задовольняє властивість Маркова.

Звідси отримуємо наступне визначення: *ланцюг Маркова* є дискретним за часом марковським процесом зі скінченною або зліченною множиною станів [7, 14 р.]. Марковський ланцюг задається множиною станів X та матрицею переходу P, яка складається з імовірностей переходу зі стану в стан за один крок.

Система масового обслуговування – система, яка призначена для обробки вимог, які до неї надходять, за допомогою обслуговуючих пристроїв. Зазвичай вважається, що вхідний потік вимог є випадковим.

Системи масового обслуговування можна класифікувати різними способами, одним із яких є класифікація за довжиною черги. Можна поділити системи на такі три види:

1. Системи без черги, в яких у разі, якщо при надходженні запиту всі обслуговуючі пристрої зайняті, запит відхиляється.
2. Системи з обмеженою чергою, в яких запит, який надходить до системи в стані, де усі обслуговуючі пристрої зайняті та черга заповнена, буде відхилено.
3. Системи з необмеженою чергою, в яких будь-який запит, який надходить до системи в стані, де всі пристрої зайняті, стає в чергу, місткість якої необмежена.

В подальшому нас цікавитимуть саме системи з обмеженою чергою.

1.2 Постановка задачі

Системи, які ми розглядатимемо надалі, можна визначити як ланцюг Маркова з обмеженою чергою. Будемо позначати їх таким чином: *M/M/K/(N −K)*, де $K \leq N < \infty$. Довжина черги є $N - K$, де N – місткість системи, а K – кількість обслуговуючих пристроїв.
Вхідний потік розподілений за законом Пуассона з параметром λ, час обслуговування експоненційний з параметром μ_k, де k приймає значення від 1 до K.

Зауважимо, що дисципліна черги в даних системах працюватиме за правилом FIFO, а отже запит, який надійшов першим, буде опрацьований першим.

Визначимо вартість обслуговування черги як $c_0 q$, де c_0 – вартість очікування, q – довжина черги, а вартість обслуговування запиту позначимо як c_k для k-го пристрою з інтенсивністю μ_k, $c_0 \geq 0$ і $c_k \geq 0$.

Керування системою полягає в тому, щоб протиставити кожному стану системи рішення таке, щоб забезпечити ефективну роботу пристроїв та мінімізувати вартість обробки запитів.

Для кожного запиту, який надходить у систему, існують такі можливі рішення:

1. Відправити запит на обробку, якщо принаймні один обслуговуючий пристрій вільний.
2. Відправити запит у чергу, якщо всі пристрої зайняті, але черга ще не заповнена до максимуму.
3. Відхилити запит у разі, якщо всі пристрої зайняті та черга заповнена.

Запит може бути відхилено виключно за умови, що вільних пристроїв та місць у черзі немає, інакше запит обов'язково має бути надісланий до обслуговуючого пристрою або в чергу.

Звідси, стратегією називатимемо послідовність рішень, описаних вище, поставлених у відповідність кожному стану з множини станів X.

Так як множини станів та рішень є скінченними, множина стратегій також є скінченною. А отже, існує така, що мінімізує середні витрати системи.

Тож, задача – мінімізувати середні витрати системи.

1.3 Середні витрати системи

Уточнимо систему та детально опишемо простори станів та рішень системи. Введемо такі позначення: *Q(t)* – довжина черги в момент часу t, а *D(t)* – вектор, який позначає, чи вільний пристрій в момент часу *t:*

$$D_k(t) = \begin{cases} 0, & \text{якщо } k\text{-й пристрій вільний в момент часу } t, \\ 1, & \text{в іншому випадку.} \end{cases} [8, 391\ p.]$$

Тепер позначимо процес обслуговування як *{Z(t)} = {(X(t), U (t))}*, процес спостереження як *{X(t)}={(Q(t), D(t))}*, а також випадковий процес кількості вимог у системі $L(x) = Q(t) + \sum_{1 \le k \le K} D_k(t)$.

Множиною станів є $X = \mathbb{N} \times \{0, 1\}^K$, де $\mathbb{N}$ приймає значення від 0 до N – K. Стан системи визначимо як $x = (q, d_1, \ldots, d_k)$.

Надалі введемо компоненти $J_0(x) = \{j: d_j(x) = 0\}$ та $J_1(x) = \{j: d_j(x) = 1\}$, які зберігатимуть номери вільних та зайнятих приладів в стані *x* відповідно.

Суми інтенсивностей роботи вільних та зайнятих пристроїв у стані *x* будемо зберігати в змінних $M_0(x)$ та $M_1(x)$ відповідно, отже:

$$M_0(x) = \sum_{j \in J_0(x)} \mu_j\,, \qquad M_1(x) = \sum_{j \in J_1(x)} \mu_j.$$

Процес керування визначимо так: $\{U(t)\}=\{(U_j(t): j \in \{0\} \cup J_1(x))\}$, де $U_0(t)$ означатиме номер пристрою, який прийматиме запит на опрацювання у разі його надходження, а $U_j(t)$ при $j \in J_1(x)$ означатиме номер пристрою, який вмикатиметься в роботу в разі закінчення роботи *j*-го пристрою. [8, 391 р.]

Визначимо множину рішень, яких набуватиме процес.

В декартовому добутку

$$A(x) = \underset{j \in \{0\} \cup J_1(x)}{\times} A_j(x),$$

компоненти якого визначаються таким чином:

$$A_0(x) = \begin{cases} J_0(x) \cup \{0\}, & \text{якщо } q(x) < N - K \text{ в стані } x \\ J_0(x), & \text{якщо } q(x) = N - K \text{ в стані } x \end{cases},$$

$$A_1(x) = \{j\} \cup A_0(x),$$

це буде множина таких значень: $a = (a_j : j \in \{0\} \cup J_1(x))$.

А отже, звідси бачимо, що множина можливих рішень для даної системи містить значення від 0 до K, де «0» означає не надсилати запит на опрацювання до пристроїв, а «k» - відправити запит на обробку до k-го сервера. [8, 391 р.]

Перейдемо до визначення функціоналу, який ми маємо мінімізувати.

Для цього введемо додаткову компоненту $e(x) = (\underbrace{0, \dots}_{i}, 1, \underbrace{0, \dots, 0}_{K-i})$ у вигляді вектора, заповненого нулями, окрім одиниці на *i*-й позиції, а також визначимо рівень втрат для стану *x* як $\mathrm{c}(x) = \mathrm{c}_0 q + \sum_{j \in J_1(x)} c_j$.

Тепер ми можемо визначити інтенсивності переходів таким чином:

$$\lambda_{xy}(a) = \begin{cases} \lambda, & \text{для } y = x + e_{a0}, \\ \mu, & \text{для } y = x - e_j + \mathbb{1}_{\{q(x) > 0\}}(-e_0 + e_{aj}), \ j \in J_1(x), \\ 0, & \text{в інших випадках.} \end{cases}$$

Звідси, функціонал, який необхідно мінімізувати, виглядає так:

$$Y(t) = \int_0^t (\mathrm{c}_0 Q(u) + \sum_{1 \le k \le K} c_k D_k(u))\, du.$$

Позначимо стратегію через δ, вона визначатиме ймовірнісний розподіл процесу, а також відповідне до розподілу математичне сподівання $E_x{}^{\delta}$. Таким чином, задача полягає в мінімізації значення середніх витрат системи:

$$g(x; \delta) = \lim_{t \to \infty} \frac{1}{t} E_x^{\delta} Y(t) \quad [8, 392\ p.]$$

1.4 Оптимальна стратегія

Справедливе твердження, що для вище описаної системи існують середні витрати системи $g = inf_{\delta}\, g(x; \delta)$ та функція цінності $v = \{v(x): x \in X\}$, причому g не залежить від стану *x*.

Тоді пара *g*, *v* задовольняє рівняння оптимальності:

$$v(x) = \min_{a \in A(x)} b\big(x, a; v(x)\big), \ x \in X,$$

де функція *b(x, a; v(x))*, яку треба мінімізувати, називається функцією Беллмана для даної моделі [8, 393 p.].

Оптимальна стратегія $f = \{f(x): x \in X\}$ визначатиметься таким чином:

$$f(x) = \underset{a \in A(x)}{arg\,min}\, b\big(x, a; v(x)\big).$$

Введемо визначення мінімальних сумарних витрат до моменту часу t:

$$V(x,t) = \underset{\delta}{inf} E_x^{\delta} Y(t).$$

Для малого часу h рівність виглядатиме таким чином:

$$V(x, t+h) = c(x)h + \big(1 - \big(\lambda + M_1(x)\big)h\big)V(x,t) + \lambda h \underset{a_0 \in A_0(x)}{min} V\big(x + e_{a_0}, t\big)$$
$$+ \mathbb{1}_{\{q(x)=0\}} \sum_{j \in J_1(x)} \mu_j h V\big(x - e_j, t\big)$$
$$+ \mathbb{1}_{\{q(x)>0\}} \sum_{j \in J_1(x)} \mu_j h \underset{a_j \in A_j(x)}{min} V\Big(x - e_j - e_0 + e_{a_j}, t\Big),$$

де перший доданок означає вартість обробки всіх запитів у системі протягом часу h, другий – сумарну вартість обробки всіх запитів у системі протягом часу t за умови, що в системі не відбулося змін, а інші доданки означають сумарну вартість обробки всіх запитів у системі протягом часу t у разі, якщо новий запит надходить у систему до завершення обслуговування, та у разі, якщо опрацьований запит залишає систему до надходження нового [8, 393 р.].

Щоб спростити рівняння, введемо наступні компоненти:

$$T_0 v(x) = \min[v(x + e_k): k \in A_o(x)],$$

$$T_j v(x) = \begin{cases} T_0 v\big(x - e_j - e_0\big) \text{ для } j \in J_1(x), q(x) > 0, \\ v\big(x - e_j\big) \text{ для } j \in J_1(x), q(x) = 0, \\ v(x) \quad \text{для } j \in J_0(x). \end{cases}$$

Підставимо компоненти в рівняння, наведене вище. Зробивши деякі перетворення і вважаючи, що h прямує до нуля, отримуємо таке рівняння:

$$\frac{\partial V(x,t)}{\partial \mathrm{t}} = -\big(\lambda + M_1(x)\big)V(x,t) + c(x) + \lambda T_0 V(x,t) + \sum_{j \in J_1(x)} \mu_j T_j V(x,t) \quad [8, 393\ p.].$$

Звідси, якщо t прямує до нескінченності, рівняння зводиться до такого:

$$V(x,t) \approx gt + v(x).$$

Після деяких перетворень, маємо рівняння оптимальності:

$$v(x) = \frac{1}{\lambda + M}\left(c(x) + \lambda T_0 v(x) + \sum_{j \in J_1(x)} \mu_j T_j v(x) - g \right).$$

Звідси можемо зробити висновок, що для цього випадку функція Беллмана, яку треба мінімізувати, прийматиме такий вигляд:

$$b(x,k) = v(x + e_k), \qquad k \in A_0(x). \quad [8, 394\ p.]$$

РОЗДІЛ 2

ПРОГРАМНА РЕАЛІЗАЦІЯ

2.1 Опис алгоритму

Розглянемо роботу алгоритму.

Алгоритм [8] базується на методі послідовних наближень Говарда [3]. Він складається з двох основних частин.

У першій частині відбувається обчислення середніх витрат системи для заданої стратегії шляхом розв’язання рівняння оптимальності, описаного вище.

У другій частині відбувається пошук нової стратегії, яка би мінімізувала значення середніх витрат системи.

В разі знаходження нової оптимальної стратегії алгоритм переходить на нову ітерацію першої частини, а коли стратегію не можна покращити – зупиняється.

Мета алгоритму полягає в тому, щоб у відповідність кожному стану системи поставити рішення таке, щоб мінімізувати середні витрати системи g.

2.2 Опис програми

Алгоритм був реалізований за допомогою мови Python.

Для складних математичних обчислень було застосовано такі бібліотеки: itertools, numpy, scipy.optimize.

Розглянемо, як працює програма.

Робота з нею розпочинається зі вводу параметрів для створення системи. Ввід параметрів відбувається через консоль. Параметри, які необхідно ввести: N – місткість системи (кількість пристроїв та кількість місць у черзі), K – кількість місць у черзі, lam – інтенсивність надходження запитів, mu – вектор інтенсивностей роботи кожного пристрою, c – вектор вартостей роботи кожного пристрою.

Система створюється за допомогою методу create_system(N, K, lam, mu, c). Метод приймає на вхід параметри, введені вище. В цьому методі задається система зі введеними параметрами, визначаються всі її можливі стани за допомогою метода get_states() та задається початкова стратегія.

Для визначення оптимальної стратегії викликається метод policy_evaluation(). В цьому методі обчислюються середні витрати системи для заданої стратегії. Для обчислення коефіцієнтів рівнянь, необхідних для знаходження середніх витрат, застосовуються функції T0(x), Tk(x), M0(x), c(x), first_coef(x), second_coef(x) та third_coef(x). Вони приймають на вхід стан системи та повертають числове значення.

Всередині методу policy_evaluation() застосовується метод subalgorithm(g, v). Цей метод покращує (отже, мінімізує) значення середніх витрат системи для конкретної стратегії, не змінюючи її. Приймає на вхід значення середніх витрат системи та коефіцієнти для його обчислення.

В кінці методу застосовується метод policy_improvement(). Тут відбувається пошук іншої стратегії, для якої витрати системи будуть меншими в конкретному стані. Метод ставить у відповідність кожному стану системи

нове рішення, якщо це необхідно, та повертає булеве значення залежно від того, чи було покращено стратегію. Якщо стратегію було покращено, переходимо на нову ітерацію policy_evaluation().

Програма повертає значення середніх витрат системи та оптимальну стратегію, де кожному стану системи у відповідь протиставлене рішення.

2.2 Робота програми на прикладі

Розглянемо систему з такими параметрами: повна місткість системи N = 4, кількість серверів K = 2, інтенсивність вхідного потоку λ = 3, вектор інтенсивностей роботи серверів μ = (6, 11), вектор вартостей роботи серверів c = (3, 3).

Введемо ці параметри через консоль.

Система з цими параметрами буде створена за допомогою методу create_system(N=4, K=2, lam=3, mu=[6, 11], c=[3, 3]).

В цьому методі множина можливих станів системи X буде записана у глобальну змінну states, а початкова стратегія f буде зберігатися в змінній policy.

Основуючись на заданих параметрах, отримуємо таку множину станів X = {(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1), (2, 0, 0), (2, 0, 1), (2, 1, 0), (2, 1, 1)}.

Звідси множина можливих рішень A = {0, 1, 2}, де 0 – не робити нічого, а 1, 2 – відправляти запит на обробку 1 і 2 серверам відповідно.

На початковому етапі в якості стратегії ми ставимо у відповідність кожному стану номер першого вільного сервера, готового до опрацювання запиту при його надходженні, або 0, якщо обидва сервери зайняті. Отже, початкова стратегія виглядатиме таким чином: f = {(0, 0, 0): 1, (0, 0, 1): 1, (0, 1, 0): 2, (0, 1, 1): 0, (1, 0, 0): 1, (1, 0, 1): 1, (1, 1, 0): 2, (1, 1, 1): 0, (2, 0, 0): 1, (2, 0, 1): 1, (2, 1, 0): 1, (2, 1, 1): 0}.

Застосовуємо метод policy_evaluation(). Через декілька ітерацій отримуємо середні витрати системи g = 0.0615 та оптимальну стратегію:

f = {(0, 0, 0): 2, (0, 0, 1): 1, (0, 1, 0): 2, (0, 1, 1): 0, (1, 0, 0): 1, (1, 0, 1): 1, (1, 1, 0): 2, (1, 1, 1): 0, (2, 0, 0): 1, (2, 0, 1): 1, (2, 1, 0): 2, (2, 1, 1): 0}.

ВИСНОВКИ

У цій роботі було досліджено оптимальне керування системами масового обслуговування з дискретним часом та з обмеженою чергою.

В теоретичній частині було розглянуто основні поняття, пов’язані з системами масового обслуговування. Було детально описано тип системи, над якою проводиться дослідження, визначено множини станів та рішень, а також сформульоване керування системою. Також було визначено рівняння оптимальності та функцію Беллмана для мінімізації середніх витрат.

У практичній частині було реалізовано алгоритм для знаходження оптимальної стратегії керування системою. Алгоритм було реалізовано за допомогою мови Python з використанням додаткових бібліотек, які дозволяють спростити математичні обрахунки. Програма дозволяє вводити бажані параметри для задання системи масового обслуговування. На вихід програма подає оптимальну стратегію для заданої системи, де кожному стану системи у відповідність ставиться рішення, та відповідне значення середніх витрат системи. Також у цій частині показано приклад роботи алгоритму для конкретної системи з заданими параметрами.

В подальшому можна оптимізувати алгоритм, знайшовши альтернативні способи розв’язання рівнянь, які потребуватимуть менше ресурсів, а також розширити алгоритм для роботи з різними типами систем масового обслуговування, зокрема враховувати дисципліну черги або інші розподіли вхідного потоку та часу обслуговування. Основною ж задачею буде розробка алгоритму для роботи з системами масового обслуговування з великою місткістю.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Gregory F. Lawler. Introduction to Stochastic Processes. Second edition. Boca Raton/London/New York : Chapman & Hall/CRC, 2006.

[2] Hamdy A. Taha. Operations Research: An Introduction / University of Arkansas, Fayetteville. Pearson, 10th edition, 2016.

[3] Hugo Gimbert. Pure stationary optimal strategies in Markov decision processes / LIAFA, Université Paris 7, France. Paris, 2007.

[4] Kitaev M., Rykov V. Controlled Queueing Systems. Boca Raton/New York/London/Tokyo : CRC Press, 1995.

[5] Lodewijk Kallenberg. Markov decision processes / University of Leiden. Leiden, 2014.

[6] Ronald A. Howard. Dynamic Programming and Markov Processes. Massachusetts : The Massachusetts Institute of Technology, 1960.

[7] Ruslan K. Chornei, Hans Daduna, Pavel S. Knopov. Control of spatially structured random processes and random fields with applications. Boston/Dordrecht/London : Springer, 2006.

[8] Rykov V., Efrosinin D. Optimal Control of Queueing Systems with Heterogenous Servers. Amsterdam : Kluwer Academic Publishers, 2004.

ДОДАТКИ

Програмний код

```
import warnings

warnings.filterwarnings('ignore', 'The iteration is not making good progress')
from itertools import product
import numpy as np
from scipy.optimize import fsolve

system = {}
accuracy = 1
states = []
policy = {}
V = {}
max_iter = 10


def create_system(N, K, lam, mu, c):
    gamma = [c[i] / mu[i] for i in range(len(mu))]
    gmc = list(zip(gamma, mu, c))
    gmc.sort()

    gamma_sorted = [g for g, m, c in gmc]
    mu_sorted = [m for g, m, c in gmc]
    c_sorted = [c for g, m, c in gmc]

    system['N'] = N
    system['K'] = K
    system['lambda'] = lam
    system['mu'] = mu_sorted
    system['c'] = c_sorted
    system['gamma'] = gamma_sorted

    system['states'] = get_states()

    for i in range(len(states)):
        c_x = system['c'][0] * states[i][0]
        for j in range(states[i][1], len(states[i]) - 1):
            ccc = system['c'][j]
            ccc *= states[i][j]
```

```
            c_x += ccc
        V[tuple(states[i])] = c_x

    for i in range(len(states)):
        if 0 not in states[i][1:]:
            policy[tuple(states[i])] = 0
        for j in range(1, len(states[i])):
            if states[i][j] == 0:
                policy[tuple(states[i])] = j
                break
    return


def get_states():
    states_list = []
    for q in range(system['N'] - system['K'] + 1):
        server_states = list(product([0, 1], repeat=system['K']))
        server_states = [(q,) + combo for combo in server_states]
        states_list.extend(server_states)
    for t in states_list:
        states.append(list(t))
    return states


def policy_evaluation():
    g_out = 1
    for m in range(max_iter):
        for x in states:
            def equations(variables):
                (a, b) = variables
                e1 = a - (c(x) + system['lambda'] * third_coef(x) + M0(x) * a +
first_coef(x) + second_coef(x) - b) / (
                        system['lambda'] + sum(system['mu']))
                e2 = a - (c(x) + system['lambda'] * T0(x) + Tk(x) - b) / (system['lambda']
+ sum(system['mu']))
                return [e1, e2]

            output = fsolve(equations, np.array([V[tuple(x)], g_out]))
            v_out = output[0]
            g_out = output[1]
            V[tuple(x)] = v_out
            g_new = subalgorithm(v_out, g_out)

        if not policy_improvement():
```

```
            return g_new, policy
    return


def subalgorithm(v, g):
    for x in states:
        difference = 0

        for m in range(max_iter):
            gm = (system['lambda'] * V[tuple(states[0])] - g) / (system['lambda'] +
sum(system['mu']))

            vm = (c(x) + system['lambda'] * third_coef(x) + M0(x) * v + first_coef(x) +
second_coef(x) - g) / (
                    system['lambda'] + sum(system['mu']))

            difference_v = abs(vm - v)
            difference_g = abs(gm - g)

            difference = max(difference, difference_v, difference_g)
            if difference < accuracy:
                break
            else:
                v = vm
                g = gm
                V[tuple(x)] = v
    return g


def policy_improvement():
    policy_improved = False
    for x in states:
        v_curr = V[tuple(x)]
        for j in range(1, len(x)):
            if x[j] == 0:
                if policy[tuple(x)] == j:
                    continue
                e = x.copy()
                e[j] += 1
                v_new = V[tuple(e)]
                if v_new < v_curr:
                    policy[tuple(x)] = j
                    policy_improved = True
                    v_curr = v_new
```

```
    return policy_improved


def T0(x):
    fourth = 1000000
    if x[0] < system['N'] - system['K']:
        e = x.copy()
        e[0] += 1
        e = [0 if x < 0 else x for x in e]
        fourth = V[tuple(e)]
    for i in range(1, system['K']):
        e = x.copy()
        e[i] += 1
        e = [0 if x < 0 else x for x in e]
        if fourth > V[tuple([1 if x > 0 else x for x in e])]:
            fourth = V[tuple([1 if x > 0 else x for x in e])]
    return fourth


def Tk(x):
    T_k = 0
    for i in range(system['K']):
        if x[0] > 0:
            e = x.copy()
            e[0] -= 1
            e[i] -= 1
            fifth = T0(e)
        if x[0] == 0:
            e = x.copy()
            e[i] -= 1
            fifth = V[tuple([0 if x < 0 else x for x in e])]
        else:
            fifth = V[tuple(x)]
        T_k += fifth * system['mu'][i]
    return T_k


def M0(x):
    m0 = 0
    for i in range(x[1], len(x) - 1):
        if x[i] == 1:
            m0 += system['mu'][i]
    return m0
```

```
def c(x):
    c_x = system['c'][0] * x[0]
    for i in range(x[1], len(x) - 1):
        c_x += system['c'][i] * x[i]
    return c_x


def first_coef(x):
    first = 0
    if x[0] == 0:
        for i in range(x[1], len(x) - 1):
            if x[i] == 1:
                e = x.copy()
                e[i] = 0
                first += system['mu'][i] * V[tuple(e)]
    return first


def second_coef(x):
    second = 0
    if x[0] > 0:
        for i in range(x[1], len(x) - 1):
            if x[i] == 1:
                e = x.copy()
                e[i] = 0
                e[0] -= 1
                e = [0 if x < 0 else x for x in e]
                ef = policy[tuple(e)]
                e[ef] += 1
                second += system['mu'][i] * V[tuple(e)]
    return second


def third_coef(x):
    e = x.copy()
    ef = policy[tuple(e)]
    e[ef] += 1
    e = [1 if x > 1 else x for x in e]
    third = V[tuple(e)]
    return third


def main():
```

```
    n = int(input("Let's create a system.\nEnter the number of full system capacity
(servers + places in queue): "))
    while n <= 0:
        n = int(
            input("The capacity of system can only be a positive integer.\nEnter the full
capacity of the system: "))

    k = int(input("Enter the number of servers: "))
    while k > n or k < 1:
        k = int(input("Number of servers cannot be bigger than the full capacity of the
system or lower than 1."
                      "\nEnter the number of servers: "))

    lmbd = int(input("Enter lambda: "))

    mu_list = list(map(int, input("Enter the servers' intensities in this format: a b c...:
").strip().split()))[:n]
    while len(mu_list) != k:
        mu_list = list(map(int, input(
            "The length of list must be equal to number of servers.\nEnter the numbers
in this format: a b c...: ").strip().split()))[
                  :n]

    c_list = list(map(int, input("Enter the servers' job costs in this format: a b c...:
").strip().split()))[:n]
    while len(c_list) != k:
        c_list = list(map(int, input(
            "The length of list must be equal to number of servers.\nEnter the servers'
job costs in this format: a b c...: ").strip().split()))[
                 :n]

    create_system(n, k, lmbd, mu_list, c_list)  # 4, 2, 3, [6, 11], [3, 3]
    g, p = policy_evaluation()
    print('System processing cost:', g)
    print('Optimal strategy:', p)


main()
```