Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: «**РІВНОВАГА ЗА НЕШЕМ У СТОХАСТИЧНИХ КОАЛІЦІЙНИХ ІГРАХ НАКОПИЧЕННЯ КАПІТАЛУ**»

Виконала: студентка 4-го року навчання,

Освітньої програми «Прикладна математика», 113

Сечкар Уляна Сергіївна

Керівник Чорней Р. К.

_____

(прізвище та ініціали)

Кваліфікаційна робота захищена з оцінкою

_____

Секретар ЕК

_____

«____» _____

20____ р.

Зміст

Анотація 3
ВСТУП 4
ТЕОРІЯ 7
Визначення моделі гри для степеневої функції корисності 7
Умови Існування рівноваги між коаліціями 13
Визначення моделі гри для логарифмічної функції корисності 16
Практична частина 21
Приклад 1.1 21
Приклад 1.2 26
Висновки для алгоритму знаходження SMPE для степеневої функції 28
Приклад 2 30
Висновки та пропозиції для подальшого покращення моделі 38
ВИСНОВОК 41
Джерела 43
Додатки 44
Додаток 1 44
Додаток 2 49

Анотація

В роботі розглядаються стохастичні коаліційні ігри з накопиченням ресурсу, умови та наявність рівноваги між учасниками та коаліційними формуваннями. В теоретичній частині роботи вводяться загальні означення та теоретичне підґрунтя умов існування рівноваги. Наведено дві моделі з різними функціями корисності та визначені методи знаходження виграшу для агентів гри. В практичній частині наводиться приклад розв’язування задач за допомогою визначеного алгоритму та описана його реалізація методами мов програмування Python та C#.

ВСТУП

Сучасна економіка характеризується постійними змінами на нестабільністю, що породжує проблему управління ресурсами та прийняття оптимальних стратегій в умовах невизначеності та стохастичності. Спрощення цих процесів до моделі, що характеризується набором змінних приводить нас до коаліційної гри накопичення капіталу. Такі моделі є дуже важливими для розуміння взаємодії між учасниками та розробки оптимальних стратегій.

Метою цієї дипломної роботи є аналіз стохастичних коаліційних ігор накопичення ресурсу на наявність рівноваги між учасниками та формуваннями учасників, визначення технічних особливостей таких ігор та визначення їх практичного застосування. Робота також передбачає дослідження Нешівської рівноваги в симетричній грі видобутку ресурсів для різних функції корисності гравців та врахування стратегій розбиттів коаліцій для визначення оптимального способу кооперування. Саме поняття рівноваги за Нешем, яке базується на концепції оптимальної відповіді кожного гравця на дії інших, стає ключовим в умовах невизначеності та стохастичності. Адаптація цього поняття до контексту стохастичних ігор дозволяє нам уявити ситуації, де рішення не є чітко визначеними, а замість цього гравці мають враховувати ймовірність різних подій та їх вплив на кінцевий результат.

Об’єктом дослідження є моделі стохастичних коаліційних ігор, зокрема ігор видобутку ресурсів з ненульовою сумою. Дослідження в цьому контексті спрямоване на розкриття особливостей та розуміння аспектів взаємодії учасників таких ігор.

Для досягнення поставлених завдань використовуються методи теорії ігор, математичного аналізу, економічного моделювання, динамічного програмування та статистики . Аналізуються та формалізуються стратегії учасників гри, вивчається стохастичний закон переходу між станами, та розглядаються властивості Стаціонарної Марківської Ідеальної Рівноваги, яку рівновага за Нешем, яку складають Марківські стратегії. Моделі гри досліджуються у симетричному форматі з метою спрощення моделі для реалізації.
Робота складається з двох частин. У кожній розглядається певна модель гри з різними модифікаціями, для кожної з моделей представлений приклад, на основі результатів якого зроблено висновки щодо найефективнішої поведінки гравців для різних початкових умов.

Було реалізовано моделі з різними функціями корисності в середовищі розробки з допомогою мови програмування Python та C#, що дозволило впровадити теоретичні результати роботи в практичних застосуваннях. Реалізація дає змогу визначати оптимальні стратегії та аналізувати динаміки поведінки гравців для різних функцій корисності, що може слугувати основою для подальших досліджень та застосувань в галузі управління ресурсами та стратегічного планування в економіці.

Мета дослідження: Метою дослідження є розгляд теорії стохастичних коаліційних ігор накопичення капіталу, вивчення математичних моделей, основних підходів до побудови таких моделей, а також розгляд та реалізація ефективних алгоритмів для максимізації прибутку як для окремих гравців, так і для коаліцій в цій грі.

Завдання дослідження:

1. Розглянути теорію стохастичних коаліційних ігор накопичення капіталу та математичні моделі створені для відтворення таких теоретичних та практичних ситуацій.
2. Вивчити основні підходи до побудови моделей стохастичних коаліційних ігор накопичення капіталу. Це включає у себе аналіз різних моделей, їхніх переваг та обмежень, а також пошук найбільш ефективних та адаптованих до конкретних ситуацій підходів.
3. Вивчити ефективні алгоритми для максимізації прибутку як для окремих гравців так і для загального прибутку коаліцій.
4. Провести моделювання розробленого алгоритму, застосувати алгоритм на прикладній задачі. Це дозволить перевірити ефективність та придатність розроблених підходів у практичних умовах та з'ясувати їхню придатність для подальшого застосування.

Методи дослідження:

- аналіз літературних джерел і публікацій з даної теми;
- математичний аналіз та моделювання стохастичних коаліційних ігор накопичення капіталу;
- застосування методів динамічного програмування для аналізу та оптимізації кооперативних ігор;
- використання програмних засобів для моделювання та аналізу .

ТЕОРІЯ

Визначення моделі гри для степеневої функції корисності

Симетрична гра – гра в якій кожному учаснику доступний однаковий набір стратегій та виграш залежить від наступних стратегій, а не від учасників при відтворенні деяких дій. Це означає, що всі гравці мають однакові можливості відповіді на поточному стані та впливу на кінцевий результат.

Коаліційна стохастична гра накопичення ресурсу є підкласом динамічних ігор, що відбувається в дискретні моменти часу. В момент часу t всі учасники незалежно приймають рішення. Така гра може тривати обмежену кількість часу (гра видобутку ресурсів з скінченним горизонтом) або мати неперервний час (гра видобутку ресурсів з нескінченним горизонтом). Перший тип гри буде вважатись завершеним після n-ої напередодні обумовленої кількості кроків, однакової для всіх учасників гри. Гра з нескінченним горизонтом буде тривати до моменту виснаженню капіталу.

В грі з нескінченним горизонтом кінцевий результат залежить від того, як ефективно кожен гравець розпоряджається ресурсами протягом гри. Такий різновид ігор відображає більш реалістичні умови, де ресурси можуть бути нескінченно доступними, але гравцям потрібно збалансувати свої рішення та стратегії, щоб досягти максимально можливого прибутку.

Модель гри I

- Нехай m – кількість гравців у грі ($m \geq 2$, $m \in N$). Кожен учасник має порядковий номер i від 1 до m.

- T – кількість всіх етапів гри.

- Стан гри s_t – кількість капіталу в момент часу гри $t \in T$. Множина всього доступного капіталу під час гри позначається $S = [0; +\infty)$.

- $A(s_t) = [0; s_t]$ – простір рішень доступних гравцеві на кожному з етапів гри $s_t \in S$, простір сукупних рішень визначається як

$$D(s_t) \coloneqq \{(x_1, \dots x_m) \in A(s_t) \times \dots \times A(s_t) : \sum_{i=1}^{m} x_i \leq s_t$$

- Функція миттєвої корисності гравців - $u : S \rightarrow [0; +\infty)$. Функція корисності є неперервною та зростаючою.

$$u(x) = cx^{\alpha},$$

Де $c \in (0; +\infty)$, $\alpha \in (0; 1)$.

- Закон переходу між станами $s_{t+1} = M(s_t, a_t, \varepsilon_t)$, де M() – неперервна функція з областю значень S, s_t - стан гри в момент часу $t \in T$, a_t - вектор рішень гравців $s_t \in D(s_t)$, ε_t - член випадкового збурення, ε_t належить певному простору Q. Послідовність станів визначається за законом

$$s_{t+1} = M(s_t, a_t, \varepsilon_t) = \left(s_t - \sum_{i=1}^{m} a_{ti}\right) \times \varepsilon_t$$

- $B \in (0, 1)$ – дисконтований множник спільний для всіх гравців, для якого виконується $\beta E[\varepsilon^{\alpha}] < 1$, де E – оператор математичного сподівання

відносно розподілу ймовірності μ. Для спрощення представимо вираз $\beta E[\varepsilon^{\alpha}]$ як $l := \beta E[\varepsilon^{\alpha}]$.

- Загальний виграш гравця i ∈ {1, … , m}:

$$\sum_{t \in T} \beta^{t-1} u(a_{ti}),$$

Де a_{ti} – i-та координата вектора a_t.

Визначимо декілька означень, якими ми будемо далі оперувати при визначенні рівноваги між коаліціями. [1]

Означення 1.1. Стратегією i-ого гравця називають послідовність $\{\pi_{it}\}_{t \in T}$, в якому π_{it} є вимірним за Борелем відображенням множини всіх можливих історій гри на кроці t в множину доступних ресурсів S. Профіль вважається сумісним якщо для всіх можливих етапів та історій гри виконується умова

$$\left(\pi_{1t}(h_t), \dots \pi_{mt}(h_{tm})\right) \in D(s_t)$$

Де h_t – історія гравця.

Позначимо $\Pi := \Pi_i \times \dots \times \Pi_m$ – простір всіх можливих сумісних стратегій гравців.

Нехай F – множина всіх борелевих функцій таких що $f: S \to S, f(s) \in A(s)$ для всіх $s \in S$.

Означення 1.2. Марківською стратегією називається послідовність $(f_1, f_2 \ldots), f_t \in F$ – функція вибору гравця на кроці t. Марківська стратегія у якої всі функції вибору рівні між собою називається стаціонарною.

На кожному кроці гравця його вибір залежить від поточного стану s як реалізація функції вибору $f(s)$, та не визначається попередньою історією гри.

Для кожного початкового стану $s_1 = s \in S$ та профілю стратегій $\pi \in \Pi$ визначається міра ймовірності P_s^π та стохастичний процес $\{S_t, X_t\}$ на просторі усіх можливих історій H, де величини S_t, і X_t характеризують стан та вектор рішень гравців в момент часу t ∈ T. Тоді для початкового стану $s \in S$ та профілю стратегій $\pi \in \Pi$ ми можемо визначити очікуваний виграш і-ого гравця як

$$\gamma_{\mathrm{i}}(\pi)(s) = E_s^\pi \left[\sum_{t \in T} \beta^{t-1} u(X_{ti}) \right],$$

Де X_{ti} – і-та координата вектора рішень гравця X_t, а E_s^π – математичне сподівання відносно міри ймовірності P_s^π.

Означення 1.3. Рівновага Неша у стохастичній грі видобутку ресурсу з дисконтованим критерієм – профіль стратегій $\pi^* \in \Pi$ при якому жодні сумісні відхилення зі сторони інших учасників не є вигідними, тобто для всіх $s \in S$ виконується умова

$$\gamma_{\mathrm{i}}(\pi^*)(s) \geq \gamma_{\mathrm{i}}(\pi_{\mathrm{i}}, \pi^*_{-\mathrm{i}})(s)$$

Означення 1.4. Якщо виконується рівновага Неша $\pi^* = (\pi_1^*, \pi_2^*, \dots \pi_m^*)$ та $\pi_j^* = \pi_{j+k}^*$ для всіх $k \in [m]$ то рівновага називається симетричною.

Означення 1.5. Якщо в стохастичній грі Марківські стратегії π_i^* утворюють Нешівську рівновагу $\pi^* = (\pi_1^*, \pi_2^*, \dots \pi_m^*)$, то π^* називають Марківською ідеальною рівновагою. Якщо T = N та π_i^* - стаціонарна стратегія, тоді π^* називають Стаціонарною Марківською Ідеальною Рівновагою (SMPE).

Означення 1.6 Профіль стратегій $\pi \in \Pi$ вважається стійким відносно кооперативного відхилення коаліції
якщо для всіх $s \in S$ та для довільного профілю стратегій σ коаліції K, такого що $(\sigma, \pi_{-k}) \in \Pi$ виконується

$$\sum_{i \in K} \gamma_i(\pi)(s) \geq \sum_{i \in K} \gamma_i(\sigma, \pi_{-K})(s)$$

Позначимо сукупність $C = \{C_1, \dots C_n\}$ розбиттям множини [m] на n коаліцій, де $C_i \neq \emptyset$ та $C_i \cap C_j \neq \emptyset$ для всіх $i, j \in [n]$ та $\bigcup_{i \in [n]} C_i = [m]$.

Означення 1.7. Профіль стаціонарних стратегій $\pi \in \Pi$ є стаціонарною рівновагою між коаліціями, якщо він є стійким відносно кооперативного відхилення для всіх $C_i \in C$ при $i \in [n]$.

Для визначення рівноваги Неша нам достатньо максимізувати прибутки гравців. Для цього $\forall s \in S$

$$f^{(1)}s := \underset{x \in A(s)}{\operatorname{argmax}}\ u(x) = \underset{x \in A(s)}{\operatorname{argmax}}\ (cx^{\alpha}) = s/m$$

$$v^{(1)}s := \max_{x \in A(s)}\ u(x) = \max_{x \in A(s)}\ (cx^{\alpha}) = \frac{cs^{\alpha}}{m^{\alpha}}$$

Де $f^{(1)}s$ – рішення гравців, $v^{(1)}s$ – виграш гравців на t = N кроці гри. Для знаходження рівноваги на кожному кроці гри ми використовуватимемо результати підгри, отримані на наступному кроці (в момент часу t+1).
Для довільного виграшу і-того гравця, що визначається функцією $v \in B_0(S)$, яка залежить від початкового стану підгри, ми можемо розглянути одноетапну симетричну гру $\Gamma(v)$ з незмінним станом гри $s \in S$. Тоді для простору рішень [0; s/m] та сукупності рішень $\bar{x} = (x_1, x_2, \dots x_m) \in D(s)$ виграш і-того гравця визначається як

$$\omega_i(\bar{x}) := cx_i^{\alpha} + \beta E\left[v((s - \sum_{j=1}^{m} x_j)\varepsilon)\right]$$

Де $c \in (0; +\infty)$ та $\alpha \in (0; 1)$ – визначені сталі, ε – випадкова величина з розподілом ймовірности μ.

Лема 1.1. Для $\forall\ v \in V$ у грі $\Gamma(v)$ існує симетрична рівновага на Нешем.
Оскільки $v(s)$ – функція що залежить від поточного стану гри, всі виграші гравці належать простору V.
З Леми випливає що існують сукупності рішень $\bar{x} = (x_1, x_2, \dots x_m) \in D(s)$ що задовільнять такі рівності для всіх гравців

$$\omega_i(\bar{x}') \coloneqq \max_{x_i \in A(s)} \omega_i(x_i, \bar{x}'_{-i}) \ (*)$$

Для кожного фіксованого стану $s \in S$ ми визначимо профіль стратегій $\bar{f} = (f_1, f_2 \dots f_m)$ у якому для ∀ i $\in [m]$ виконується

$$f_i(s) \coloneqq \frac{s}{m + \lambda}$$

де $\lambda \coloneqq (kl/c)^{\frac{1}{1-\alpha}}$, k – довільна додатна константа.

Даний профіль стратегій є симетричною Нешевою рівновагою у грі $\Gamma(v)$. Тоді при оцінці виграшу гравця при використанні профілю стратегій $\bar{f}$ для $\forall s \in S$ та ∀ i $\in [m]$ ми отримаємо

$$\omega_i(f_1, \dots f_m) = c\big(f_i(s)\big)^{\alpha} + kl(s - \sum_{j=1}^{m} f_j(s))^{\alpha} = c(\frac{s}{m + \lambda})^{\alpha} + kl(s - \frac{ms}{m + \lambda})^{\alpha}$$

$$= c\left(\frac{s}{m + \lambda}\right)^{\alpha}\left(1 + \frac{kl}{c}\lambda^{\alpha}\right) = c\left(\frac{s}{m + \lambda}\right)^{\alpha}(1 + \lambda) \ (1.1)$$

Оскільки не існує відмінного від $\bar{x}'$ вектора що задовольняв би умови задачі для всіх i $\in [m]$, рівновага Неша для нашого профілю стратегій є єдиною в грі.

Умови Існування рівноваги між коаліціями

Теорема 2.1 У стохастичній грі накопичення капіталу з числом учасників m та нескінченним часом для розбиття $C = \{C_1, \dots C_n\}$ існує стаціонарна рівновага між коаліціями для моделі гри I. [1]

Твердження 2.1 Оскільки дисконтовані виграші учасників кількість яких дорівнює числу m є збіжними для довільного початкового стану $s \in S$ та $\pi \in \Pi$ та $\forall\, v \in V$ існує розв'язок що задовольняє рівняння

$$v_i(s) = \sup_{\pi_i \in \Pi_s} \gamma_i(\pi_i, \bar{f}^*_{-i})(s)$$

для всіх $s \in S$ та $\pi \in \Pi$, де визначена міра ймовірності для стохастичної гри накопичення капіталу для моделі гри I, ми отримаємо що
m-агентна гра накопичення капіталу з нескінченним часом, що задовольняє умови моделі I має єдину симетричну Стаціонарну Марківську Ідеальну Рівновагу (SMPE).
При застосуванні гравцями профілю стратегій $\pi^k \in \Pi^C$ сумарний очікуваний виграш для кожної з коаліцій для всіх $k \in [n]$, матиме вигляд

$$\sum_{i \in C_k} \gamma_i(\pi)(s) = m_k E_s^{\pi}\left[\sum_{t=1}^{\infty} \beta^{t-1} u\left(\frac{1}{m_k} \sum_{j \in C_k} X_{tj}\right)\right]$$

$$= E_s^{\pi}\left[\sum_{t=1}^{\infty} \beta^{t-1} {m_k}^{1-\alpha} c\left(\sum_{j \in C_k} X_{tj}\right)^{\alpha}\right] \quad (2.1)$$

За твердженням 2.1 для n-агентної гри накопичення капіталу з нескінченним часом, що задовольняє умови моделі I з n-агентним стаціонарним профілем стратегій $\sigma^* = (\sigma_1^*, \ldots\, \sigma_n^*)$ визначимо профіль стратегій $\pi^* = (\pi_1^*, \ldots\, \pi_n^*) \in \Pi^C$, з використанням нерівності

$$\pi_i^*(s) = \frac{\sigma_k^*(s)}{m_k} \quad (2.2)$$

Для всіх $k \in [n], i \in C_k, s \in S$.

Тоді для всіх $k \in [n]$ та початкового стану , $s \in S$ справджується нерівність

$$\sum_{i \in C_k} \gamma_i(\pi^*)(s) \geq \sum_{i \in C_k} \gamma_i(\pi)(s)$$

Внаслідок чого π^* є стаціонарною рівновагою між коаліціями в грі накопичення ресурсу з нескінченним горизонтом для розбиття на коаліції C.

Визначення моделі гри для логарифмічної функції корисності

Моделювання видобутку загальних відновлюваних морських ресурсів в сучасному світі є складним завданням, особливо коли ця промисловість є вирішальною для країн, що займаються рибальством. У такому контексті поняття «трагедії спільних ресурсів», набуває особливого значення. Це може призвести до виникнення так званих "рибних воєн", подібних до Кодової війни між Ісландією та Великою Британією, що послужила мотивацією для провідної статті Левгарі і Мірмана. Це була перша модель рибних воєн, яка використовувала інструменти динамічних ігор і призначалася для опису ситуації видобутку загальних ресурсів.

Уявімо собі, що деякі країни використовують загальні морські ресурси для забезпечення життєвих потреб своїх громадян. Однак вичерпання цих ресурсів може призвести до катастрофічних наслідків, включаючи навіть вимирання видів, на яких ґрунтується існування цих громад. Для цього були введені спеціальні логарифмічні функції винагороди, які допомагають краще розуміти динаміку експлуатації.

Цей розділ досліджує не лише математичні моделі видобутку ресурсів, а й розвиток числових методів для аналізу таких моделей та їх практичне застосування.

Для визначення моделі стохастичної коаліційної гри для логарифмічної функції корисності ми трохи змінимо модель визначену для степеневої функції в першому розділі.

Модель гри II

- Нехай n – кількість гравців у грі ($n \geq 2$, $n \in N$). Кожен учасник має порядковий номер i від 1 до n.

- T – кількість всіх етапів гри.

- Стан гри x – кількість капіталу в момент часу гри $t \in T$. Множина всіх можливих значень x = [0;1]. В стані x не можна використати/отримати більше x/n одиниць ресурсу.

Кожний учасник зацікавлений в довгостроковому ефекті від використання ресурсу, причому кожному учаснику потрібно враховувати ходи інших гравців при прийнятті рішення в стані x. Ці аспекти враховуються застосування методів динамічного програмування та концепції Нешівської рівноваги в знаходженні розв'язків моделі.

- Функція миттєвої корисності гравців - u для витрат ресурсу c_i

$$u(c_i) = ln(c_i)$$

(для $\ln(0) = \infty$).

- Термінований виграш гравця після переривання гри дорівнює $\frac{\ln(x)}{n}$.

- $B \in (0, 1)$ – дисконтований множник спільний для всіх гравців, для якого виконується $\beta E[\varepsilon^{\alpha}] < 1$, де E – оператор математичного сподівання відносно розподілу ймовірності μ.

- Закон переходу між станами X, що залежить від вибору стратегії c визначається рівнянням

$$X_{t+1} = \left(X_t - \sum_{i=1}^{n} c_{ti}\right)^{\alpha}$$

- Загальний виграш гравця i ∈ {1, … , n}:

$$\Pi_i(c) = \sum_{t=1}^{T} \beta^{t-1} \ln\big(c_i(t)\big) + \beta^T \ln\left(\frac{X(T+1)}{n}\right)$$

де $c_i(t)$ – витрата ресурсу в момент часу t.

Для логарифмічної функції корисності ми будемо шукати рівновагу Неша в стратегіях закритого циклу. Для цього використовується умова оптимальності Беллмана яка визначена для умов скінченого горизонту та обумовлених виграшів. Рівняння Беллмана використовує допоміжну функцію V : $\{1, \dots, T+1\} \times [0,1] \to R \cup \{-\infty\}$ де $V\ (\bar{t}, \bar{x})$ – оптимальний виграш в момент початку гри $\bar{t}$ в стані $\bar{x}$.

Знаходження соціально-оптимального профілю стратегій полягає в знаходженні розв’язку рекурентного рівняння [2]

$$V(T+1, x) = n \ln\left(\frac{x}{n}\right) \quad (3.0)$$

$$V(t, x) = \sup_{c_1, \dots, c_n \in [0, \frac{x}{n}]} \sum_{i=1}^{n} \ln c_i + \beta V\left(t+1, \left(x - \sum_{i=1}^{n} c_i\right)^{\alpha}\right) \quad (3.1)$$

Оптимальним вважається профіль, що для всіх t задовольняє умову

$$c(t,x) \in \underset{c_1,\ldots,c_n \in [0,\frac{x}{n}]}{\operatorname{argmax}} \sum_{i=1}^{n} \ln c_i + \beta V\left(t+1, \left(x - \sum_{i=1}^{n} c_i\right)^{\alpha}\right) (3.2)$$

В статті [2] соціальний оптимум визначений як кооперативне рішення, що відноситься до класу оптимальних за Парето профілів. Для змінних виплат гравців ми зосередимося лише на таких профілях c, що максимізують суму дисконтованих виплат $\sum_{i=1}^{n} \Pi_i(c)$.

За означенням 1.4 оскільки наші профілі симетричні, ми можемо перетворити нашу оптимізацію на одновимірну. Тоді рівняння (3.1) та (3.2) матимуть вигляд [2]

$$V(t,x) = \sup_{c \in [0,\frac{x}{n}]} n \ln c + \beta V(t+1, (x-nc)^{\alpha}) \ (3.3)$$

$$c_i(t,x) = c(t,x) \in \underset{c \in [0,\frac{x}{n}]}{\operatorname{argmax}} n \ln c + \beta V(t+1, (x-nc)^{\alpha}) \ (3.4)$$

Для знаходження рівноваги $\bar{c}$ ми розглянемо оптимізаційну проблему і-ого гравця, стратегія якого полягає в найвигідніших рішеннях-відповідях на стратегії інших гравців. Профіль рівноваги це зафіксована точка функції складеної з отриманих векторів рішень всіх гравців.
Оскільки найоптимальніше рішення буде однаковим для всіх $\bar{c}_{\sim i}$, ми можемо скоротити його до суми рішень o.

Для суми стратегій інших гравців $o(t,x)$ функція виграшу визначена рекурентно [2]

$$V_i(T+1,x) = \ln(\frac{x}{n}) (4.0)$$

$$V_i(t,x) = \sup_{c_i \in [0,\frac{x}{n}]} \ln c_i + \beta V_i(t+1,\left(x - c_i - o(t,x)\right)^{\alpha}) (4.1)$$

Оптимальним вважається профіль, що для всіх t задовольняє умову

$$c_i(t,x) = \operatorname{argmax}_{c_i \in [0,\frac{x}{n}]} \ln c_i + \beta V_i(t+1,\left(x - c_i - o(t,x)\right)^{\alpha}) (4.2)$$

Метою є знайти фіксовану точку – профіль c, такий, що кожне c_i є оптимальним рішенням до $\sum_{j \neq i} c_i = o$.

В симетричних стратегіях ми шукати лише профіль c, такий що оптимальне рішення є ідентичним до $\bar{c}$, яке вибране іншими гравцями. Тоді рівняння матиме вигляд

$$V(t,x) = \sup_{c \in [0,\frac{x}{n}]} \ln c + \beta V(t+1,(x - c - (n-1)\bar{c})^{\alpha}) \ (4.3)$$

Оптимальним вважається профіль, що для всіх t задовольняє умову

$$c_i(t,x) = c(t,x) \in \operatorname{argmax}_{c \in \left[0,\frac{x}{n}\right]} \ln c + \beta V(t+1,(x - c - (n-1)\bar{c})^{\alpha}) \ (4.4)$$

ПРАКТИЧНА ЧАСТИНА

Приклад 1.1

Згідно з теоремою 2.1 та твердженням 2.1 m-агентна коаліційна гра накопичення ресурсу, що задовольняє умови моделі I будується за допомогою Стаціонарної Марківської Ідеальної Рівноваги n-агентної гри видобутку ресурсу з кількістю коаліцій n.

Нехай $\sigma^* = (\sigma_1^*, \dots \sigma_n^*)$ – положення стаціонарної рівноваги де σ_i^* відповідає стратегії i-го учасника, степенева функція $k^* s^\alpha$ – очікуваний дисконтований виграш n-ого гравця з початкового стану $s \in S$. Наша мета знайти рівновагу між коаліціями $\pi^* = (\pi_1^*, \dots \pi_n^*)$, де π_i^* відповідає стратегії i-тої коаліції, гри накопичення ресурсу для розбиття C на n коаліцій. Ця рівновага знаходиться за допомогою нерівності (2.2).

Використовуючи рівняння 2.1, ми можемо записати сумарний очікуваний дисконтований виграш гравців кожної коаліції для кожного $i \in [n]$ та $s \in S$, при використані рівноваги $\pi^* = (\pi_1^*, \dots \pi_n^*)$

$$m_i^{1-\alpha} k^* s^\alpha$$

та очікуваний дисконтований виграш кожного учасника коаліції визначається як:

$$\frac{k^*}{m_i^\alpha} s^\alpha$$

Розглянемо стохастичну гру накопичення ресурсу з нескінченним часом, у якій виконуються умови моделі I.

Кількість гравців m = 4, c = 2, α = 0.45, β = 0.9, μ = 0.22, σ = 0.0004 – параметри логнормального розподілу випадкової величини ξ: $\xi \sim Log(0.22, 0.0004)$.
Для 4 гравців у нас буде 5 унікальних розбиттів на коаліції – кількість гравців у коаліції варіюватиметься від одного до 4, кількість коаліцій варіюватиметься також від одного до 4. Щоб знайти групування гравців ми створимо функцію, яка для кількості гравців m повертатиме список унікальних коаліційних комбінацій.
Позначимо h – кількість унікальних розбиттів на коаліції для кількості учасників m. Тоді для кожного $i \in [h]$ ми будемо застосовувати алгоритм побудови Стандартної Марківської Ідеальної рівноваги для кількості коаліцій в унікальному розбитті i, оскільки міжкоаліційна рівновага є рівновагою Неша для даної моделі.
Оскільки параметри логнормального розподілу для всіх гравців однакові, за формулою моментів для випадкової величини ми можемо знайти l:

$$l = \beta E[\varepsilon^{\alpha}] = \beta\left(e^{\mu\alpha+\frac{\sigma\alpha^2}{2}}\right) = 0.99369985, \qquad l \in (0;\ 1)$$

Для знаходження SMPE для кожної коаліції достатньо рекурсивно обчислити значення виграшу в ситуації рівноваги використовуючи формулу (1.1) та правила побудови послідовності функцій $\{v^{(t)}\}_{t\in N}$ для всіх $t \in N$ отримаємо [1]

$$k_1 = \frac{c}{m^{\alpha}}$$

$$k_{t+1} = \frac{c(1 + (k_t l/c)^{\frac{1}{1-\alpha}}}{(m + \left(\frac{k_t l}{c}\right)^{\frac{1}{1-\alpha}})^{\alpha}}$$

та

$$v^{(t)}(s) = k_t s^{\alpha}$$

$$f^{(1)}(s) = \frac{s}{m}$$

$$f^{(t+1)}(s) = \frac{s}{m + (\frac{k_t l}{c})^{\frac{1}{1-\alpha}}}$$

для всіх $s \in S$. Ми розглядаємо гру з нескінченним горизонтом. Близькими до $t \to \infty$ значеннями вважатимемо значення функцій $f^*(s)$ та $v^*(s)$ такі, що $\Delta f^*(s)$ та $\Delta v^*(s) \geq \delta$, де δ – надзвичайно мале, наперед визначене число. В функції для обчислення очікуваних дисконтованих виграшів встановлена максимальна кількість ітерацій для знаходження $f^{(t)} \to \infty$ для уникання входження в нескінченний цикл.

На першому кроці роботи алгоритму ми шукаємо значення l для заданих параметрів логнормального розподілу за допомогою визначеної функції з використанням бібліотек для математичних операцій. Значення повертаються в проміжку від 0 до 1, в іншому випадку функція повертатиме 0.
На наступному кроці ми звертаємося до функції яка визначає HashSet для заданої кількості гравців та конвертує його в список. Функція повертатиме унікальні комбінації дійсних коаліцій за допомогою допоміжної функції, яка імплементує сам функціонал розбиттів. Для кожного елементу отриманого списку розбиттів ми шукаємо очікувані дисконтовані виграші.

Якщо формується лише одна коаліція, тобто всі гравці діють узгоджено оптимальною поведінкою гравців є застосування єдиного соціально-оптимального профілю стратегій. Тому для пошуку відповідних виграшів агентів використовується формула для початкового стану $s_1 \in S$:

$$\tilde{v}^*(s_1) = \frac{c s_1^{\alpha}}{m^{\alpha}(1 - l^{\frac{1}{1-\alpha}})^{1-\alpha}}$$

Для кількості коаліцій $n > 1$ алгоритм пошуку очікуваних виграшів зводиться до знаходження Стаціонарної Марківської Ідеальної Рівноваги для n-агентної симетричної гри накопичення ресурсу, що і було відтворено в алгоритмі.

З використанням визначених функцій з Додатку 1 отримаємо такі результати для очікуваних дисконтованих виграшів у рівновазі між коаліціями, виведені в консоль:

```
[ 1 1 1 1 ](1.7711675, 1.7711675, 1.7711675, 1.7711675) *s1^a
[ 1 1 2 ](2.6716685, 2.6716685, 1.9557759, 1.9557759) *s1^a
[ 1 3 ](10.114488, 6.1693487, 6.1693487, 6.1693487) *s1^a
[ 2 2 ](7.404238, 7.404238, 7.404238, 7.404238) *s1^a
[ 4 ](12.539399, 12.539399, 12.539399, 12.539399) *s1^a
```

Рисунок 1 Очікувані дисконтовані виграші гравців для різних унікальних розбиттів h.

Де $[h_1 \dots h_n]$- унікальне групування, h – число учасників коаліції та n – число коаліцій. Наприклад [1 3] позначатиме, що утворилось 2 коаліції з одним та трьома учасниками.

Порівнюючи виграші першого розбиття та всіх наступних можемо побачити закономірність, що будь-яка участь у коаліції є взаємовигідною для всіх учасників на відміну від одноосібної гри.

Розглянемо гру, де дана закономірність справджується не для всіх коаліцій.

Приклад 1.2

Розглянемо стохастичну гру накопичення ресурсу з нескінченним часом, у якій виконуються умови моделі I.
Кількість гравців m = 4, c = 10, α = 0.6, β = 0.65, μ = 0.22, σ = 0.0004 – параметри логнормального розподілу випадкової величини ξ: $\xi \sim Log(0.22, 0.0004)$.

Оскільки кількість гравців в задачі співпадає з попередньою умовою задачі, процес знаходження коаліційних групуваннь буде аналогічним до попереднього прикладу. Для кожного $i \in [h]$, де h – кількість унікальних розбиттів, ми будемо застосовувати алгоритм побудови Стандартної Марківської Ідеальної рівноваги для кількості коаліцій в унікальному розбитті i.

Враховуючи, що параметри логнормального розподілу для всіх гравців однакові, за формулою моментів для випадкової величини ми можемо знайти l:

$$l = \beta E[\varepsilon^{\alpha}] = \beta\left(e^{\mu\alpha+\frac{\sigma\alpha^2}{2}}\right) = 0.74177384, \qquad l \in (0;\ 1)$$

Нехай початковий стан гри дорівнює $s_1 \in S$.

За алгоритмом знаходження Стаціонарної Марківської Ідеальної Рівноваги знайдемо значення очікуваних дисконтованих виграшів у міжкоаліційних рівновагах. Користуючись напередодні визначеними функціями, з Додатку 1 отримаємо такі результати

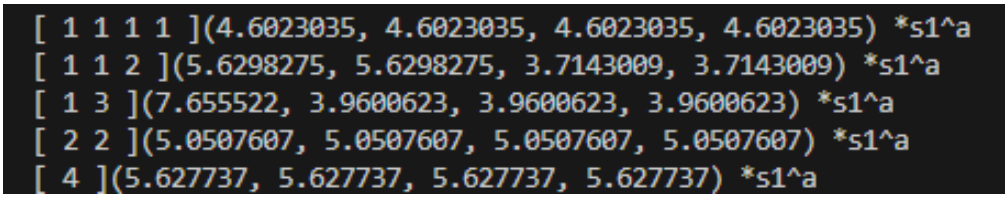

Рисунок 2 Очікувані дисконтовані виграші гравців для різних унікальних розбиттів h.

Оцінюючи значення очікуваних дисконтованих виграшів, можна помітити що певні розбиття не є вигідними для сторін, що вирішують вступати в коаліції. Також, якщо з якихось причин один чи два учасника відмовляються вступати в коаліцію, результати стверджують що залишатись в коаліції не є вигідним рішенням для поєднуваних сторін. Отже найкращою поведінкою гравців в такій ситуації є дзеркальна поведінка до рішень інших гравців. Належність Стаціонарної Марківської рівноваги до класу оптимальних за Парето стратегій надає рівновазі можливість існування положень, за яких виграші учасників є кращими при відмові від індивідуалістичної поведінки.

Найбільш вигідним рішенням для всіх сторін є об’єднання в одну велику коаліцію, що в свою чергу є більш сприятливим розбиттям ніж випадок, коли кожен із гравців діє самостійно. Винятком є лише ті об'єднання, які є вигідними для гравців незалежно від умов, наприклад, перехід до рівноваги між однаковими коаліціями або до соціально-оптимального профілю стратегій. Це правило завжди виконується в довільній грі накопичення ресурсу з кількістю учасників m, що задовольняє умови моделі I. [1, с.138]
Пояснення цьому полягає в тому, що утворення об'єднань між такими гравцями призводить до менших очікуваних виграшів у відповідній рівновазі між агентами, порівняно з використанням інших стратегій. Таким чином, мотивація гравців формувати різні коаліційні структури може суттєво варіюватися в залежності від значень параметрів моделі.

Підсумки для алгоритму знаходження SMPE для степеневої функції

Описана у прикладі гра є предметом дослідження в галузі теорії ігор та економічної теорії. Вона моделює ситуацію, в якій гравці об'єднуються в коаліції з метою максимізації своїх виграшів. Параметри гри визначаються за допомогою стохастичного процесу з логнормальним розподілом, що характеризується середнім значенням, дисперсією та іншими параметрами.

Головною метою дослідження є знаходження рівноваги між коаліціями та визначення оптимальних стратегій для гравців. Для цього використовуються алгоритм побудови Стаціонарної Марківської Ідеальної Рівноваги.

Алгоритм імплементує симетричну модель гри видобутку ресурсів, де гравці об'єднані в фіксовані коаліції. Початкові дані, такі як кількість агентів (m) та коефіцієнти, використовуються для обчислення різних параметрів і функцій, які визначають стратегії гравців та їх виграші.

Алгоритм використовує математичні моделі, такі як логнормальний розподіл та послідовності, для обчислення очікуваних виграшів імплементованих стратегій. Ці значення використовуються для оцінки рівноваг між різними коаліціями гравців.

Алгоритм також дозволяє обчислити соціально-оптимальні стратегії та виграші, які можуть бути досягнуті при єдиній коаліції всіх гравців. Це дозволяє порівняти виграші від різних варіантів коаліційних утворень та визначити оптимальні стратегії для кожного гравця.

Код використовує рекурсивні функції та алгоритми комбінаторики для генерації унікальних комбінацій гравців у коаліціях. Це дозволяє врахувати різноманітні можливості співпраці між гравцями. Крім того, алгоритм може бути легко адаптований для різних моделей гри та сценаріїв, що дозволяє використовувати його для різних досліджень і додавання нових застосувань за потреби.

Враховуючи можливість коаліційного співробітництва, розглянута модель покликана покращити становище учасників гри порівняно з рівновагою за Нешем. Цікавим аспектом є існування стаціонарної міжкоаліційної рівноваги, яка може сприяти оптимальній кооперації гравців. Важливою особливістю є те, що оптимальні стратегії кожної коаліції є ідентичними, що дозволяє ефективно вирішувати проблеми співпраці. Додаткові експерименти вказують на те, що різноманітність коаліційних утворень може мати значний вплив на ефективність відтворення ресурсу, що відкриває можливості для створення стійких соціальних механізмів в управлінні ресурсами.

Отримані результати можуть бути використані для аналізу різних економічних ситуацій та стратегічного планування в умовах конкуренції та обмежених ресурсів. Такі дослідження можуть мати практичне значення для різних галузей економіки, де важливо розуміти, як формуються та функціонують коаліції гравців.

Приклад 2

Розглянемо стохастичну гру накопичення ресурсу з нескінченним часом, у якій виконуються умови моделі I з логарифмічної функцією корисності.
Кількість гравців n = 2, α = 0.6, β = 1/1.02, часовий відлік - T = 10,
$x_0 = 0.025\, x^*$ початковий стан, в якому $x^* = (\alpha\beta)^{\frac{\alpha}{1-\alpha}}$. Будемо вважати, що всі рішення є симетричними з ідентичними c_i для кожного гравця.

Алгоритм пошуку розв'язків рівнянь (3.3-3.4) та (4.3-4.4) полягає в знаходженні виграшів та профілів гравців для кожного дискретного моменту часу $t \in [T]$ для кожного значення x з проміжку [0, ½], розбитого на 100 точок.

Для знаходження соціального оптимуму, спочатку, ми знаходимо початкове значення виграшу з рівняння (3.0) $V(T+1, x)$ для всіх x на проміжку. Оскільки T = 10, ми будемо шукати значення виграшу для останнього моменту часу t = 11 за допомогою функції v_tplus1(x, n) з Додатку 2. Функція v_tplus1_values(x, n) повертатиме список значень функцій виграшів для визначених точок на проміжку для кількості гравців n.

Починаючи з останнього моменту часу t = 11, до t = 1 ми рекурсивно шукатимемо значення $V(t, x)$ з рівняння (3.3) для всіх точок на проміжку, використовуючи знайдені значення для наступного моменту часу. Оскільки точка $(x - nc_i)^\alpha$ не належить вибраному проміжку, значення виразу шукатимемо за допомогою кубічної інтерполяції з бібліотеки scipy.interpolate в Python. На кожному кроці часу t задається нова функція інтерполяції.

На першому кроці роботи алгоритму, ми розбиваємо заданий інтервал (в даному випадку [0, ½]) на задану кількість точок. Далі ми визначаємо функції для знаходження кінцевого виграшу в стані x для кількості гравців n для обох станів рівноваги та відповідні допоміжні функції, що обраховують виграші для всіх точок на заданому проміжку.

Для подальшої роботи алгоритму ми визначаємо параметри з якими ми будемо працювати та змінні, в які записуватимемо отримані результати. Основний метод, який відповідає за знаходження значень цільової функції працює за наступним алгоритмом:

1) Ініціалізуються значення цільової функції для останнього моменту часу T. Використовуючи бібліотеки Python для кубічної інтерполяції, ми визначаємо функцію на основі отриманих значень.
2) Створюємо цикл з ітераціями від моменту часу T до початкового моменту часу. Для кожного моменту часу t ми знаходимо для всіх точок на проміжку ми знаходимо таке значення c, яке є локальними мінімумом (точка $x^* \in \Omega$ є локальним мінімумом оптимізаційної проблеми $min_{x \in \Omega} f(x)$, якщо існує таке $\varepsilon > 0$, що $f(x^*) \leq f(x)$ при $x \in \Omega$, що задовольняє $\left||x - x^*|\right| \leq \varepsilon$) для рівнянь (3.3)-(3.4).
3) Результати записуємо у змінні. Для наступного кроку ми зберігаємо у пам’яті попередні значення цільової функції та знову визначаємо кубічну інтерполяцію. Повторюємо на всіх ітераціях для кожного дискретного моменту часу t.

Для побудови графіків з отриманими результатами ми створюємо допоміжні функції для розбиття результатів на відповідні проміжки часу та для запису їх у відповідні структури даних. За допомогою бібліотек Python для побудови

малюнків в системі з трьома координатними осями, ми будуємо точкову діаграму value function та consumption.

Отримані результати значень функцій $V(t,x)$ та $c_i(t,x)$ можемо побачити у вигляді графіків з двома залежними змінними.

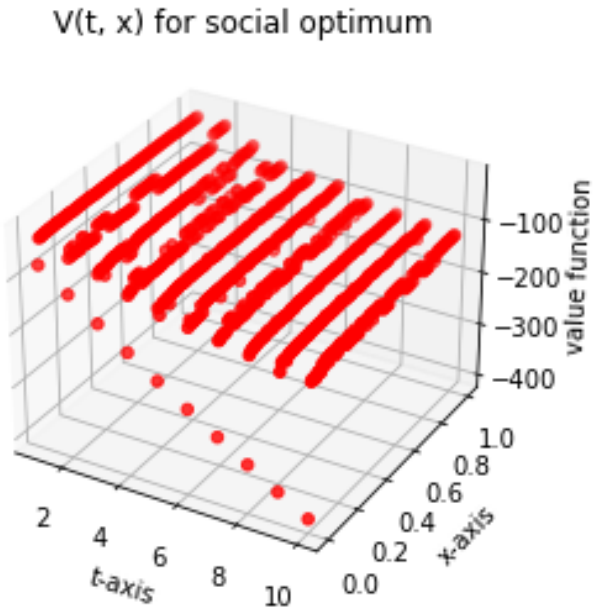

Рисунок 3 Значення функції V(t, x) в моменти часу t для x на заданому проміжку для соціального оптимуму

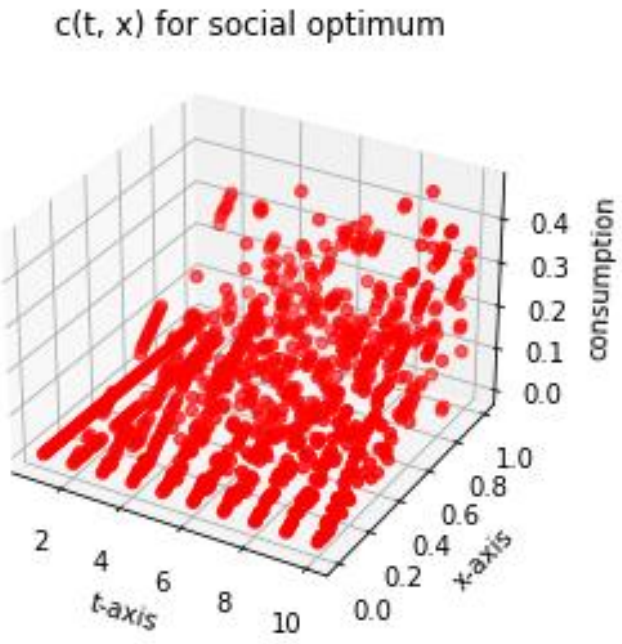

Рисунок 4 Значення функції c(t, x) в моменти часу t для x на заданому проміжку для соціального оптимуму

Для знаходження рівноваги за Нешем, спочатку, ми знаходимо початкове значення виграшу з рівняння (4.0) $V(T+1,x)$ для всіх x на проміжку.

Починаючи з останнього моменту часу t = 11, до t = 1 ми рекурсивно шукатимемо значення допоміжних функцій $V_{\bar{c}}(t,x,\bar{c})$ та $c_{\bar{c}}(t,x,\bar{c})$з рівняння (3.3) почергово для кожного дискретного моменту часу $t \in [T]$ для кожного значення $\bar{c}$ з проміжку [0, ½], розбитого на 21 точку. Оскільки точка $(x - c - (n-1)\bar{c})^{\alpha}$ не належить вибраному проміжку, значення виразу шукатимемо за допомогою кубічної інтерполяції з бібліотеки scipy.interpolate в Python. Зі всіх значень $\bar{c}$ ми вибираємо таке, що мінімізує вираз $|\bar{c} - c_{\bar{c}}(t,x,\bar{c})|$. Оскільки $\bar{c} \in [0,\frac{1}{2}]$, ми можемо змінити область визначення $\bar{c}$ на сусідні значення $[\bar{c}_{i+1},\bar{c}_{i-1}]$. Знову розіб'ємо проміжок на 21 точку, та повторюватимемо попередні кроки

доти, поки не досягнемо заданої точності розрахунків. Було вибрано зробити 4 ітерації.

Перші кроки роботи алгоритму співпадають з тими, що було зроблені для визначення соціального оптимуму, - ми задаємо параметри з умови задачі та розбиваємо заданий інтервал (в даному випадку [0, ½]) на задану кількість точок. Для знаходження кінцевого виграшу в стані x для кількості гравців n для обох станів рівноваги в момент часу T ми користуємося функціями визначеними напередодні.

Для подальшої роботи алгоритму ми визначаємо змінні, в які записуватимемо отримані результати. Основний метод, який відповідає за знаходження значень цільової функції працює за наступним алгоритмом:

1) Ініціалізуються значення цільової функції для останнього моменту часу T згідно формули (4.0). Використовуючи бібліотеки Python для кубічної інтерполяції, ми визначаємо функцію на основі отриманих значень,
2) Створюємо цикл з ітераціями від моменту часу T до початкового моменту часу. Для кожного моменту часу t ми задаємо інтервал [0, 1/n] та розбиваємо його на задану кількість точок (в нашому випадку ми обрали c_gridpoints = 21). Далі в новому циклі ми шукаємо таке c яке є локальними мінімумом для рівнянь (3.3)-(3.4) для кожної з точок на інтервалі [0, 1/n]. З отриманих мінімумів ми обираємо найменше c, та замінюємо інтервал, на той що відповідає вибраному мінімуму. Повторюємо процедуру для нового інтервалу до настання умови завершення циклу.
3) Результати записуємо у змінні. Для наступного кроку ми зберігаємо у пам’яті попередні значення цільової функції та знову визначаємо кубічну

інтерполяцію. Повторюємо на всіх ітераціях для кожного дискретного моменту часу t.

Для побудови графіків з отриманими результатами, так само, як і для соціального оптимуму, ми створюємо допоміжні функції для розбиття результатів на відповідні проміжки часу та для запису їх у відповідні структури даних. За допомогою бібліотек Python для побудови малюнків в системі з трьома координатними осями, ми будуємо точкову діаграму value function та consumption.

За допомогою допоміжних функцій можемо обрахувати значення $V(t,x)$ та $c_i(t,x)$.

Отримані результати значень функцій $V(t,x)$ та $c_i(t,x)$ представимо у вигляді графіків з двома залежними змінними.

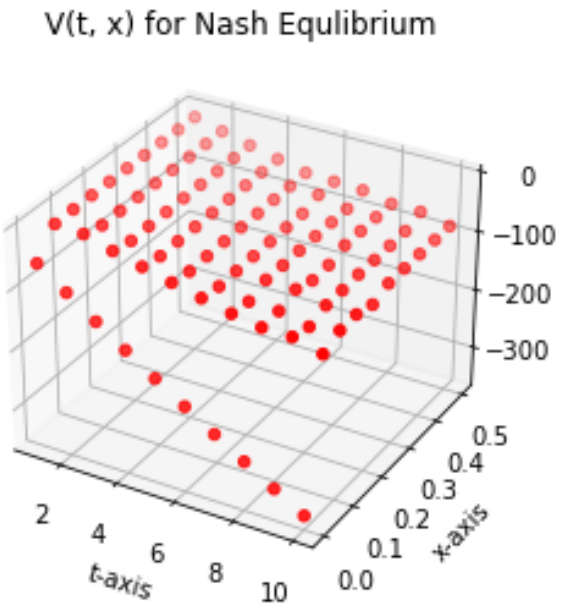

Рисунок 5Значення функції V(t, x) в моменти часу t для x на заданому проміжку для рівноваги Неша

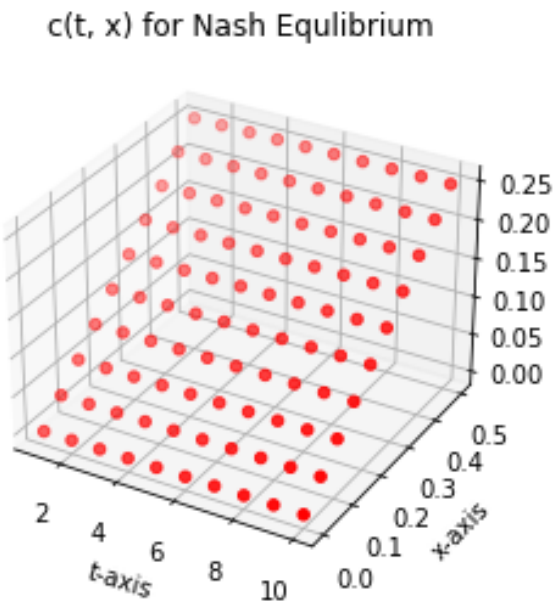

Рисунок 6 Значення функції c(t, x) в моменти часу t для x на заданому проміжку для рівноваги Неша

З міркувань доцільності, $\bar{c}$ було представлено у вигляді виразу ax, та оптимізація розраховувалась над сталим інтервалом [0, ½] для уникнення проблем при $V(t,x) \to \infty$ коли $x \to 0$.

Щоб пояснити відхилення при визначення функції виграшу визначимо значення V та V^{num} як функцію виграшу в проблемі оптимізації та її наближене значення, та OPT і OPT^{num} як оптимальні стратегії та вичислені оптимальні стратегії відповідно. Визначимо такі умови, за яких відхилення не впливають на точність обчислень: [2]

1) Якщо $V(t,x) \leq V^{num}(t,x)$ для всіх t та x, таких що для $\forall c \in OPT^{num}$ $x \neq X(t)$, де X – траєкторія, що відповідає c, і $V(t,x) = V^{num}(t,x)$ у всіх інших точках, тоді різниця не впливає на точність розрахунків траєкторій для стану рівноваги та траєкторію витрат $c\big(t, X(t)\big)$.

2) Якщо $V(t,x) \geq V^{num}(t,x)$ для всіх t та x, таких що для $\forall c \in OPT^{num}$ $x \neq X(t)$, де X – траєкторія, що відповідає c, і $V(t,x) = V^{num}(t,x)$ у всіх інших точках, тоді різниця не впливає на точність розрахунків траєкторій для стану рівноваги та траєкторію витрат $c\big(t, X(t)\big)$.

3) Різниця $V(t,x) - V^{num}(t,x)$ визначена на множині станів (t,x), така що для кожної стратегії c з множини $OPT \cup OPT^{num}$ та для $x \neq X(t)$, не впливає на точність розрахунків траєкторій для стану рівноваги та траєкторію витрат.

Це можна довести за допомогою доведення від супротивного взявши будь-яке c_0, яке не максимізує рівняння

$$n \ln c + \beta V^*(t+1, (x - nc)^\alpha)$$

для соціального оптимуму та рівняння

$$\ln c + \beta V^*(t+1, (x - c - (n-1)\bar{c})^\alpha)$$

для рівноваги за Нешем. Для c, що максимізує рівняння $V(*) = V^{num}(*)$, тоді як $V(*) \geq V^{num}(*)$ в іншому випадку. Отже для такого c, що максимізує дані рівняння, функція виграшу в проблемі оптимізації та її наближене значення даватимуть однакові результати.

Підсумки та пропозиції для подальшого покращення моделі

Запропонована модель відтворює ситуацію в грі для кількості гравців n. Метою кожної сторони є максимізувати миттєві та кінцеві виплати. Використана техніка максимізації використовує методи динамічного програмування, яке в контексті моделі є аналогом стохастичної гри в дискретному часі. Створений алгоритм для отримання $c(t,x)$ та $V(t,x)$ моделі підтверджує висновки зроблені в [3] – рівновага Неша призводить до більшого використання ресурсу, як функції, що залежить від розміру капіталу. Для лінійної моделі це не виключає повне виснаження ресурсу.

Алгоритм імплементує модель гри з дискретним часом та неперервним простором стратегій, що дозволяє аналізувати поведінку гравців та їхні наслідки на кожному кроці гри. Код, в тому числі, включає застосування числових методів оптимізації, таких як minimize_scalar з бібліотеки SciPy, для знаходження оптимальних стратегій гравців. Була застосована кубічна інтерполяція для апроксимації значень функцій виграшу та стратегій між визначеними точками, що дозволимо отримати результати та графіки залежності значень функцій від часу та стану гри.

Отримані результати були візуалізовані в тривимірних координатах, що допомагає в розумінні динаміки гри та впливу стратегій гравців на результати. Графіки демонструють, як змінюються значення функцій виграшу та стратегій зі зміною часу та стану гри, що сприяє легкому аналізу та порівнянню різних стратегій.

Реалізація включає алгоритм для знаходження рівноваги Неша, що дозволяє визначити оптимальні стратегії гравців у випадку, коли вони діють відповідно до концепції Неша.

Покращення існуючої моделі полягає в виведенні моделі що дозволяє гравцям вступати в коаліції та взаємодіяти з метою отримання більших нагород. В роботі [3] розглядається різновиди симетричної гри з розбиттям на одну коаліцію з можливістю виходу та вступу до формування. Існування такої моделі включає точку рівноваги з $\eta = n - m + 1$ гравцями для загальної кількості агентів n та m агентів, що беруть участь у формуванні. Функція виграшу для і-того збігається з визначеною для моделі II

$$V_i(s) = \max_{c_i}\{\ln c_i + \beta V_i((s - \sum_{k=1}^{n} x_k)^a)\}$$

Для визначеного формування були зроблені висновки, що кожен учасник такого формування отримуватиме в m разів менший виграш, ніж агент, що утримався від вступу до формування. Для визначеного параметру $\delta = \frac{\alpha\beta}{1-\alpha\beta} > 0$ з його збільшенням зменшується залишковий запас ресурсу. В той же час, існує обернена залежність між кількістю агентів, що не брали участі в формуванні та функції залишкового запасу ресурсів. Це приводить нас до результатів, що було отримані для степеневої функції – участь в коаліціях позитивно впливає на кількість та відновлюваність ресурсу. Також збільшення кількості учасників в коаліції при загальній прибутковості є в інтересах учасників коаліції. Наостанок, для даної кількості ресурсу s, змінної з часом, вигода для гравців поза коаліцією є вищою для всіх $m > 1$.

Формування є вигідним лише в тому випадку, коли винагорода в кооперації є більшою за одноосібну гру для всіх сторін. Для цього визначається функція вигідності $\pi : [1, n] \to R$: [3]

$$\pi(m) = p_m^c(s) - \ p_1^0(s) = (1-\beta)^{-1}((\delta+1)\ln\frac{n+\delta}{\eta+\ \delta} - \log m)$$

В якій $p_m^c(s)$ та $p_1^0(s)$ функції нагороди для гри в формуванні та без відповідно. Аналіз функції показує, що доречність в створенні формуванні є при достатньо великих значеннях $\alpha\beta$.

Також в роботі розглядались випадки гри з більше ніж однією коаліцією, які підпадали під умови внутрішньої стабільності – жодне формування не втрачає учасників та зовнішньої стабільності – агенти не приєднуються до вже утворених кооперувань. Було зазначено що для більшості значень параметрів α та β не існуватиме стабільних формувань, які задовольняли би попередні умови.

Згідно попередніх спостережень, незважаючи на переваги розширення кооперацій, гравці поза формуванням не збільшать свого прибутку за спільної гри для $m \geq 2$, та лише в окремих випадках прослідковується вигода, як наприклад формування з чотирьох гравців на противагу одноосібної гри проти формування з двох гравців. Очевидно, що перспектива розширення коаліції не піддається поняттю зовнішньої стійкості щодо коаліції розміром два, яка може відповідно розширюватися до коаліції розміром чотири.

Отже, підсумовуючи, участь у коаліціях може бути вигідною лише у випадку, коли вигода від співпраці перевищує вигоду від одиночної гри для всіх сторін.

ВИСНОВОК

Результатом дослідження стала реалізація моделей гри з різними функціями корисності у середовищі програмування Python та C#. Це дозволило впровадити теоретичні результати в практичні застосування та визначити оптимальні стратегії для різних умов.

Був реалізований алгоритм, який відтворює процес гри для симетричної моделі стохастичної гри видобутку ресурсів, де гравці об'єднані в фіксовані коаліції з початковими даними, такими як кількість агентів та коефіцієнти, що використовуються різними методами і функціями для визначення стратегії гравців та їх виграшів.

Основною метою алгоритму є обчислення стратегій в точці рівноваги та виграшів, які можуть бути досягнуті при об’єднанні всіх гравців в єдину коаліцію або в декілька різних. Це дозволяє порівняти виграші від різних варіантів коаліційних утворень та визначити оптимальні стратегії для кожного гравця.

У реалізації алгоритму використовуються рекурсивні функції та алгоритми комбінаторики для генерації унікальних комбінацій гравців у коаліціях. Це дозволяє врахувати різноманітні можливості співпраці між гравцями. Крім того, алгоритм може бути легко адаптований для різних моделей гри та сценаріїв, що дозволяє використовувати його для різних досліджень і додавання нових застосувань за потреби.

Аналіз результатів вказує на те, що модель сприяє покращенню становища учасників гри порівняно з рівновагою за Нешем. Особливо важливою є можливість формування стаціонарної міжкоаліційної рівноваги, яка сприяє оптимальній кооперації гравців. Важливо також враховувати вплив різноманітності коаліційних утворень на ефективність відтворення ресурсу.

Отримані результати можуть мати практичне застосування в аналізі економічних ситуацій та стратегічному плануванні в умовах конкуренції та обмежених ресурсів. Дослідження в цьому напрямку можуть допомогти в розумінні процесів формування та функціонування коаліцій гравців у різних галузях економіки.

Запропонований алгоритм значно розширює можливості аналізу та розв'язання завдань у контексті гри з видобутку ресурсів. Він відображає ситуацію в грі для будь-якої кількості гравців та дозволяє досліджувати різні варіанти стратегій та їх наслідки на результати гри.

Список використаних джерел

1. Силенко І. В. Марковська ідеальна рівновага у грі видобутку ресурсів зі степеневими перевагами агентів // Київ. - 2023. – С. 39 – 138.

2. Agnieszka Wiszniewska-Matyszkiel, Rajani Singh. Numerical versus analytic calculation of optima and equilibria in Fish Wars model with finitetime horizon // Institute of Applied Mathematics and Mechanics Warsaw University. - 2016. - С. 3 – 11.

3. M. Breton, M. Y. Keoula. Farsightedness in a Coalitional Great Fish War // Springer Science+Business Media B.V. - 2011 - С. 298-314.

4. M. Breton, M. Y. Keoula. A Great Fish War Model with Asymmetric Players // Montr´eal, - 2010.

Додатки

Додаток 1

```
namespace Dyploma;
using System;

public class MyProgram
{
    int m = 5; //number of agents
    int m_co = 4;
    float beta = 0.8f; // discount rate

#region
    float mu = 0.22f; // average value
    float sigma = 0.0004f; // dispersion

    float alpha = 0.5f;

    float c_co = 2;
    float alpha_co = 0.45f;
    float beta_co = 0.9f;


#endregion
    public void Run()
    {
        float l = LogNormalMoments(mu, sigma, alpha_co, beta_co);
        Console.WriteLine(l);
        DiscountWinsVector(4, c_co, alpha_co, l);
        Console.WriteLine("***");
        float l2 = LogNormalMoments(mu, sigma, 0.6f, 0.65f);
        Console.WriteLine(l2);
        DiscountWinsVector(4, 10, 0.6f, l2);
    }

    public float LogNormalMoments(float mu, float sigma, float alpha, float beta)
    {
        float l = (float) Math.Exp(mu*alpha + (sigma*alpha*alpha) / 2.0)*beta;
        if (l>0 && l<1) return l;
        return 0;
    }

    public void DiscountWinsVector(int m, float c, float alpha, float l)
    {
        List<List<int>> combinations = GetUniqueCombinations(m);
```

```
        foreach (var combination in combinations) {
            List<float> vector = new List<float>();
            int coalitionNum = combination.Count;
            if (coalitionNum == 1) {
                for (int i = 0; i < combination[0]; i++) {
                    vector.Add(ContiniousUtilityFunction(c, m, l, alpha));
                }
            } else {
                float coalitionWin = CalculateLimit(c, coalitionNum, l, alpha,
1000);
                foreach (var item in combination)
                {
                    if (item == 1) {
                        vector.Add(coalitionWin);
                    } else {
                        for (int i = 0; i < item; i++) {
                            vector.Add(DiscountWin(coalitionWin, item, alpha));
                        }
                    }
                }
            }


            string comb = "[";
            foreach (int item in combination)
            {
                comb+= " " + item ;
            }

            Console.WriteLine(comb + " ]" + "(" + string.Join(", ", vector) + ")
*s1^a");
        }
    }


    static List<List<int>> GetUniqueCombinations(int target)
    {
        List<List<int>> combinations = new List<List<int>>();
        HashSet<List<int>> uniqueCombinations = new HashSet<List<int>>(new
ListComparer<int>());
        GenerateCombinations(target, new List<int>(), combinations,
uniqueCombinations);
        return combinations;
    }
```

```
    static void GenerateCombinations(int remaining, List<int> currentCombination,
List<List<int>> combinations, HashSet<List<int>> uniqueCombinations)
    {
        if (remaining == 0)
        {
            List<int> sortedCombination = currentCombination.OrderBy(x =>
x).ToList();
            if (uniqueCombinations.Add(sortedCombination))
            {
                combinations.Add(new List<int>(sortedCombination));
            }
            return;
        }

        for (int i = 1; i <= remaining; i++)
        {
            currentCombination.Add(i);
            GenerateCombinations(remaining - i, currentCombination, combinations,
uniqueCombinations);
            currentCombination.RemoveAt(currentCombination.Count - 1);
        }
    }


    class ListComparer<T> : IEqualityComparer<List<T>>
    {
        public bool Equals(List<T> x, List<T> y)
        {
            return x.OrderBy(e => e).SequenceEqual(y.OrderBy(e => e));
        }

        public int GetHashCode(List<T> obj)
        {
            return obj.OrderBy(e => e).Aggregate(17, (acc, val) => acc * 23 +
val.GetHashCode());
        }
    }

#region SMPE
    public float SMPEKSequence(int t, float c, float m, float l, float alpha)
    {
        if (t == 1) return c/MathF.Pow(m, alpha);
        else {
            float temp = MathF.Pow(SMPEKSequence(t - 1, c, m, l, alpha)*l/c,
1.0f/(1-alpha));
            float res = (c*(1+temp))/MathF.Pow(m + temp, alpha);
```

```
            return res;
        }
    }

    public float CalculateLimit(float c, float m, float l, float alpha, int
maxIterations = 1000)
    {
        float result = 0.0f;
        for (int t = 1; t <= maxIterations; t++) {
            float currentResult = SMPEKSequence(t, c, m, l, alpha);
            if (MathF.Abs(currentResult - result) < 1e-6) {
                result = currentResult;
                break;
            }
            result = currentResult;
        }
        return result;
    }

    public float SMPEChoiceFunction(int t, float c, float m, float l, float alpha)
    {
        if (t == 1) return 1/m;
        else {
            float temp = MathF.Pow(SMPEKSequence(t - 1, c, m, l, alpha)*l/c,
1.0f/(1-alpha));
            float res = 1/(m + temp);
            return res;
        }
    }

    public float DiscountWin(float win, int agentsNumber, float alpha) //
(k*)/mi^(-alpha)*s^alpha
    {
        return win*MathF.Pow(agentsNumber, -1*alpha);
    }

#endregion


#region Social Optimum
    public float KSequence(int t, float c, float m, float l, float alpha)
    {
        if (t == 1) return c/MathF.Pow(m, alpha);
        else {
            float temp = MathF.Pow(m*KSequence(t - 1, c, m, l, alpha)*l/c, 1/(1-
alpha));
```

```
            return c*MathF.Pow(m + temp, 1- alpha)/m;
        }
    }

    public float ChoiceFunction(int t, float c, float m, float l, float alpha)
    {
        if (t == 1) return 1/m;
        else {
            float temp = MathF.Pow(m*KSequence(t - 1, c, m, l, alpha)*l/c, 1.0f/(1-
alpha));
            float res = 1/(m + temp);
            return res;
        }
    }

    public float ContiniousChoiceFucntion(float m, float l, float alpha)
    {
        return (1-MathF.Pow(l, 1/(1-alpha)))/m;
    }

    public float ContiniousUtilityFunction(float c, float m, float l, float alpha)
    {
        return c/(MathF.Pow(m, alpha)*MathF.Pow(1 - MathF.Pow(l, 1/(1-alpha)), 1 -
alpha));
    }

#endregion

}

public class Program
{
    static void Main(string[] args)
    {

        MyProgram myProgramInstance = new MyProgram();


        myProgramInstance.Run();
    }
}
```

Додаток 2

[1]:

```
from scipy.optimize import minimize_scalar
from mpl_toolkits import mplot3d
```

[2]:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

[3]:

```
def split_interval(left_border, right_border, num):
    if num - 1 <= 0:
        raise ValueError("Number of intervals (x) must be greater than 0")

    interval_size = (right_border - left_border) / (num - 1)
    result = [round(left_border + i * interval_size, 10) for i in range(num)]
    return result
```

[4]:

```
import math

def V_Tplus1(x, n):
    if x == 0:
        print("got zero")
        return 0  # Handle the case where x is zero to avoid division by zero
    else:
        return n * np.log(x/n)

def V_Tplus1_values(x_values, n):
    return [V_Tplus1(x, n) for x in x_values]

#Nash Equilibria

def NashV_Tplus1(x, n):
    if x == 0:
        return 0  # Handle the case where x is zero to avoid division by zero
    else:
        return np.log(x/n)

def NashV_Tplus1_values(x_values, n):
    return [NashV_Tplus1(x, n) for x in x_values]
```

[5]:

```
from scipy.interpolate import CubicSpline
```

```
import matplotlib.pyplot as plt
```

```
[6]:
```

```
alpha = 0.6
beta = 1/1.02
T_max = 10
n = 2

x_grid_left_border = 0.00000001
x_grid_right_border = 1
num = 100

x_list = split_interval(x_grid_left_border, x_grid_right_border, num)
initial_values = V_Tplus1_values(x_list, n)

# Cubic interpolation
cubic_interp = CubicSpline(x_list, initial_values, bc_type='natural')
f_x_function = cubic_interp

v_values = []
c_values = []

def valueFunction(T):
    values = V_Tplus1_values(x_list, n)
    cubic_interp = CubicSpline(x_list, values, bc_type='natural')
    f_x_function = cubic_interp
    for t in range(T, 0, -1):
        res_val = []
        c_val = []
        for x in x_list:
            result = minimize_scalar(lambda c: n*np.log(c) +
                                     beta*f_x_function((x-n*c)**alpha), bounds=(0,
x/n), method='bounded')
            supremum = result.fun  #change bounds
            argmax = result.x
            res_val.append(supremum)
            c_val.append(argmax)
        v_values.append(res_val)
        c_values.append(c_val)
        values = res_val
        cubic_interp = CubicSpline(x_list, values, bc_type='natural')
        f_x_function = cubic_interp


valueFunction(10)
```

```
[7]:
```

```
def t_values(v_values):
    t_val = []
    i = 0
```

```
    for t in v_values:
        i += 1
        for v in t:
            t_val.append(i)
    return t_val

def x_values(v_values, x_list):
    x_val = []
    for t in v_values:
        for i in range(len(t)):
            x_val.append(x_list[i])
    return x_val

def vtx_values(v_values):
    vtx_val = []
    for t in v_values:
        for v in t:
            vtx_val.append(v)
    return vtx_val

def ctx_values(c_values):
    c_val = []
    for t in c_values:
        for v in t:
            c_val.append(v)
    return c_val

#print(vtx_values(v_values))
```

[8]:

```
from mpl_toolkits.mplot3d import Axes3D

# Example data
Y = x_values(v_values, x_list)
X = t_values(v_values)
Z = vtx_values(v_values)

# Creating a 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Creating scatter plot
ax.scatter(X, Y, Z, c='r', marker='o')

# Setting labels and title
ax.set_xlabel('t-axis')
ax.set_ylabel('x-axis')
ax.set_zlabel('value function')
ax.set_title('V(t, x) for social optimum')

plt.show()
```

[9]:

```
 # Example data
Y = x_values(v_values, x_list)
X = t_values(v_values)
Z = ctx_values(c_values)

# Creating a 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Creating scatter plot
ax.scatter(X, Y, Z, c='r', marker='o')

# Setting labels and title
ax.set_xlabel('t-axis')
ax.set_ylabel('x-axis')
ax.set_zlabel('consumption')
ax.set_title('c(t, x) for social optimum')

plt.show()
```

[10]:

```
grid_left_border = 0.0000000001
grid_right_border = 1/2
num = 10
alpha = 0.6
beta = 1/1.02
T_max = 10
n = 2
```

[11]:

```
c_grpoints = 21
res_val = []
c_val = []
c_grpoints_list = split_interval(0, 1/2, c_grpoints)
max_iterations = 2
T = 10

nash_x_list = split_interval(grid_left_border, grid_right_border, num)

nash_v_values = []
nash_c_values = []
nash_c_rig = []

def nash_value_function(T):
    init_values = NashV_Tplus1_values(nash_x_list, n)
    nash_cubic_interp = CubicSpline(nash_x_list, init_values, bc_type='natural')
```

```
    nash_f_x_function = nash_cubic_interp
    for t in range(T, 0, -1):
        nash_res_val = []
        nash_c_val = []
        for x in nash_x_list:
            supremum = 0
            argmax = 0
            l_neighbour = 0
            r_neighbour = 1/n
            c_grpoints_list = split_interval(l_neighbour, r_neighbour,
c_grpoints)
            for j in range(max_iterations):
                dif = 0
                c_iteration = 0
                for i in range(len(c_grpoints_list)):
                    result = minimize_scalar(lambda c: np.log(c*x) +
beta*nash_f_x_function((x - c*x - (n - 1) * c_grpoints_list[i]*x)**alpha),
                                             bounds=(l_neighbour, r_neighbour),
method='bounded')
                    temp_supremum = result.fun
                    temp_argmax = result.x*x
                    temp_dif = np.abs(temp_argmax - c_grpoints_list[i])
                    if (i == 0):
                        dif = temp_dif
                        supremum = temp_supremum
                        argmax = temp_argmax
                        #print(supremum)
                    if (temp_dif < dif):
                        dif = temp_dif
                        c_iteration = i
                        supremum = temp_supremum
                        argmax = temp_argmax
                if (dif == 0): break
                if(i == 0):
                    l_neighbour = c_grpoints_list[0]
                else:
                    l_neighbour = c_grpoints_list[i-1]
                if(i == len(c_grpoints_list) - 1):
                    r_neighbour = c_grpoints_list[i]
                else:
                     r_neighbour = c_grpoints_list[i+1]
                c_grpoints_list = split_interval(l_neighbour, r_neighbour,
c_grpoints)
            print(argmax)
            nash_res_val.append(supremum)
            nash_c_val.append(argmax)
        nash_v_values.append(nash_res_val)
        nash_c_values.append(nash_c_val)
        nash_cubic_interp = CubicSpline(nash_x_list, nash_res_val,
bc_type='natural')
        nash_f_x_function = nash_cubic_interp
        print(nash_f_x_function(0.3))
```

```
nash_value_function(T)
```

```
[14]:
```

```
yn = x_values(nash_v_values, nash_x_list)
xn = t_values(nash_v_values)
zn = vtx_values(nash_v_values)

print(zn)
```

```
[15]:
```

```
# Creating a 3D plot for V(t, x)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Making scatter plot
ax.scatter(xn, yn, zn, c='r', marker='o')

# Setting labels and title
ax.set_xlabel('t-axis')
ax.set_ylabel('x-axis')
#ax.set_zlabel('value function')
ax.set_title('V(t, x) for Nash Equlibrium')


plt.show()
```

```
[16]:
```

```
# Example data
CN = ctx_values(nash_c_values)

# Creating a 3D plot for c(t, x)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Making scatter plot
ax.scatter(xn, yn, CN, c='r', marker='o')

# Setting labels and title
ax.set_xlabel('t-axis')
ax.set_ylabel('x-axis')
#ax.set_zlabel('consumption')
ax.set_title('c(t, x) for Nash Equlibrium')

plt.show()
```