

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут імені Ігоря Сікорського»
Фізико-технічний інститут

«На правах рукопису»

УДК 501

«До захисту допущено»

Завідувач кафедри

С.О. Воронов

(підпис)

(ініціали, прізвище)

Магістерська дисертація
на здобуття ступеня магістра

зі спеціальності 105 «Прикладна фізика та наноматеріали»

(код і назва)

на тему: **Підвищення інформаційної ефективності молекулярних машин**

Виконав: студент 2 курсу, групи ФФ-91-2мн

Поляков Володимир Юрійович

(прізвище, ім'я, по батькові)

(підпис)

Науковий керівник доцент, к. т. н., доц., Гордійко Н. О

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Консультант

(назва розділу)

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Рецензент д.т.н., проф. Воронов С.О.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

РЕФЕРАТ

Молекулярні машини та наномотори лежать в основі кожного важливого біологічного процесу. У процесі еволюції неспроста відточувались саме такі механізми функціонування живих організмів. Багато з найкращих винаходів людства запозичені із живої природи. Настав час створювати свої молекулярні машини.

За останні два десятиліття людство досягло значного прогресу стосовно можливостей створювати власні молекулярні машини. Збільшуються амплітуди на які вони можуть рухатись та кількість роботи яку вони можуть виконувати. Вчені все краще можуть керувати створеними молекулярними машини та все краще розуміють принципи їх функціонування.

Незважаючи на великий прогрес у цій галузі досягнуто немало прогресу, вона є новою та існує багато напрямків у яких дослідники можуть рухатись. Зокрема є необхідність у більшій кількості фундаментальних досліджень, щоб збільшити обсяг знань про машини таких розмірів, які можуть виконувати корисну роботу.

В переважній більшості для опису роботи таких машин використовується статистична фізика, оскільки законами класичної механіки настільки малі механізми описати неможливо. Окрім того, вони оперують у середовищах що кардинально визначають їх проведінку, наприклад зміна концентрацій або температур. В таких умовах у молекулярних машин виникає необхідність отримувати інформацію про стан середовища, що теж неможливо без затрачення певної кількості енергії. Саме ця проблема досліджується у роботі. Проблема отримання максимальної кількості інформації молекулярною машиною про середовище у якому вона знаходиться, з мінімальними затратами енергії.

Об'єктом дослідження є модель молекулярної машини, розроблена авторами на основі існуючої та способи її оптимізації. Запропонована авторами модель є більш гнучкою та адаптивною.

Метою роботи є дослідження моделі молекулярної машини, що дозволить проаналізувати інформаційну ефективність, ввести її чітке визначення та запропонувати спосіб її підвищити.

У даній роботі було проаналізовано вже існуючі математичні моделі молекулярних машин, а саме ті роботи, що фокусуються на швидкості отримання інформації машиною та енергетичній ефективності. На основі однієї з таких моделей, що використовує Марківський процес, було розроблено нову математичну модель молекулярної машини. Оскільки модель була розроблена на основі прототипу, було збережено зворотню сумісність та відтворено результати інших дослідників. Було проведено обширну перевірку роботи моделі та її відповідності реальним неперервним процесам, Досліджено енергію, що затрачається на функціонування машини та інформацію, що отримується в процесі. Для різних конфігурацій моделі знайдено траєкторії та параметри, що забезпечують оптимальне підвищення інформаційної ефективності. Зроблено відповідні висновки.

Проміжні результати роботи було опубліковано у вигляді тез доповіді на XVII Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми з фізики, математики та інформатики», 2019.

КЛЮЧОВІ СЛОВА

Інформація, теорія інформації, ентропія, молекулярні машини, інформаційна ефективність, оптимальна траєкторія.

ABSTRACT

Molecular machines and nanomotors underlie every important biological process. In the process of evolution, it was not without reason that such mechanisms of functioning of living organisms were honed. Many of the best inventions of mankind are borrowed from wildlife. It's time to build your own molecular machines.

Over the past two decades, humanity has made significant progress in building its own molecular machines. The amplitudes they can move to and the amount of work they can do increase. Scientists are better able to control the created molecular machines and better understand the principles of their operation.

Despite much progress in this area, considerable progress has been made, it is new and there are many areas in which researchers can move. In particular, there is a need for more basic research to increase knowledge about machines of a size that can do useful work.

The vast majority use statistical physics to describe the operation of such machines, because the laws of classical mechanics can not describe such small mechanisms. In addition, they operate in environments that radically determine their conductivity, such as changes in concentrations or temperatures. Under such conditions, molecular machines need to obtain information about the state of the environment, which is also impossible without consuming a certain amount of energy. This problem is investigated in the work. The problem of obtaining the maximum amount of information by the molecular machine about the environment in which it is, with minimal energy consumption.

The object of research is a model of a molecular machine developed by the authors on the basis of existing and methods of its optimization. The model proposed by the authors is more flexible and adaptive.

The aim of the work is to study the model of a molecular machine, which will analyze the information efficiency, introduce its clear definition and suggest a way to improve it.

In this paper, the existing mathematical models of molecular machines were analyzed, namely those works that focus on the speed of information retrieval by the machine and energy efficiency. Based on one of these models, which uses the Markov process,

a new mathematical model of the molecular machine was developed. Because the model was developed on the basis of a prototype, backward compatibility was maintained and the results of other researchers were reproduced. Extensive verification of the model operation and its compliance with real continuous processes was carried out. For different configurations of the model, trajectories and parameters have been found that provide an optimal increase in information efficiency. Appropriate conclusions are made.

Obtained intermediate results were published on XVII All-Ukrainian Scientific and Practical Conference of Students, Ph.D. Students, and Young Scientists.

Key words:

Information, information theory, entropy, molecular machines, information efficiency, optimal trajectory.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ.	7
ВСТУП	8
РОЗДІЛ 1. Молекулярні машини.	10
1.1. Класифікація та огляд сучасного стану молекулярних машин.	10
1.2. Базові величини, що використовуються у дослідженні	15
1.3. Огляд вихідної моделі та її симуляції	17
1.4. Висновки до розділу 1	18
РОЗДІЛ 2. Розробка симуляції молекулярної машини та її дослідження	19
2.1. Запропонована модель та досліджувані величини	19
2.2. Опис роботи симуляції	23
2.3. Відтворення наявних результатів.	24
2.4. Дослідження границь роботи симуляції	25
2.5. Висновки до розділу 2	27
РОЗДІЛ 3. Аналіз отриманих результатів.	28
3.1. Розсіяна енергія	28
3.2. Взаємна Інформації.	28
3.3. Інформаційна ефективність	29
3.4. Висновки до розділу 3	32
РОЗДІЛ 4. Пошук закономірностей	33
4.1. Змінне середовище	33
4.2. Пошук оптимальних траєкторій	37
4.3. Систематизація отриманих траєкторій	39
4.4. Висновки до розділу 4	42
ВИСНОВКИ	43
ПЕРЕЛІК ПОСИЛАНЬ	45
ДОДАТОК А. Код симуляції	47

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Латинські символи:

H — Інформаційна ентропія;

p — Імовірність;

I — Інформація;

W, E — Енергія;

F — Вільна енергія;

D — Розходження Кульбака-Лейбнера;

a, b — імовірності переходів;

X, Y — стани машини та середовища;

Індекси просторових компонент позначені латинськими літерами.

Грецькі символи:

φ — Характеристика нерівності;

k — Швидкість переходу;

β — Обернена температура;

Скорочення:

АММ — Штучні молекулярні машини.

ВСТУП

Мета та задачі. Мета роботи — Дослідження моделі молекулярної машини, що дозволить краще дослідити питання інформаційної ефективності та запропонувати спосіб її підвищити:

1) Аналіз наявної літератури на тему, зокрема робіт дослідницької команди з Simon Fraser University та їх моделі молекулярної машини.

2) Розробка нової моделі на основі прототипу. Написання коду для забезпечення функціонування даної моделі. Перевірка коректності моделі шляхом відтворення існуючих результатів.

3) Дослідження режимів роботи симуляції та її обмежень. Обрахунок можливих похибок.

4) Дослідження енергії, що затрачається машиною на функціонування та кількості інформації, що нею отримується. Введення нового означення інформаційної ефективності.

5) Пошук шляхів підвищення інформаційної ефективності.

Об'єкт дослідження. Розроблена математична модель молекулярної машини.

Предмет дослідження. Інформаційна ефективність молекулярних машин, зокрема запропонованої авторами.

Методи дослідження. Всі дослідження проведені з використанням мови програмування python, включаючи симуляцію та всі математичні методи аналізу, що було застосовано.

Наукова новизна отриманих результатів: Введено та досліджено поняття інформаційної ефективності та запропоновано спосіб її підвищення.

Практичне значення отриманих результатів. На основі даної роботи можливо проводити подальші дослідження молекулярних машин, зокрема стосовно інформаційної ефективності. Оскільки ще немає можливості створювати повноцінні

молекулярні машини в серійних масштабах, будь які їх дослідження будуть корисними. Зокрема, ця робота дає можливість зрозуміти, як саме можуть працювати більш ефективні сенсори, або молекулярні машини, що реагують на стан середовища.

Персональний внесок здобувача. Було запропоновано означення інформаційної ефективності та отримано результати, що дозволяють збільшити інформаційну ефективність запропонованої моделі.

Апробація результатів. Основні результати дослідження було обговорено й схвалено на XVI всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених.

РОЗДІЛ 1.

МОЛЕКУЛЯРНІ МАШИНИ

1.1. Класифікація та огляд сучасного стану молекулярних машин

Молекулярна машина, також наніт або наномашина – це молекулярний компонент, який виробляє квазімеханічний рух у відповідь на певні подразники (середовище). У біології макромолекулярні машини часто виконують важливі для життя функції, такі як: реплікація ДНК, транспортування молекул, виробництво кінетичної енергії і синтез АТФ. Назва часто більш широко застосовується до молекул, які просто імітують функції, які відбуваються на макроскопічному рівні.

Один з перших великих проривів у розвитку молекулярних машин відбувся на початку 1980-х років, коли французький хімік Жан-П'єр Соваж зробив механічно зчеплену молекулу, відому як катенан. Його конструкція складалася з двох нерозривних молекулярних кілець. Така конструкція також означала, що одне кільце могло обертатися навколо іншого при подачі енергії. Це було перше покоління молекулярних машин.

У 1990-х роках Джеймс Фрейзер Стоддарт розробив молекулу, названу ротаксаном [1]. У знаковій статті, яка розкрила світ молекулярних машин, Ротаксан являв собою перший молекулярний човник і був побудований з молекулярної осі, пронизаної через молекулярне кільце. Подаючи енергію, кільце можна було змуситись ковзати по довжині штока, забезпечуючи відповідь (рух) на вхід (енергія).

Пізніше в 1990-х голландський хімік Бернард Ферінга синтезував перший молекулярний двигун. Його конструкція дозволяла контролювати рух молекул у відповідь на світло і тепло. Поворотна конструкція використовувала світло, як джерело енергії для безперервного обертання в певному напрямку. Ферінга пішов ще далі і за допомогою свого молекулярного двигуна створив один з перших у світі нанокарів. Хоча, лише у кілька нанометрів довжиною та з робочою температурою -266°C .

У 2016 році троє вчених розділили Нобелівську премією з хімії за свою роботу, а Нобелівський комітет назвав інструменти, розроблені цими хіміками, "найменшими машинами у світі"[2].

За класифікацією існують природні та штучні молекулярні машини. Штучні машини називають АММ (Artificial Molecular Machine), тоді ж як біологічні називають біологічними і їх можна знайти у природі.

Одним з класів біологічних молекулярних машин є біологічні двигуни. Це біологічні молекули, здатні перетворювати хімічну енергію в рух і можуть бути важливими для таких біологічних функцій, як: скорочення м'язів, переміщення джгутиків бактерій та гідроліз АТФ. Ці двигуни можуть виробляти або лінійні (скорочення м'язів, рух джгутиків), або обертальні (АТФ-гідроліз) рухи.

Приклади біологічних молекулярних машин включають в себе міозин, кінезин, динеїн та рибосоми.

Міозин - це білок, що міститься у м'язах, та який відповідає за скорочення м'язів.

Кінезин - це білок, який переміщує "вантажі" всередині клітини.

Динеїн - це білок, який входить до джгутиків рухливих війок і відповідає за рух цих білків.

Рибосоми є важливою частиною синтезу білка, де мРНК перекладається у відповідний поліпептидний ланцюг. Під час цього процесу мРНК зчитується малою субодиницею, а велика субодиниця приєднується до відповідних амінокислот, утворюючи поліпептидний ланцюг.

Однією з переваг біологічних молекулярних машин є те, що вони здатні виконувати складні функції. Однак, будучи біологічними, вони є недостатньо стійкими. Штучні молекулярні машини хоча і є складними у розробці, володіють підвищеною стабільністю відносно біологічних. Розуміючи, як працюють біологічні молекулярні машини, можна створити інші, які зможуть виявляти ракові клітини, або подорожувати всередині людського тіла та виявляти потенційні проблеми зі здоров'ям.

Оскільки повністю синтетичні молекулярні машини вже існують не один рік з'явилося багато їх різновидів [3].

Молекулярні двигуни

Молекулярні двигуни - це молекули що створюють обертальний рух навколо зв'язку (одинарного або подвійного зазвичай). Однозв'язкові двигуни живляться енергією хімічних реакцій. Двозв'язкові працюють від енергії світла. В залежності від рішень використаних під час проектування молекулярного двигуна, можна досягти різних параметрів, таких як швидкість обертання [4].

Молекулярні пропеллери

Молекулярні пропелери - це молекули, що мають форму аналогічну звичайним пропеллерам і аналогічне призначення, перекачування рідин у процесі роботи. Молекулярні пропеллери будують шляхом прикріплення лопастей молекулярних масштабів до наномасштабного вала [5].

Молекулярні перемикачі

Молекулярний перемикач - це молекула, що володіє кількома стабільними станами (зазвичай двома) і має можливість їх змінювати. Вона може реагувати практично на будь-які зміни: рН, світла, температури, електричного струму, мікрооточення або у присутності ліганду [6].

Молекулярні човники

Молекулярний човник - це молекула, яка, аналогічно макроскопічному човнику, може курсувати між двома пунктами, а якщо точніше, переміщувати молекули або іони між цими пунктами. Зазвичай це цілісна конструкція, де ротаксан виступає у якості гантелі, тобто маршруту по якому курсує макроцикл [7].

Нанокари

Нанокари - це найцікавіші представники молекулярних машин. Це демонстраційні варіанти які не мають корисного призначення, окрім як, демонструвати можливості науки та підігрівати інтерес суспільства до галузі. Вони будуються за аналогією до справжніх автомобілів з рами та коліс, але їздять використовуючи молекулярну дифузію на поверхнях [8]. Перші нанокари були синтезовані Джеймсом М. Туром в 2005 році. Вони мали H-подібне шасі і 4 молекулярні колеса (фулерени), прикріплені до

чотирьох кутів. У 2011 році Бен Ферінга та його колеги синтезували перший моторизований нанокар, який мав молекулярні двигуни, прикріплені до шасі як обертові колеса. Автори мали змогу продемонструвати спрямований рух нанокару на поверхні міді, надаючи енергію від скануючого наконечника тунельного мікроскопа.

Згодом було проведено першу у світі гонку таких машин під назвою Nanocar у 2017 [9].

Молекулярні балансатори

Молекулярний балансатор - це молекула, яка може змінювати свою конфігурацію, або конформаційний стан під дією внутрішніх міжмолекулярних сил. Це можуть бути такі рушійні сили, як водневі зв'язки, сольвофобні, гідрофобні ефекти, π взаємодії, стеричні або дисперсійні взаємодії [10].

Молекулярні пінцети

Молекулярний пінцет та молекулярні кліпси - це молекули "хости" з відкритими порожнинами, здатними зв'язувати гостьові молекули. Відкрита порожнина молекулярного пінцета може пов'язувати гостей за допомогою нековалентного зв'язку, що включає водневий зв'язок, координацію металів, гідрофобні сили, сили Ван-дер-Ваальса, π — π -взаємодії та/або електростатичні ефекти [11]. Ці комплекси є підмножиною макроциклічних молекулярних рецепторів і їх функціонування забезпечується тим, що дві "руки які пов'язують гостьову молекулу між собою, з'єднані лише на одному кінці, що призводить до певної гнучкості цих рецепторних молекул (індукована модель пристосування) [12].

Молекулярні сенсори

Молекулярний сенсор або хемосенсор - це молекулярна структура, яка використовується для зондування аналіту для отримання виявленої зміни або сигналу. Дія хемосенсора залежить від взаємодії, що відбувається на молекулярному рівні і як правило, включає постійний моніторинг активності хімічних видів у даних матрицях, таких як, розчин, повітря, кров, тканини, стічні води, питна вода тощо. Застосування хемосенсорів називають хемосенсингом, який є формою молекулярного розпізнавання. Всі хемосенсиори розроблені для того, щоб містити сигнальний

фрагмент та фрагмент розпізнавання, які підключені або безпосередньо один до одного, або через якийсь роз'єм або розпірний елемент. Сигналом часто є електромагнітним випромінюванням на оптичній основі, що спричиняє зміни будь-якого (або обох) ультрафіолетового та видимого поглинання або емісійних властивостей датчиків. Хемосенсиори також можуть мати електрохімічну основу. Малі молекулярні датчики пов'язані з хемосенсиорами. Однак їх традиційно вважають структурно простими молекулами і зазначають необхідність утворення хелатуючих молекул для комплексоутворення іонів в аналітичній хімії. Хемосенсиори - це синтетичні аналоги біосенсиорів, різниця полягає в тому, що біосенсиори включають біологічні рецептори, такі як антитіла, аптамери або великі біополімери [13].

Молекулярні логічні затвори

Молекулярно-логічні затвори - це молекули, які виконують логічну операцію, що виконується на одному, або декількох фізичних, або хімічних входах і одному виході. Гальзь пройшла шлях від простих логічних систем, заснованих на одному хімічному або фізичному вході, до молекул, здатних до комбінаторних та послідовних операцій, таких як: арифметичні операції (обчислення) та алгоритми зберігання пам'яті [14].

Молекулярні асемблери

Молекулярний асемблер - це молекулярна структура, здатна керувати хімічними реакціями шляхом позиціонування реакційноздатних молекул з атомною точністю". Деякі біологічні молекули, такі як рибосоми, відповідають цьому визначенню. Вони отримують вказівки від РНК-месенджера, а потім збирають специфічні послідовності амінокислот для побудови білкових молекул. Однак термін "молекулярний асемблер" зазвичай відноситься до теоретичних пристроїв, створених людиною [15].

Молекулярні шарніри

Молекулярний шарнір являє собою молекулу, яку можна вибірково перемикаєти з однієї конфігурації в іншу оборотним чином [16].

Поки що найскладнішими молекулярними машинами є біологічні, тобто ті, що, знаходяться всередині живих клітин. Зазвичай це мультипротеїнові комплекси. Деякі з них виступають у ролі моторів, наприклад, міозин, який відповідає за скорочення

м'язів, або кінезин, який переміщує вантажі по мікротрубкам. Деякі молекулярні машини відповідають за генерацію енергії з використанням протонних градієнтів. Не можна забувати про машини, що відповідають за експресію генів. Всі біологічні молекулярні машини є набагато складнішими, ніж будь-які молекулярні машини побудовані штучно.

Оскільки це відносно нова галузь, дослідження молекулярних машин проводяться у різноманітних напрямках. Розробляються різні математичні моделі для їх опису. Перевіряється правильність теоретичних висновків отриманих раніше [17], [18]. Досліджуються теоретичні границі швидкості, з якою сенсори можуть отримувати інформацію [19] і як додаткова пам'ять може вплинути на характеристики таких сенсорів [20]. Також досліджується інформаційний потік молекулярних машин та його зв'язок із Демоном Максвелла [21].

1.2. Базові величини, що використовуються у дослідженні

Кожна фізична конструкція (наприклад молекулярна машина, чи сенсор), що здатна змінювати свій стан самостійно, чи під дією середовища має доступ до інформації про свій поточний стан (за умови, що ми розглядаємо конструкцію без пам'яті). Інформацію про свій поточний стан можна використати, щоб дізнатись поточний стан середовища, або стан у якому середовище опиниться через певну кількість часу.

Незважаючи на те, що біологічні молекулярні машини діють далеко від своїх положень рівноваги, фундаментальні дослідження необхідно проводити на машинах, що перебувають із середовищем у рівновазі. Таким чином можна сфокусуватись на основному питанні цієї роботи, наскільки ефективно молекулярна машина може отримувати інформацію про оточуюче середовище. На даний момент схожі дослідження проводяться нашими американськими та канадськими колегами.

В даній роботі фігуруватимуть методи та теореми з стохастичної термодинаміки [22], а також деякі поняття і величини з теорії інформації та статистичної фізики [23].

Оскільки теорія інформації не є стандартним інструментом фізики, необхідно зупинитись на ній детальніше. Вона вивчає кількісну оцінку зберігання інформації її передачу. Розвиток теорії інформації почався з того, що у 1948 році Клод Шеннон у важливій статті під назвою "Математична теорія зв'язку" запропонував знайти фундаментальні обмеження на обробку сигналів та операції зв'язку, такі, як стиснення даних.

Основною величиною в теорії інформації є ентропія. Ентропія, як і у фізиці, кількісно визначає величину невизначеності, пов'язану із випадковою величиною, або випадковим процесом. Наприклад, виявлення результату справедливого підкидання монети (з двома однаково ймовірними результатами) дає менше інформації (нижча ентропія), ніж визначення результату з кидання кубика (з шістьма однаково ймовірними результатами). Також для цілей даної роботи використовуватимуться такі величини, як: взаємна інформація, умовна і відносна ентропія.

Основна величина - ентропія - кількісна міра невизначеності випадкової величини або процесу:

$$H(X) = - \sum_x p(x) \log_2(p(x)) \quad (1.1)$$

ентропія для двох пов'язаних змінних:

$$H(X) = - \sum_x \sum_y p(x, y) \log_2(p(x, y)) \quad (1.2)$$

умовна ентропія - ентропія у випадку, коли одна із пов'язаних випадкових змінних визначена:

$$H(X) = - \sum_x \sum_y p(x, y) \log_2(p(x|y)) \quad (1.3)$$

інформація - кількісна міра ентропії, що зникає, при визначенні однієї із пов'язаних випадкових змінних.

$$I = - \sum_{x,y} p(x,y) \log_2 \frac{p(x,y)}{p(x)p(y)} = H(Y) - H(X,Y) \quad (1.4)$$

де $p(x, y)$ - це спільний розподіл імовірностей для випадкових величин x та y . При використанні основи 2, тобто \log_2 , пораховану інформацію можна вимірювати у бітах.

1.3. Огляд вихідної моделі та її симуляції

Вихідна модель мала за мету дослідження нерівності [17]:

$$I_{mem} - I_{pred} \leq \beta < W_{diss} \quad (1.5)$$

В процесі дослідження було введено величину, *varphi*.

$$\varphi = \frac{I_{mem} - I_{pred}}{W_{diss}} \quad (1.6)$$

Параметризація моделі дозволяє ввести середню швидкість машини.

$$k_{sys} = \frac{2}{\frac{1}{k_+} + \frac{1}{k_-}} \quad (1.7)$$

Основним варіюємим параметром моделі 1.1 виступає енергія переходу між станами машини ΔE .

$$\beta \Delta E = \log \frac{k_+}{k_-} \quad (1.8)$$

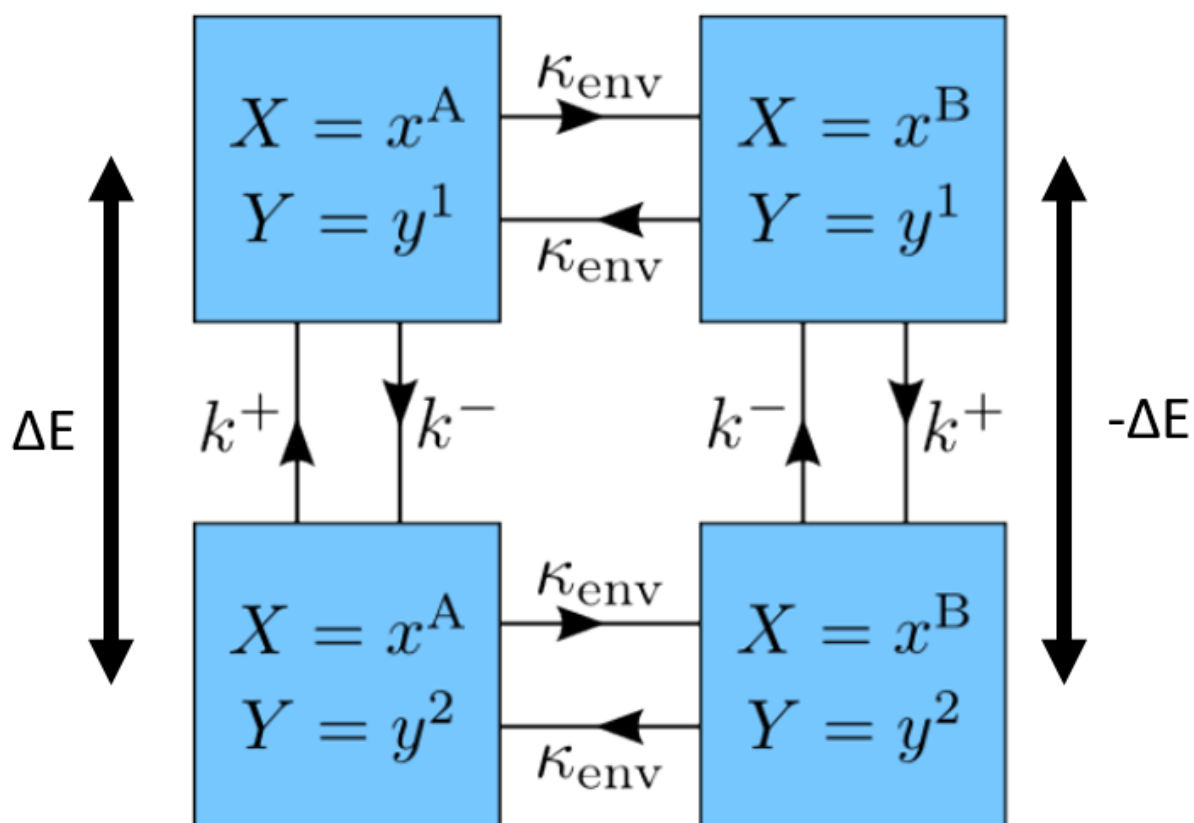


Рис. 1.1. Вихідна параметризація

1.4. Висновки до розділу 1

Наведено огляд молекулярних машин, продемонстровано актуальність даної роботи. Описано величини, що використовуються у дослідженні та модель-прототип, яку використано за основу для моделі авторів.

РОЗДІЛ 2.

РОЗРОБКА СИМУЛЯЦІЇ МОЛЕКУЛЯРНОЇ МАШИНИ ТА ЇЇ ДОСЛІДЖЕННЯ

2.1. Запропонована модель та досліджувані величини

Практично завжди перед конструюванням тестового зразка, або прототипу, проводяться симуляції, оскільки це дешевше і простіше. Але такий підхід потребує вибору коректної моделі в рамках поставлених задач. Для даної роботи модель повинна описувати максимально узагальнені (але в певних межах) властивості молекулярної машини, бути достатньо гнучкою, забезпечувати зв'язок між середовищем і машиною. Запропонована авторами модель цим вимогам задовольняє [24].

Модель складається з двох складових: середовища (X) і машини (системи) (Y). Кожна з компонент має кілька станів (у цьому дослідженні два) (A, B), в яких вона може перебувати з певною імовірністю. (Очевидно, що сумарна імовірність перебування хоча б у одному з цих станів дорівнює 1, $X_A + X_B = Y_A + Y_B = 1$). Стан, у якому в кінці певного кроку перебуває машина (Y_A, Y_B) залежить від стану середовища (X_A, X_B), завдяки імовірностям переходів між станами машини (a, b) та (c, d), які обираються в залежності від стану середовища 2.1. Щоб отримати швидкості переходів необхідно імовірності переходів розділити на час одного кроку.

Для даної параметризації також можна ввести середню швидкість середовища K_{env} . Середня швидкість середовища визначає за скільки кроків симуляції середовище досягне стану рівноваги (точки рівноваги). Фінальний розподіл, тобто розподіл у точці рівноваги задається в симуляції чисельно. Також можна задати початковий розподіл. Щоб отримати кілька шляхів, якими середовище приходить до фінального стану.

Запропонована авторами схема ґрунтується на моделі, наведеній у [17]. У цій моделі енергії переходу між станами машини (Y), міняються місцями при різних станах середовища (X). В рамках нової моделі, це описується наступним рівнянням:

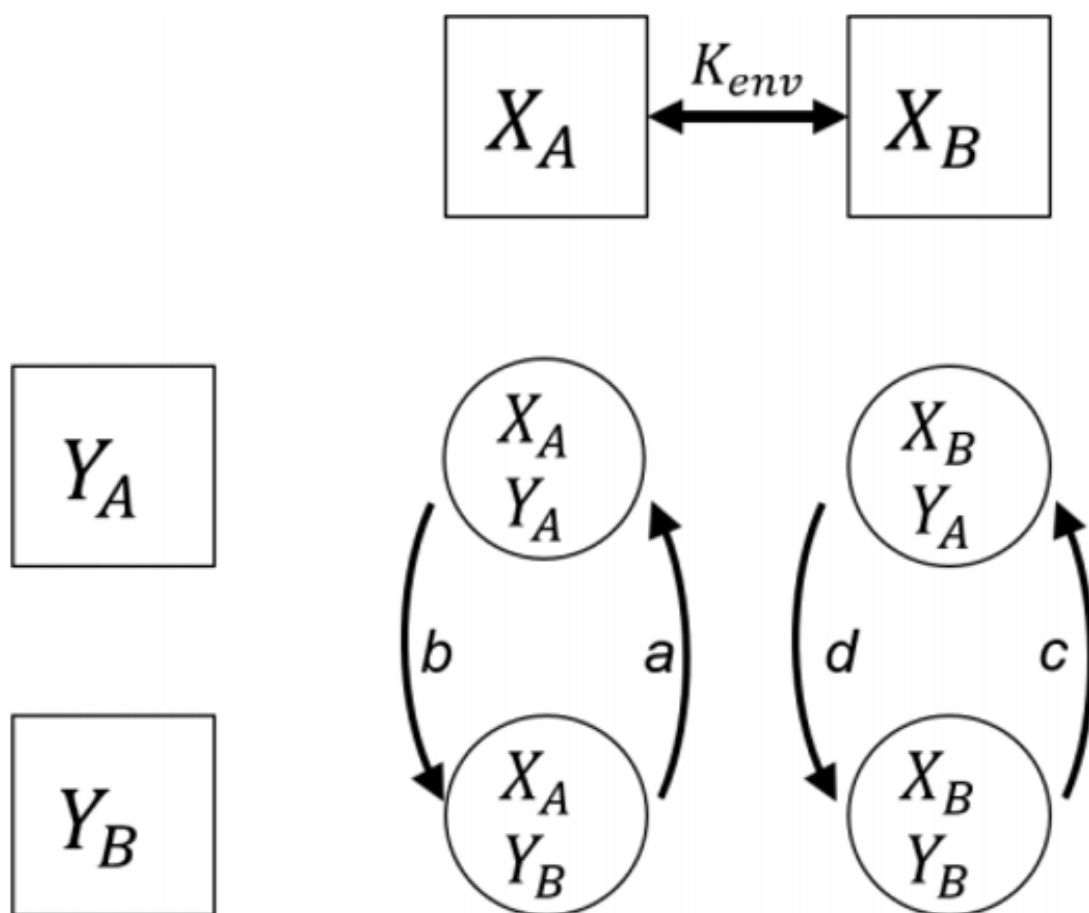


Рис. 2.1. Вихідна параметризація

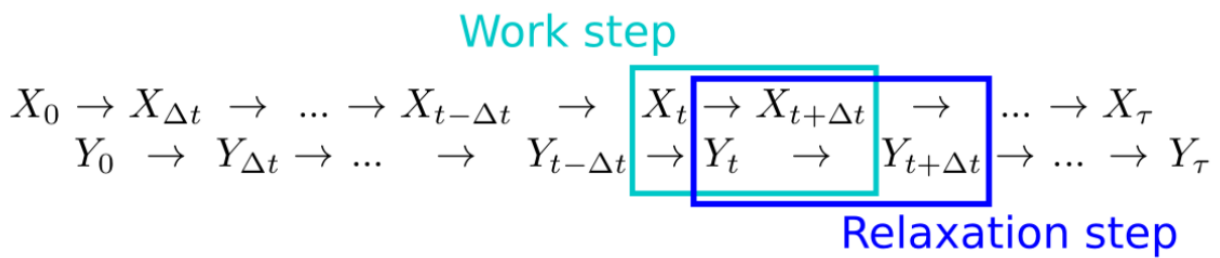


Рис. 2.2. Вихідна параметризація

$a = c, b = d$. Авторами було ”знято це обмеження шляхом додавання двох степенів вільності. Таким чином, отримано можливість варіювати всі імовірності переходів (a, b, c, d) .

У симуляція використовуються Марківські процеси, де імовірності переходів між станами з часом не змінюються.

Один крок симуляції розбивається на два: робочий та релаксаційний. Під час робочого кроку застосовується матриця переходів середовища і змінюється розподіл імовірностей середовища (X). Під час релаксаційного - застосовується аналогічна матриця, але для машини і змінюється розподіл імовірностей машини (Y) 2.2.

Інформацію можна рахувати між будь-якими парами випадкових величин, але в межах цієї роботи, необхідними є лише дві комбінації:

Перша, I_{mem} – інформація між поточним станом машини та поточним станом середовища. Ця інформація показує наскільки добре машина ”зчитує” поточний стан середовища.

$$I_{mem} = I(Y_t, X_t) \quad (2.1)$$

Друга, I_{pred} – інформація між поточним станом машини та наступним станом середовища. Ця інформація є показником наскільки добре машина може передбачити наступний стан середовища.

$$I_{pred} = I(Y_t, X_{t+1}) \quad (2.2)$$

Дослідницькою групою д. Сівака для зручності було введено інформаційну величину - Ностальгію. Ностальгія, за Сіваком, – це кількість некорисної інформації:

$$I_{nos} = I_{mem} - I_{pred} \quad (2.3)$$

Вся робота можлива завдяки наступній тотожності, доведеній групою д. Сівака, використовуючи "Ностальгію". Тотожність показує зв'язок між Інформацією та розсіяною енергією. Таким чином встановлюється зв'язок між величинами з теорії інформації та термодинамічними величинами.

$$\beta < W_{diss} > = I_{nos} - \beta < \Delta F_{neq}^{relax} > \quad (2.4)$$

З вищенаведеного рівняння можна обрахувати витрати енергії. Обернену температуру β , що фігурує в рівнянні (2.4) взято за одиницю. ΔF_{neq}^{relax} тут - зміна вільної нерівноважної енергії Гельмгольца за релаксаційний крок. З рівняння випливає, що ностальгія завжди менша, ніж розсіяна енергія. Ця нерівність підтверджується у інших дослідників з команди Сівака.

В межах цієї роботи необхідно розглядати не ностальгію, а інформацію, яка відповідає зв'язку між поточним станом системи і майбутнім станом середовища I_{pred} і те, яким чином її збільшити, затрачаючи мінімальну кількість енергії W_{diss} . Іншими словами, необхідно розглядати інформаційну ефективність η та спосіб її збільшити.

$$\eta = \frac{I_{pred}}{W_{diss}} \quad (2.5)$$

Щоб обрахувати витрати енергії необхідно також обчислити F_{neq}^{relax} . Це нерівноважна енергія Гельмгольца. Її можна розбити на дві частини: рівноважну і додаткову.

$$F_{neq}[p(Y_t|X_t)] = F_t + F_t^{add}[p(Y_t|X_t)] \quad (2.6)$$

Додаткова вільна енергія є функцією спільного розподілу, а вільна енергія за релаксаційний крок не змінюється і її різниця дорівнює нулю.

$$F_{neq}^{relax}[X_t : Y_{t-1} \rightarrow Y_t] = F_t^{add}[p(Y_t|X_t)] - F_t^{add}[p(Y_{t-1}|X_t)] \quad (2.7)$$

Додаткова енергія Гельмгольца обчислюється з розходження Кульбака-Лейбнера.

$$F_t^{add} = k_B T D_{KL}[p(Y_t|X_t) || p_{eq}(Y_t|X_t)], \quad (2.8)$$

$$\text{де } D_{KL}[p(x) || q(x)] = \left\langle \ln \left[\frac{p(x)}{q(x)} \right] \right\rangle_{p(x)} \quad (2.9)$$

Це є всі величини, необхідні для оцінки інформаційної ефективності η .

2.2. Опис роботи симуляції

Всі обрахунки і дослідження було проведено використовуючи мову програмування python. Вихідними даними була симуляція групи Д. Сівака. На її основі було написано новий код, що запезпечує більшу кількість варіюємих параметрів і можливість збільшення можливих станів. Для контролю версій використовувався сервіс github. До роботи додана лише остання версія коду, функціонал якого не дозволяє отримати всі результати, оскільки деякі з них отримано на старших версіях.

Математичну модель можна представити у вигляді квадратної матриці 2.3 з чотирма елементами, кожен з яких відповідає за суміжний стан середовища і машини з імовірностями переходів між ними.

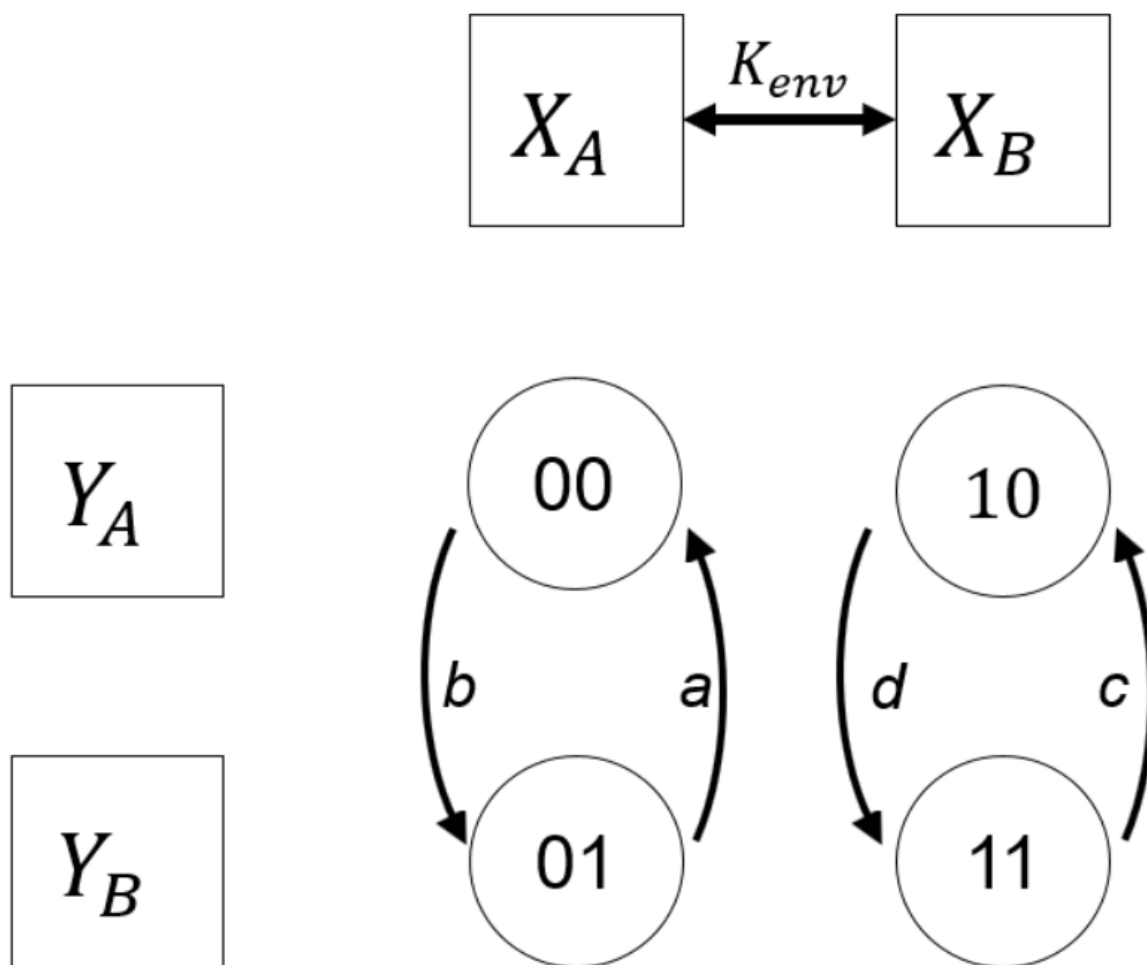


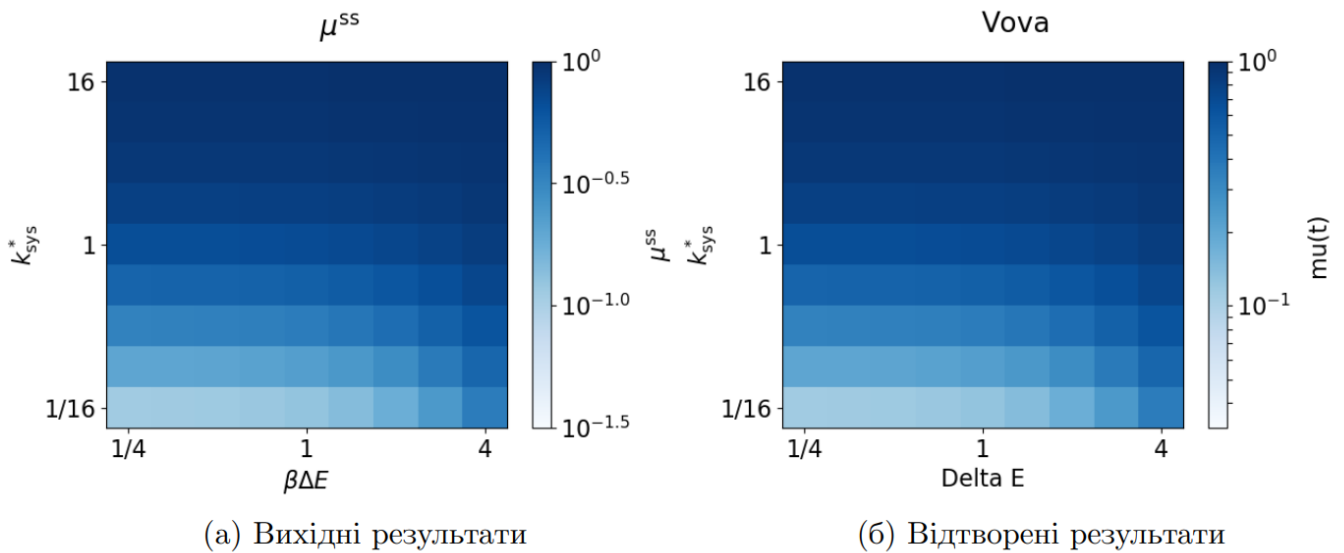
Рис. 2.3. Матриця станів системи

Ця модель не включає в себе можливість машини пам'ятати свої попередні стани. Моделі з пам'яттю створені для вивчення інших величин. Якщо виникне потреба у такому дослідженні наведену модель можна модифікувати, додавши пам'ять машини.

2.3. Відтворення наявних результатів

Оскільки було забезпечено зворотній зв'язок, шляхом накладання двох обмежень на імовірності переходу, можна відтворити модель-прототип і повторити дослідження колег.

Було відтворено 2.4 основну досліджувану величину φ з (1.6).

Рис. 2.4. Відтворення теплової карти μ

Також відтворено фінальний результат дослідження 2.5. φ не може бути більше одиниці, Чорна нижня лінія показує насичення нерівності.

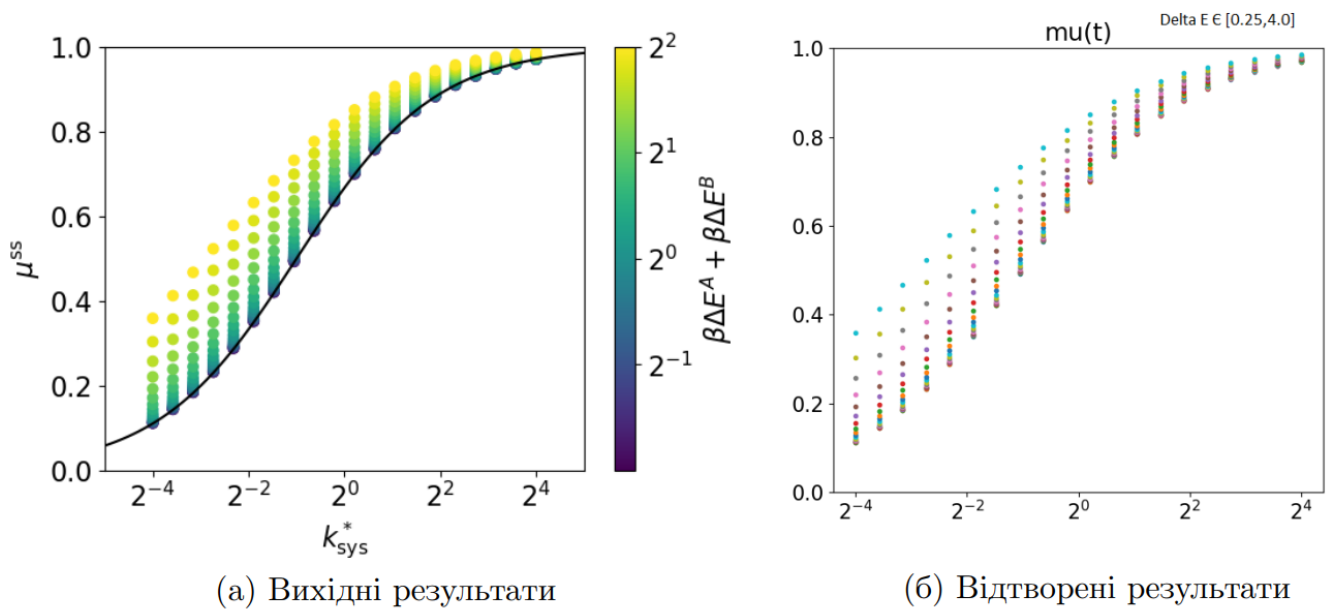


Рис. 2.5. Відтворення підтвердження нерівності

2.4. Дослідження границь роботи симуляції

Було досліджено зв'язок машини і середовища шляхом вимкнення робочих кроків 2.6. Зв'язок та його вплив на вільну енергію було знайдено.

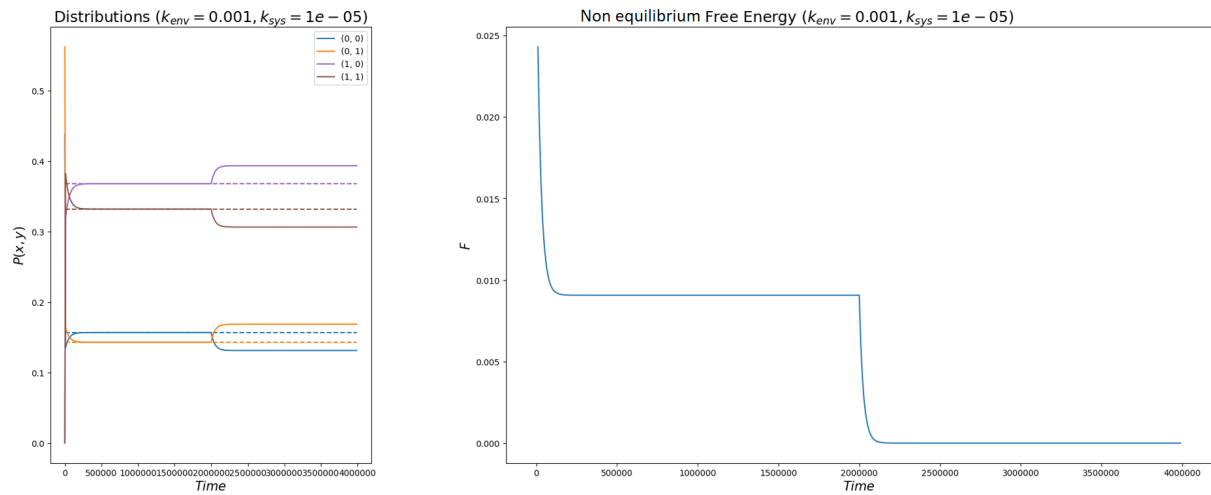


Рис. 2.6. Динаміка розподілу імовірностей при маніпуляціях з робочим кроком

У 2.7 змінився лише спільний розподіл, але відособлені розподіли не змінилися. Це якраз пояснюється наявністю зв'язку, який було розірвано.

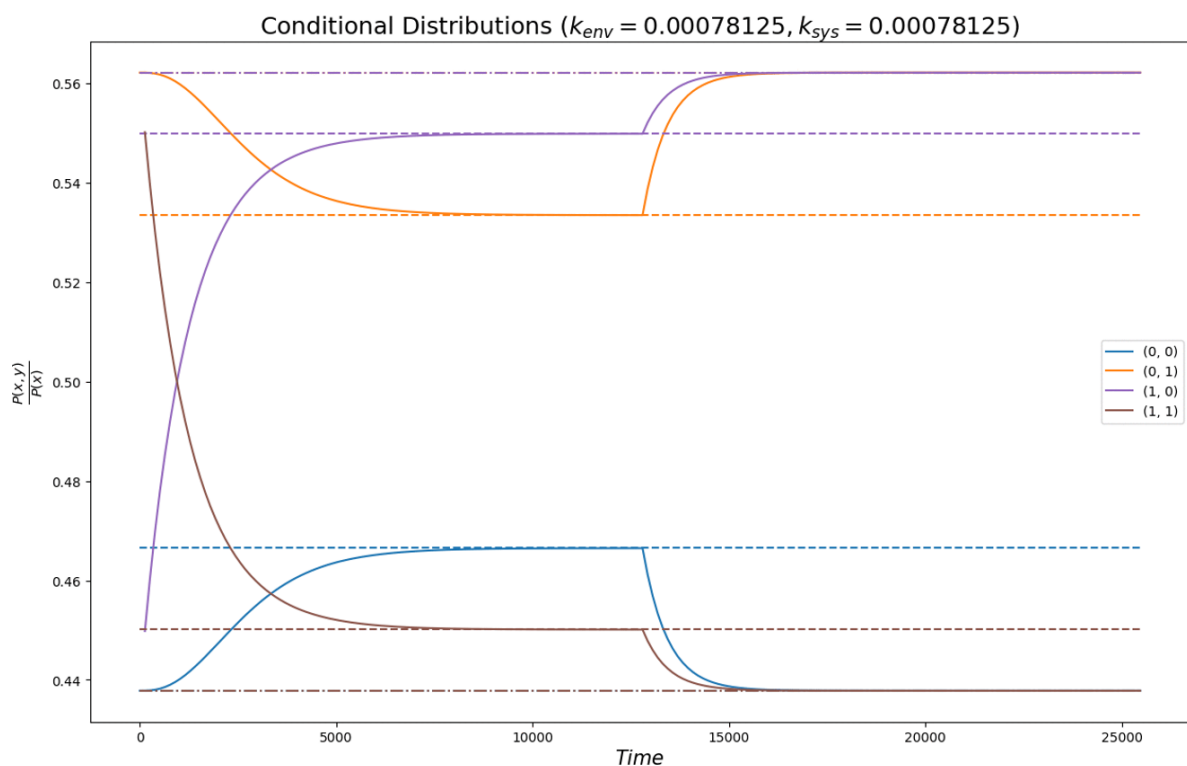


Рис. 2.7. Динаміка умовного розподілу імовірностей при маніпуляціях з робочим кроком

Досліджено більше детально зміну вільної енергії машини 2.8:

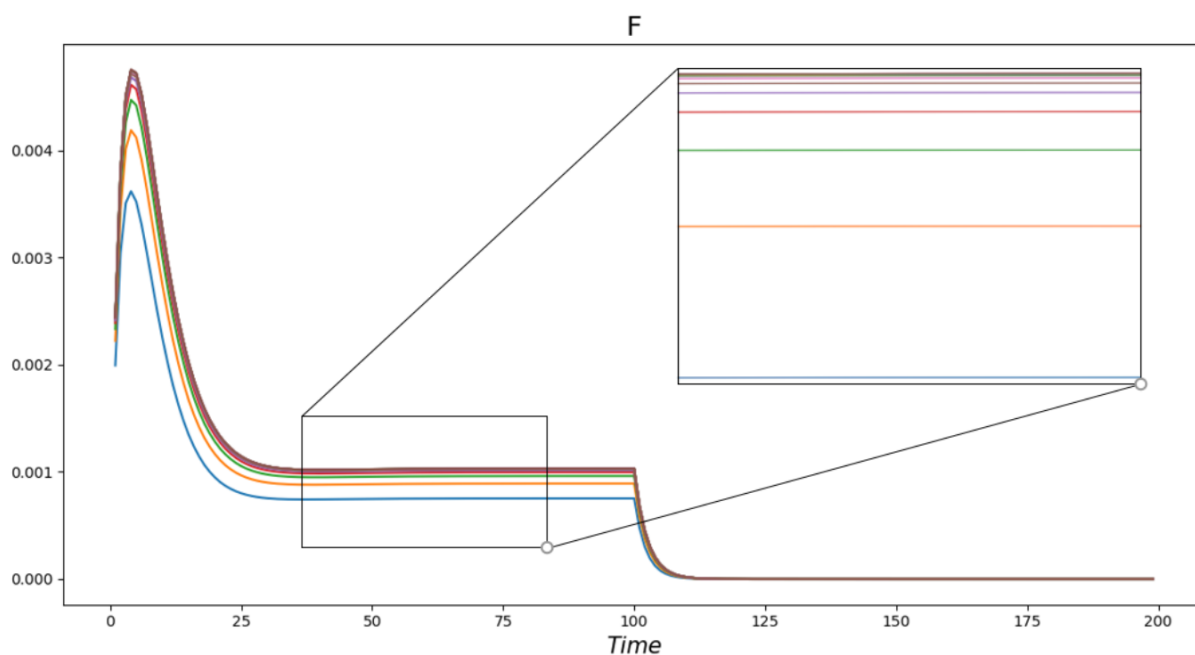


Рис. 2.8. Зміна Вільної енергії при зміні кроку симуляції

Також було досліджено вплив дискретизації симуляції на її точність та межі в яких похибкою дискретизації можна знехтувати.

2.5. Висновки до розділу 2

Описано зв'язок між теорією інформації та стохастичною Термодинамікою. Пояснено принцип функціонування моделі авторів та симуляції. Досліджено коректність її роботи та границі застосування. Відтворено результати досліджень іншої групи.

РОЗДІЛ 3.

АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1. Розсіяна енергія

На одному графіку зафіксовано по дві імовірності переходів а по осям відкладається відношення між імовірностями переходів.

Перший отриманий результат тривіальний 3.1 - мінімальна енергія відповідає мінімальному зв'язку системи і середовища.

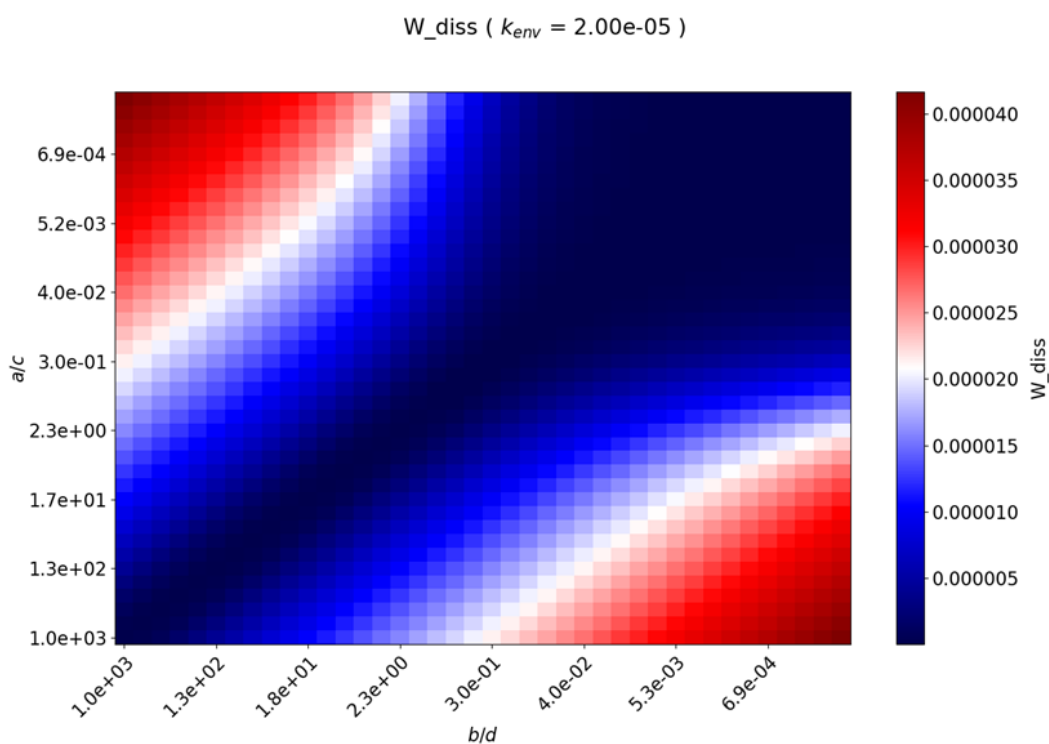


Рис. 3.1. Залежність розсіяної енергії від імовірностей переходів

3.2. Взаємна Інформації

Результати аналогічного дослідження для I_{pred} наведено на 3.2

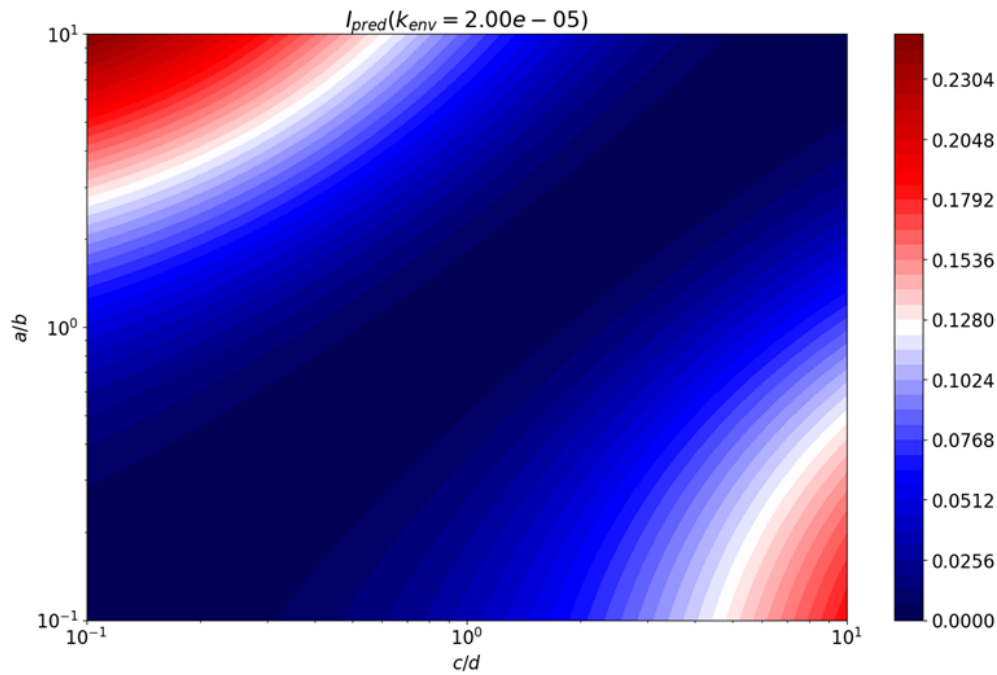


Рис. 3.2. Залежність інформації від варіюємих параметрів

Видно що при простій мінімізації затраченої енергії отримана інформація також зменшується. Також було з'ясовано що необхідно внести поправку у визначення інформаційної ефективності.

$$\eta = \frac{I_{pred}}{W_{dissn}} \quad (3.1)$$

$$\text{де } W_{diss} = W_{diss}/K_{env} \quad (3.2)$$

3.3. Інформаційна ефективність

З наступних графіків видно що проста мінімізація енергії не єдиний спосіб підвищення інформаційної ефективності.

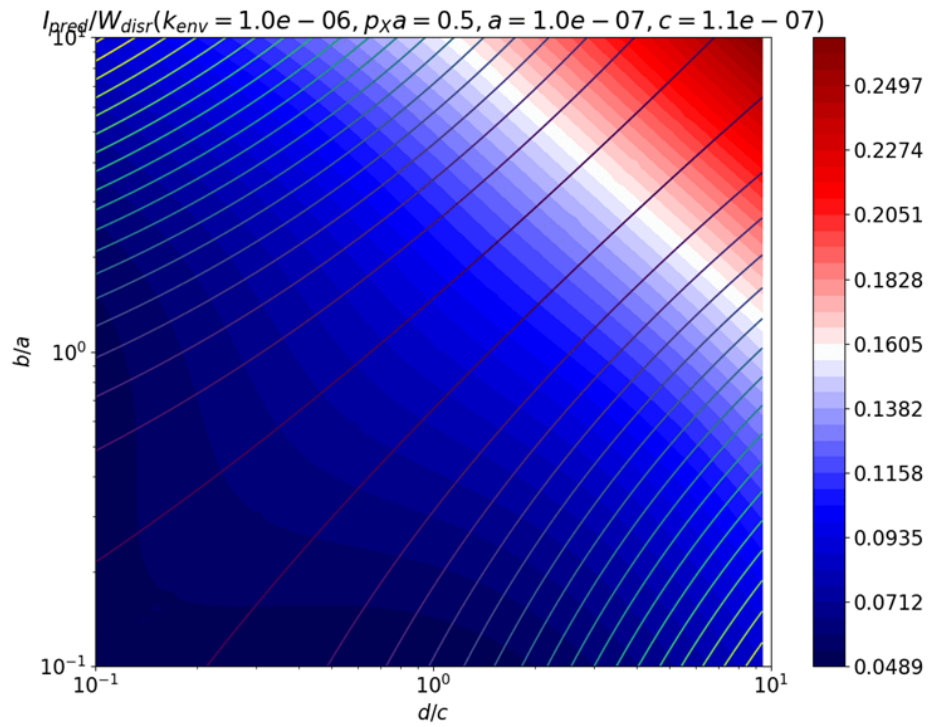


Рис. 3.3. Залежність інформаційної ефективності від імовірностей переходів

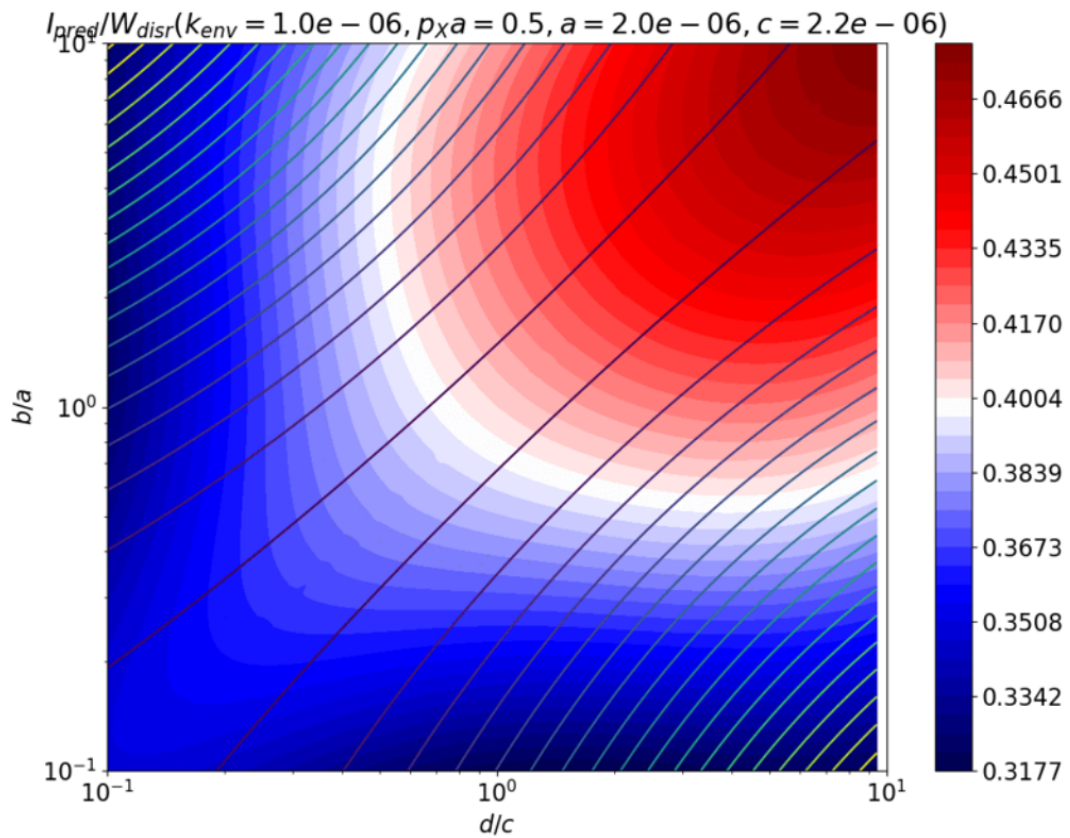


Рис. 3.4. Залежність інформаційної ефективності від імовірностей переходів(1)

Неперервні лінії на 3.3 – це ізолінії розсіяної енергії W_{diss} .

З обох рисунків видно, що для підвищення ефективності необхідно збільшувати імовірності переходів, тобто підвищувати швидкість машини.

Якщо рухатись у наміченому напрямку можна знайти зону максимальної ефективності машини для даного середовища 3.4.

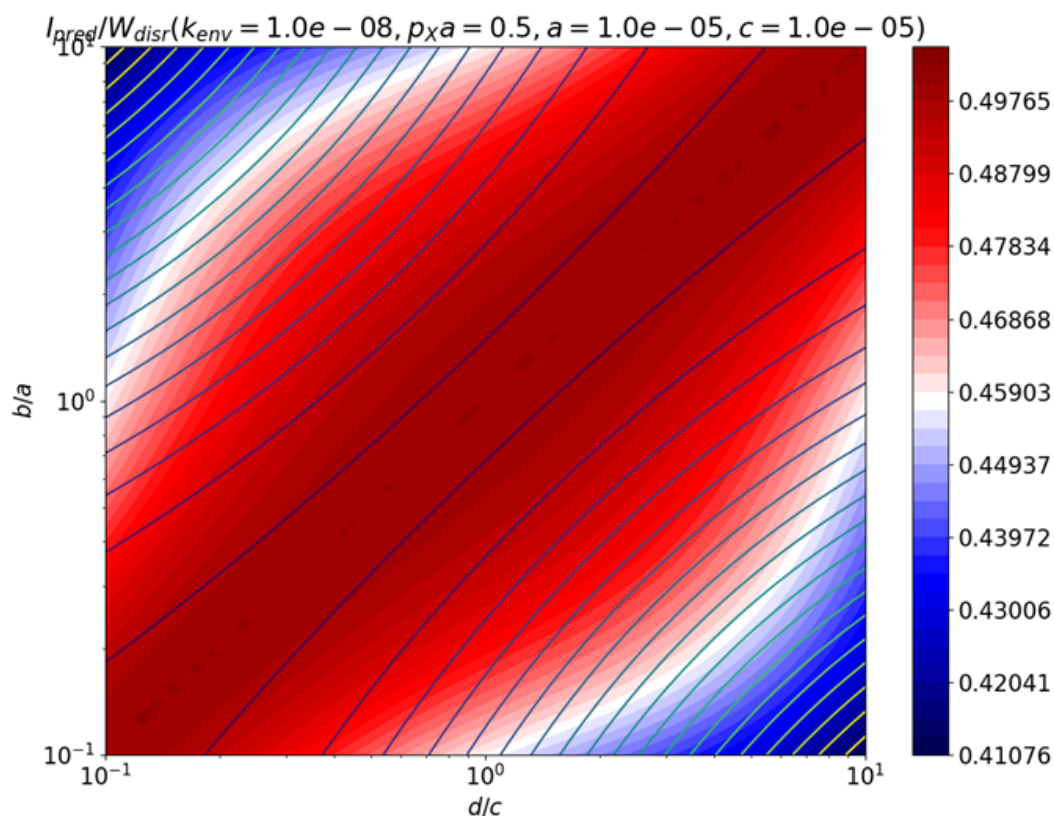


Рис. 3.5. Залежність інформаційної ефективності від збільшених імовірностей переходів

Всі вищенаведені дослідження проведені для середовища з незміщеною точкою рівноваги $p(X_A) = p(X_B)$. Якщо точку рівноваги змістити можна отримати додаткові результати.

З наступного рисунка видно, що для середовища із зміщеною точкою рівноваги є потреба підтримувати певне співвідношення між імовірностями переходів ??.

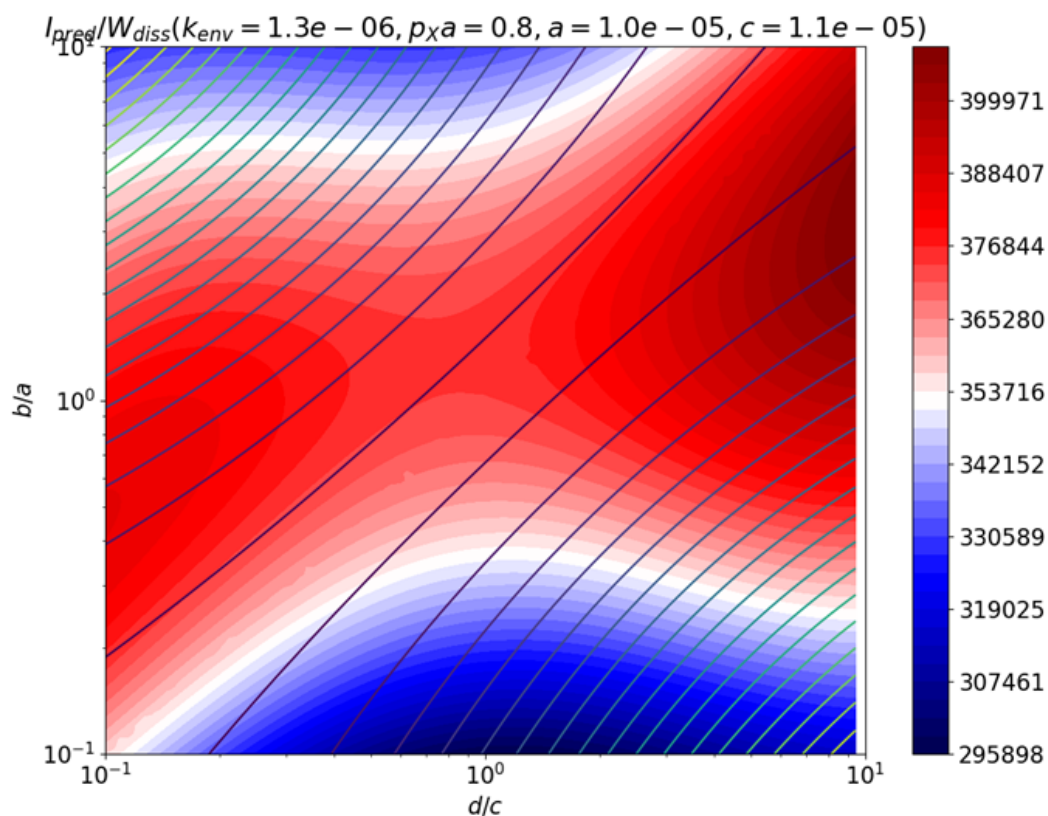


Рис. 3.6. Залежність інформаційної ефективності від збільшених імовірностей переходів для середовища із зміщеним станом рівноваги

3.4. Висновки до розділу 3

Якісно показано тривіальні та нетривіальні шляхи максимізації інформаційної ефективності. Такі як, підвищення швидкості машини або підтримання певного співвідношення між імовірностями. Ці співвідношення вказують на необхідність співставлення чітко виражених станів середовища не чітко вираженим станам машини і навпаки.

РОЗДІЛ 4.

ПОШУК ЗАКОНОМІРНОСТЕЙ

4.1. Змінне середовище

Далі для зручності температурні мапи $\frac{W_{diss}}{I_{pred}}$ називатимемо картами ефективності.

Було досліджено, як саме зміщення стану рівноваги середовища вплине на поведінку машини та досліджувані показники ефективності. З використанням сервісу Adobe Spark було візуалізовано випадки для різних $p_X a$ та k_{env} .

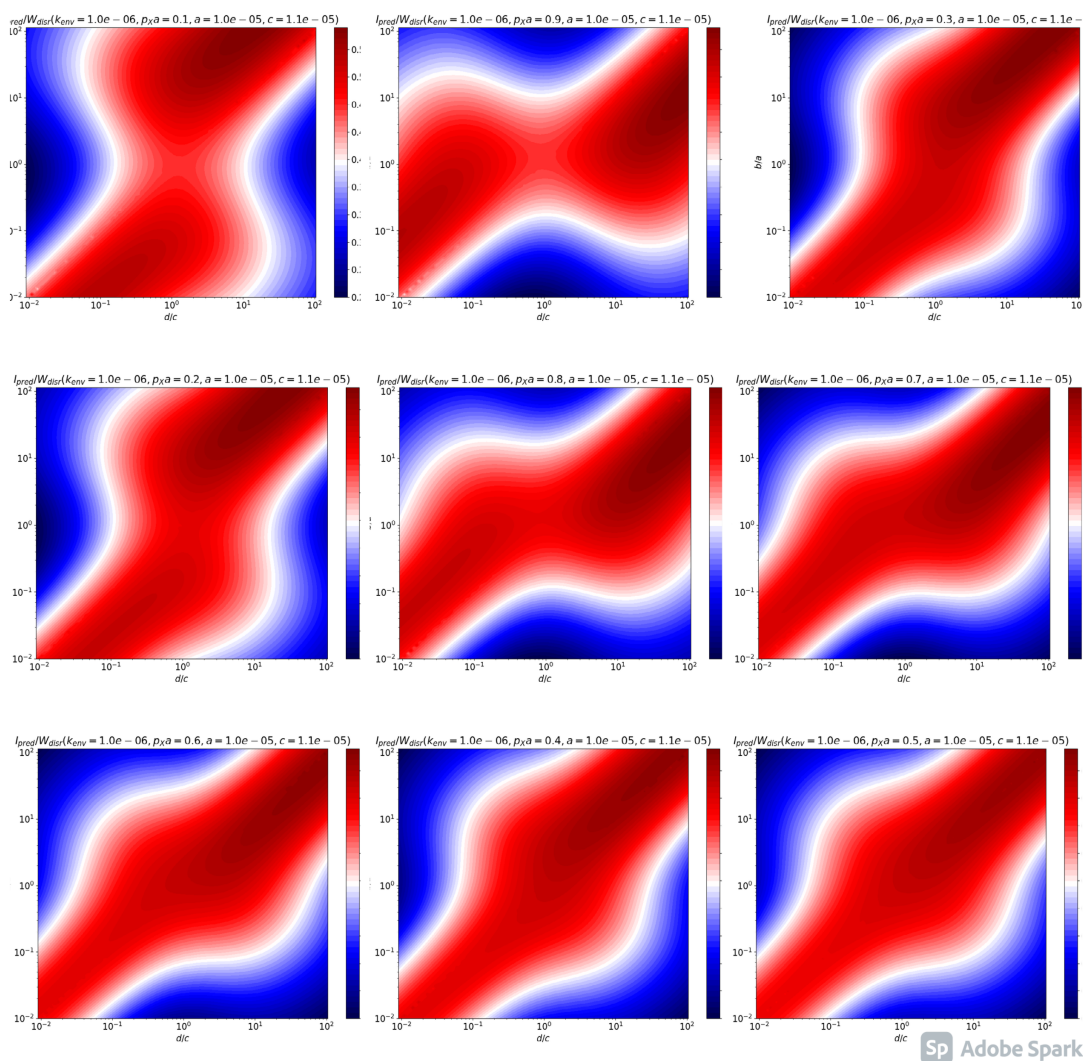


Рис. 4.1. Карта ефективності для різних $p_X a$, $k_{env} = 10^{-6}$

Наведений рисунок 4.1 є найближчим випадком до наведеного у кінці розділу 3 з параметром швидкості середовища $k_{env} = 10^{-6}$.

Також наведено варіанти для інших швидкостей середовища $k_{env} = 2 \cdot 10^{-6}$ та $k_{env} = 10^{-5}$.

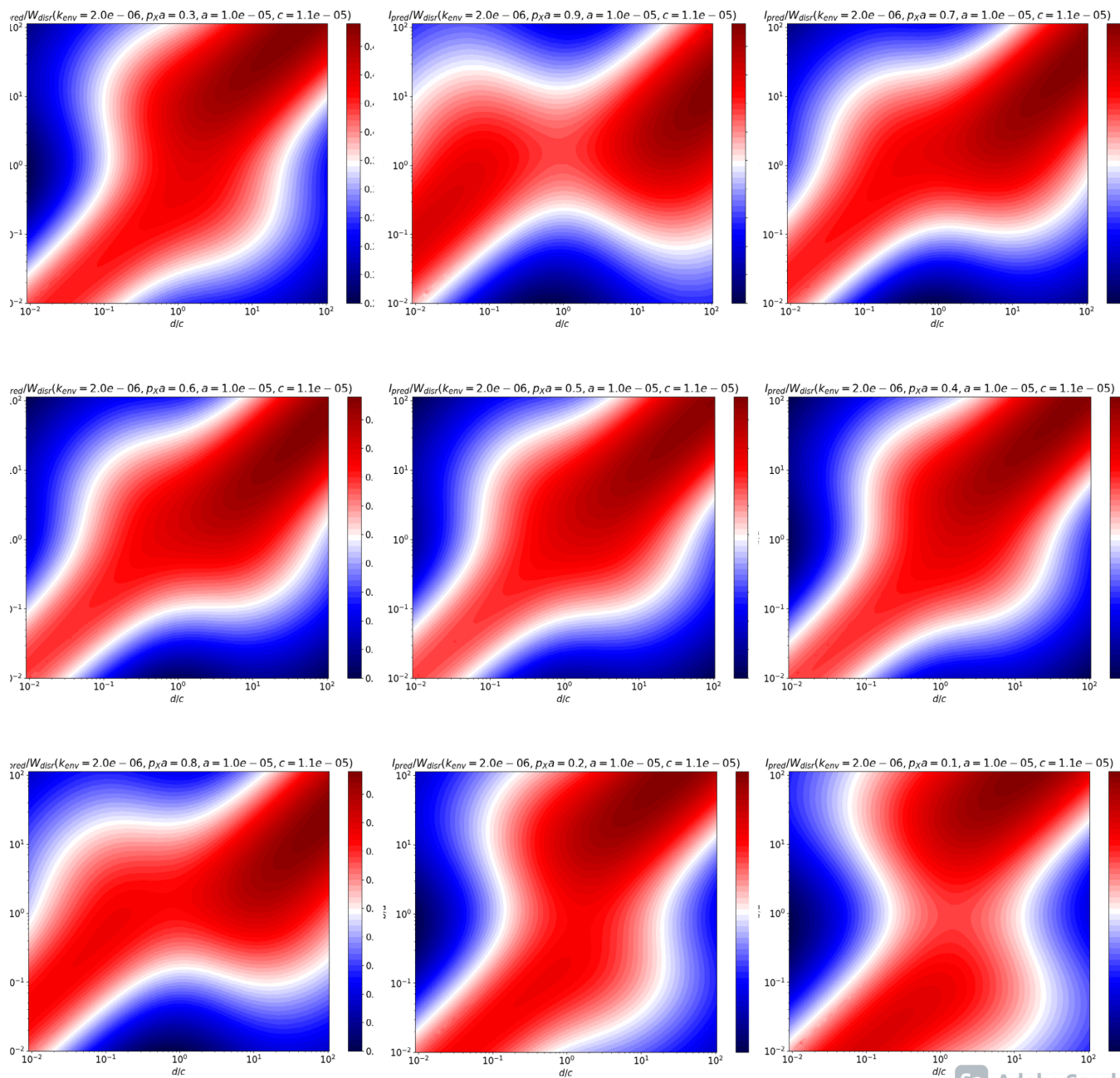


Рис. 4.2. Карта ефективності для різних $p_X a$, $k_{env} = 2 \cdot 10^{-6}$

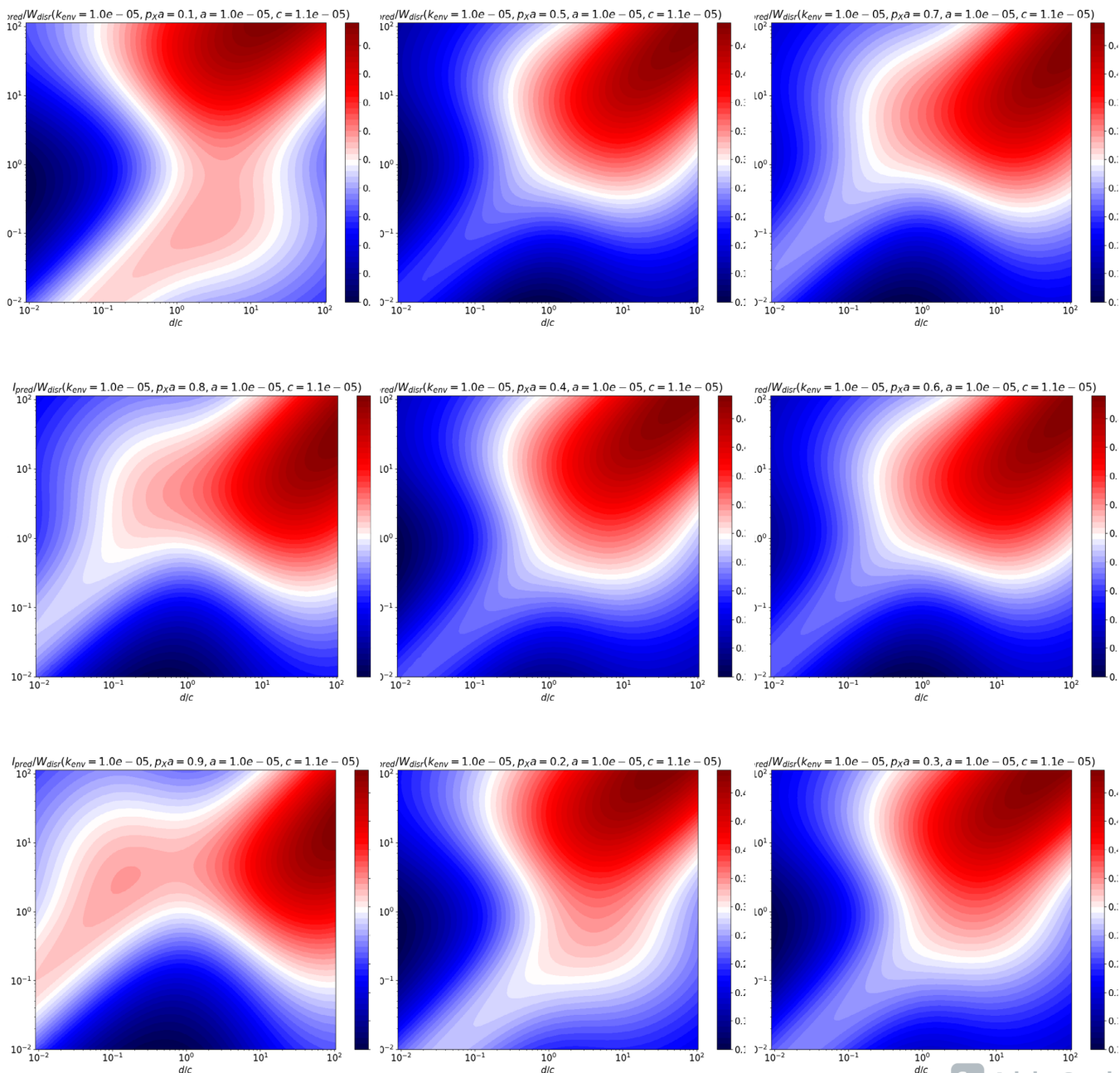


Рис. 4.3. Карта ефективності для різних $p_X a$, $k_{env} = 10^{-5}$

З наведених рисунків видно, що діапазону зміни k_{env} достатньо, оскільки для граничного випадку всі середовища зі зміщеними точками рівноваги починають вироджуватись до незміщених. Тобто зникає необхідність підтримувати певне співвідношення між імовірностями переходів і з'являється необхідність пришвидшити систему в цілому.

Також видно, що співвідношення кардинально відрізняються для різних $p_X a$, та, що вони є симетричними відносно незміщеного стану і мають лінійний вигляд на логарифмічному графіку.

Далі у демонстраційних цілях наведено карту ефективності для одного випадку $p_X a = 0.9$, $k_{env} = 10^{-5}$ у збільшеному варіанті 4.4 та її складові W_{diss} та I_{pred} .

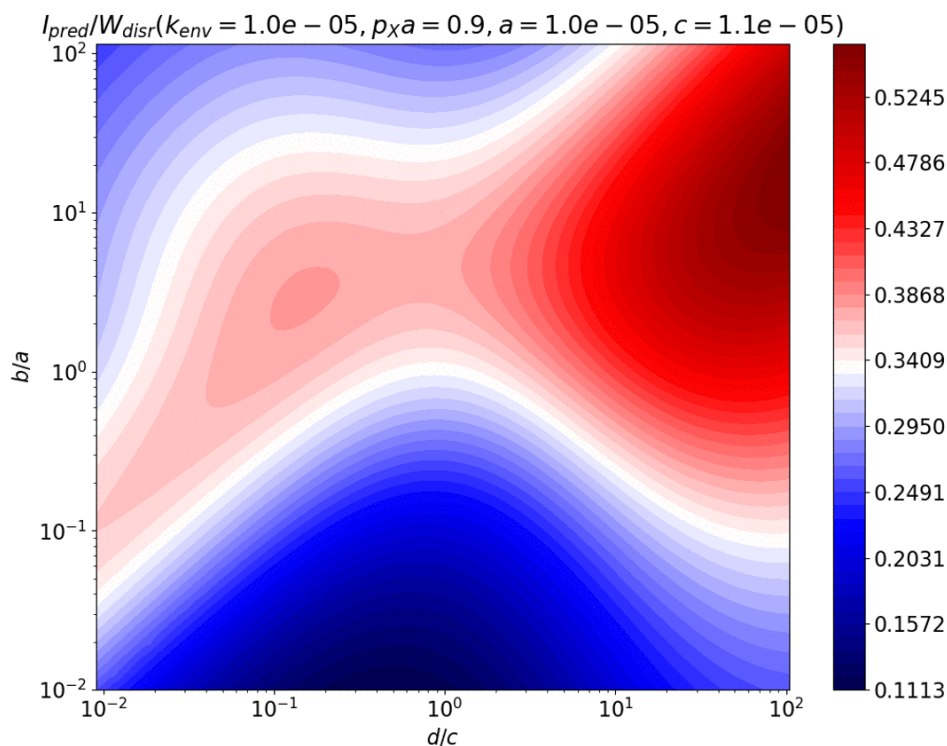


Рис. 4.4. Карта ефективності $p_X a = 0.9$, $k_{env} = 10^{-5}$

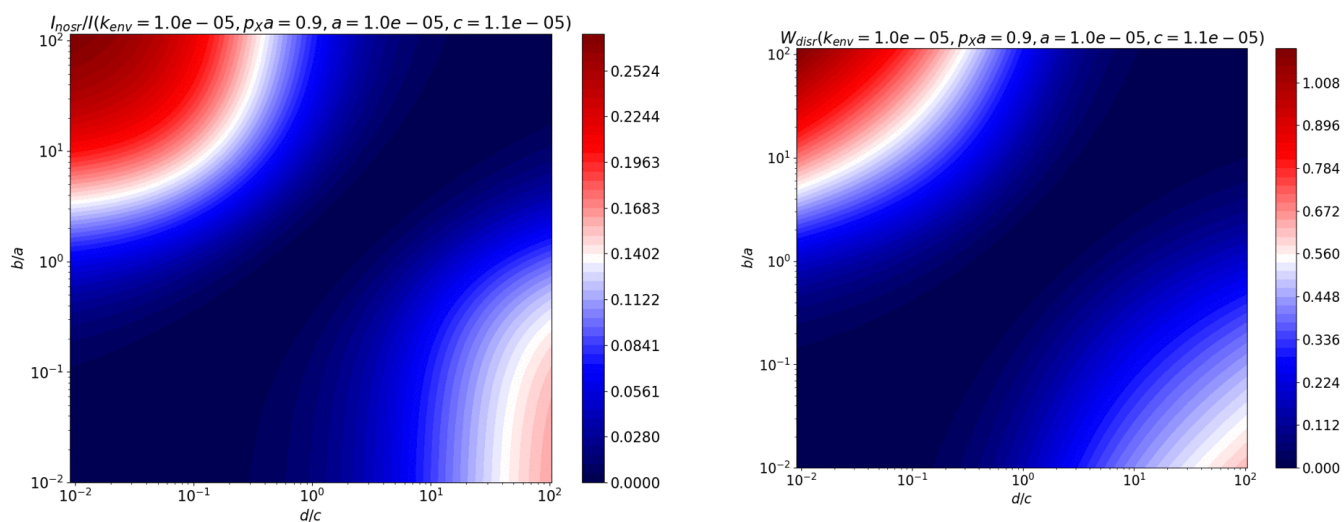


Рис. 4.5. Складові інформаційної ефективності

Наступним логічним кроком є пошук конкретних аналітичних залежностей. Скористаємось візуальним аналізом, щоб оцінити вид залежності. Як приклад на 4.6 наведено залежність виду $y = x + a$. Очевидно що на логарифмічному графіку це не пряма, а на лінійному це пряма, але така, що не задовольняє збільшенню ефективності найшвидшим шляхом.

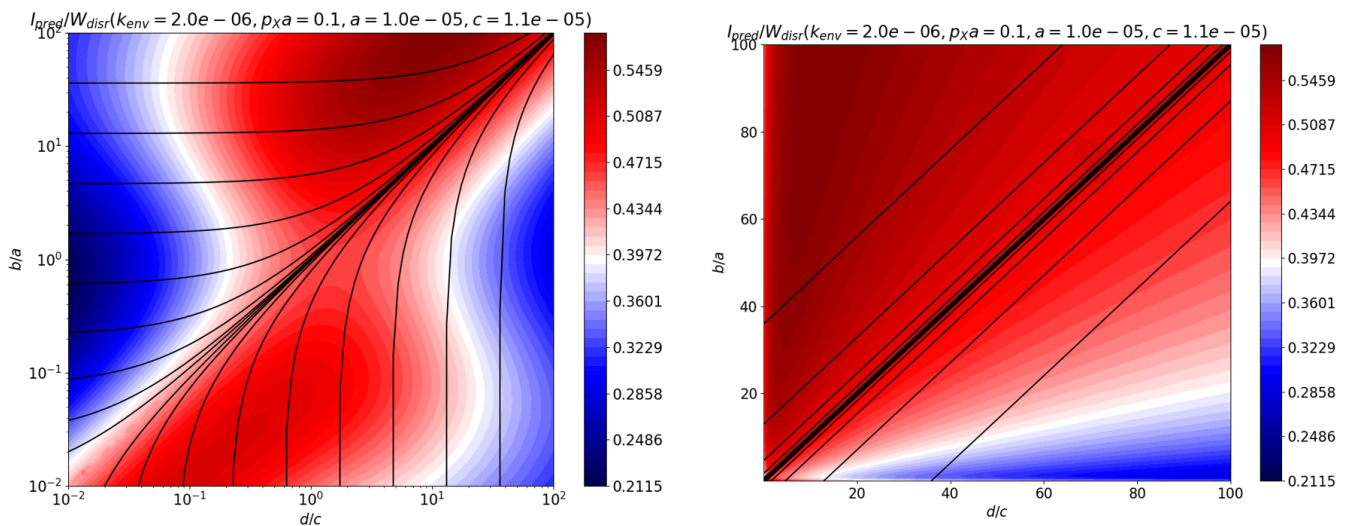


Рис. 4.6. Різниця між логарифмічним та нелогарифмічним представленням

4.2. Пошук оптимальних траєкторій

Опираючись на наведені вище міркування було обрано залежність виду $y = a \cdot x$. Коефіцієнт зсуву було опущено, оскільки не було знайдено фізичного обґрунтування для існування такого коефіцієнту. Підтримання певного співвідношення у свою чергу означає підтримання постійної енергії переходу.

Для кожного положення зміщеної точки рівноваги для двох швидкостей середовища $k_{env} = 2 \cdot 10^{-6}$, $k_{env} = 10^{-5}$ було знайдено пряму (чорна лінія) що забезпечує найбільший приріст інформаційної ефективності під час руху вздовж неї.

Спершу було знайдено оптимальні траєкторії для випадку $k_{env} = 2 \cdot 10^{-6}$.

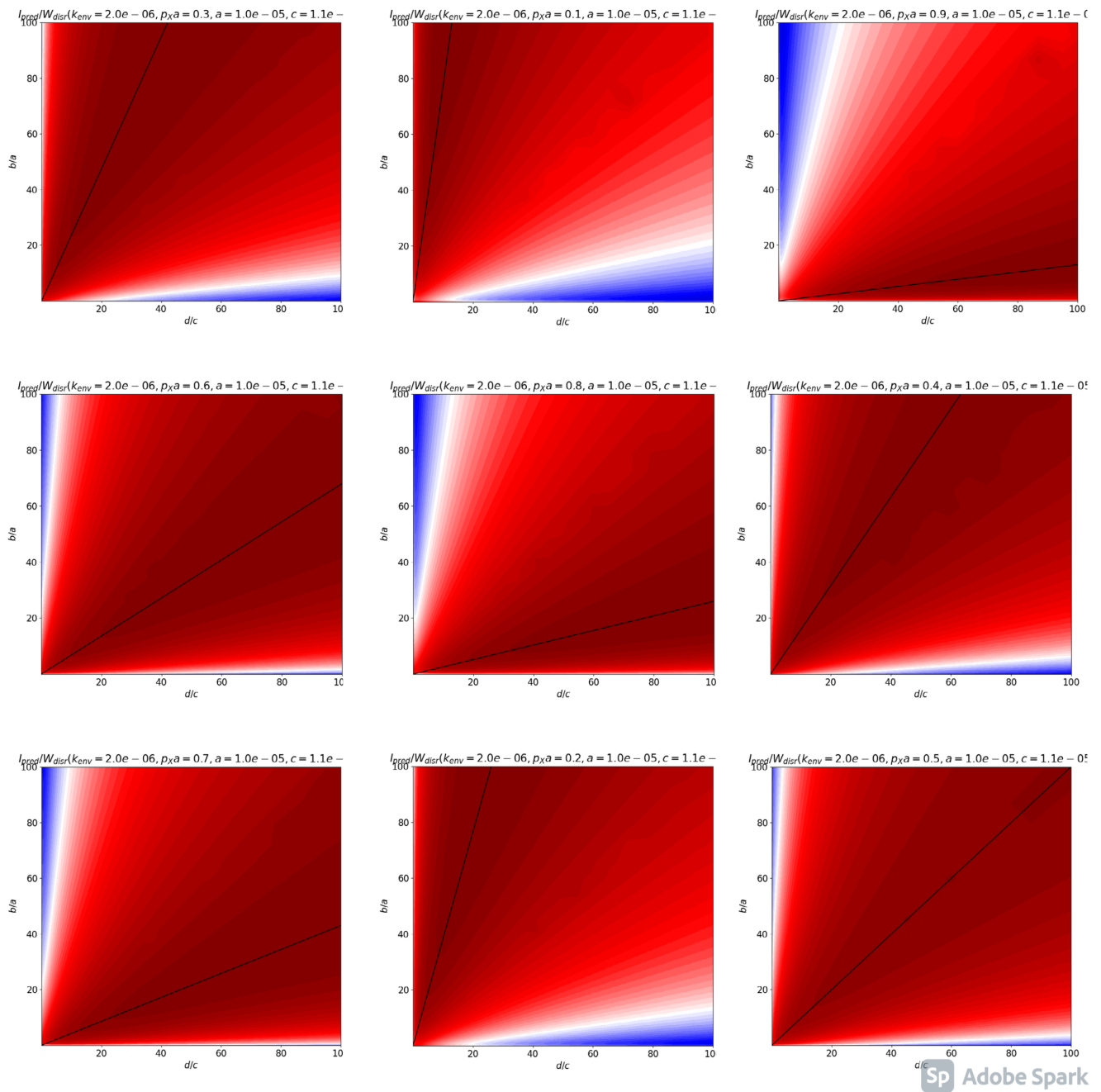


Рис. 4.7. Не логарифмічна карта ефективності для різних $p_X a$, $k_{env} = 2 \cdot 10^{-6}$

Щоб впевнитись, що обраний підхід є універсальним було також взято граничне значення.

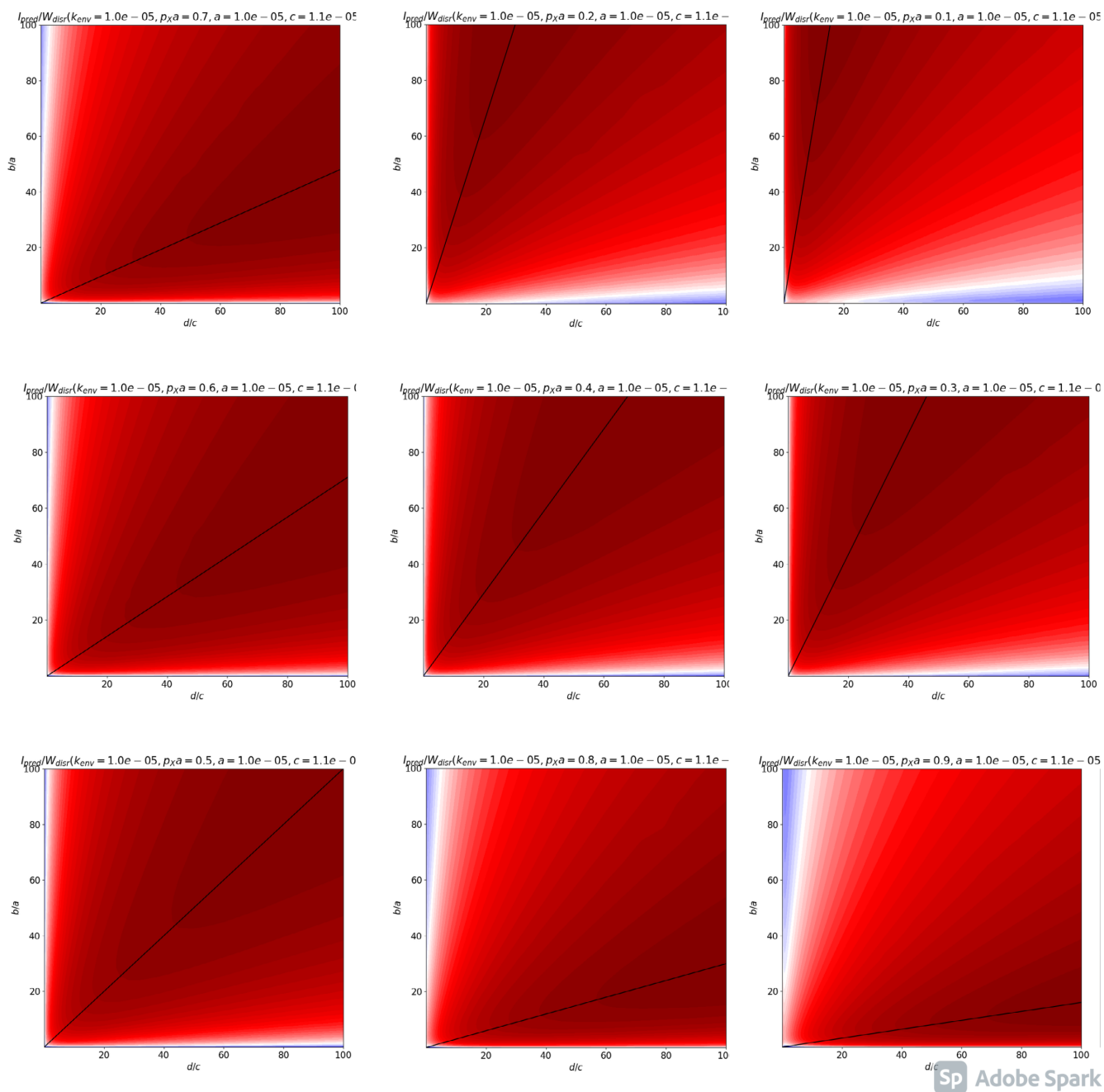


Рис. 4.8. Не логарифмічна карта ефективності для різних $p_X a$, $k_{env} = 10^{-5}$

4.3. Систематизація отриманих траєкторій

Коефіцієнти нахилу траєкторій наведено у таблиці ??:

$p_X a$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
$k_{env} = 2 \cdot 10^{-6}$	7,71	3,83	2,37	1,58	1,00	0,68	0,43	0,26	0,13
$k_{env} = 10^{-5}$	6,46	3,36	2,17	1,47	1,00	0,71	0,48	0,30	0,16

Отриманні під час попереднього кроку коефієнти тепер необхідно систематизувати. Оскільки карти ефективності є симетричними відносно незміщеного положення середовища, отримані коефіцієнти також володіють цією симетрією. Відповідно необхідно спробувати апроксимувати їх залежністю вигляду $1/x$. Незважаючи на вищенаведені міркування лінійна апроксимація на логарифмічному графіку показала не гірші результати. Для відкритості наведено обидва види апроксимацій для обох k_{env} .

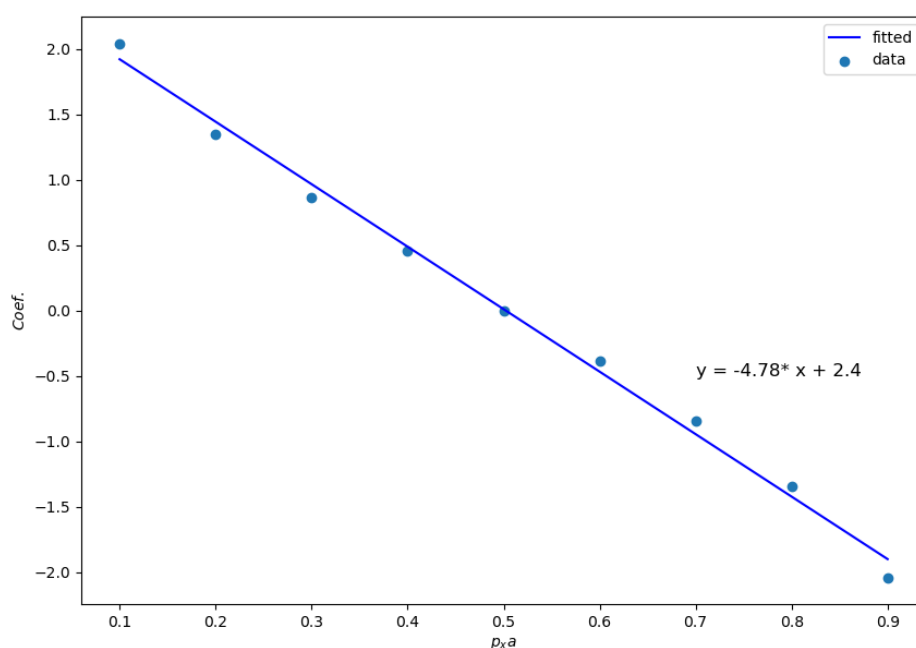


Рис. 4.9. Експоненційна апроксимація оптимальної траєкторії для $k_{env} = 2 \cdot 10^{-6}$

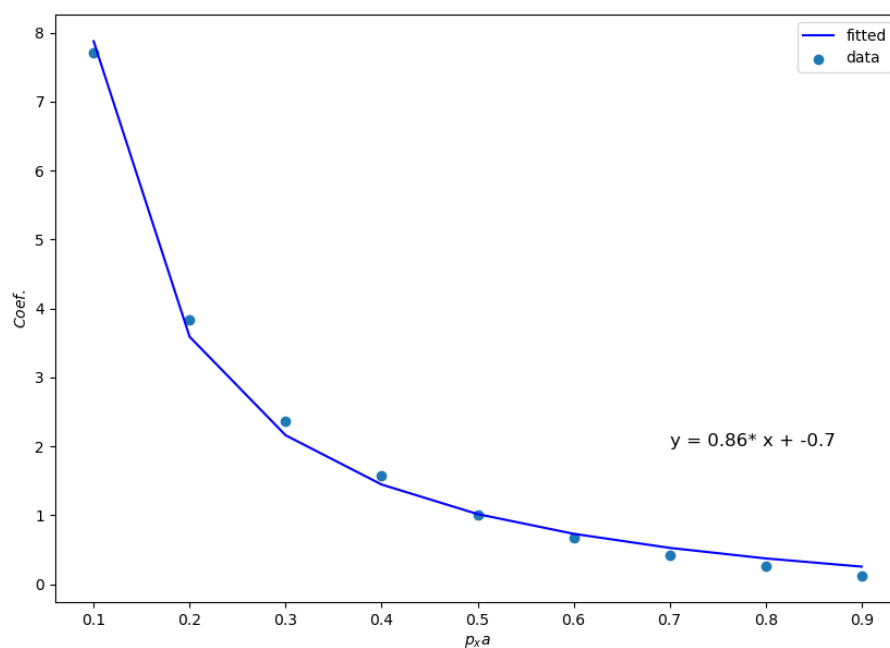


Рис. 4.10. Апроксимація $1/x$ оптимальної траєкторії для $k_{env} = 2 \cdot 10^{-6}$

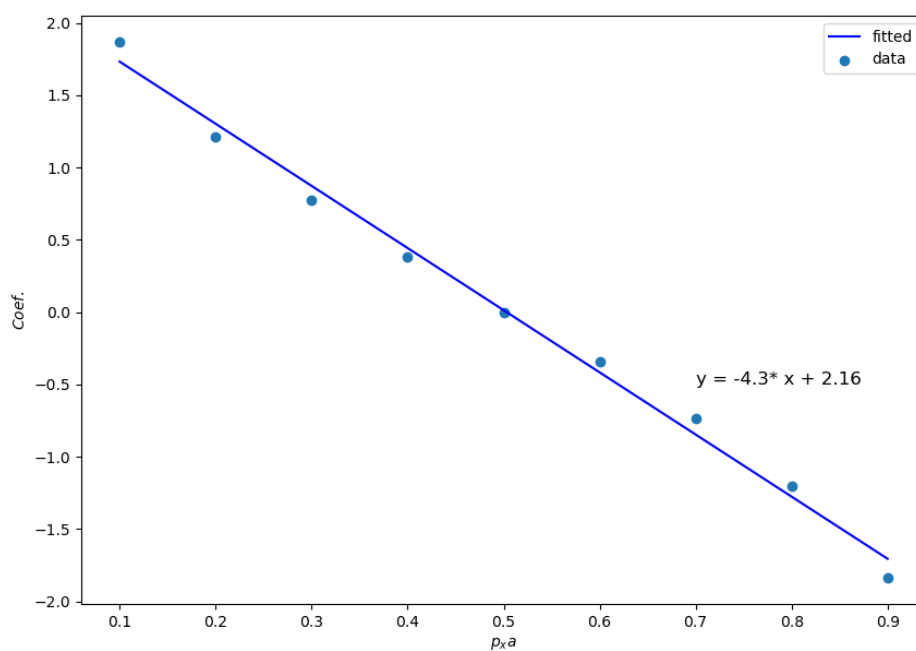


Рис. 4.11. Експоненційна апроксимація оптимальної траєкторії для $k_{env} = 10^{-5}$

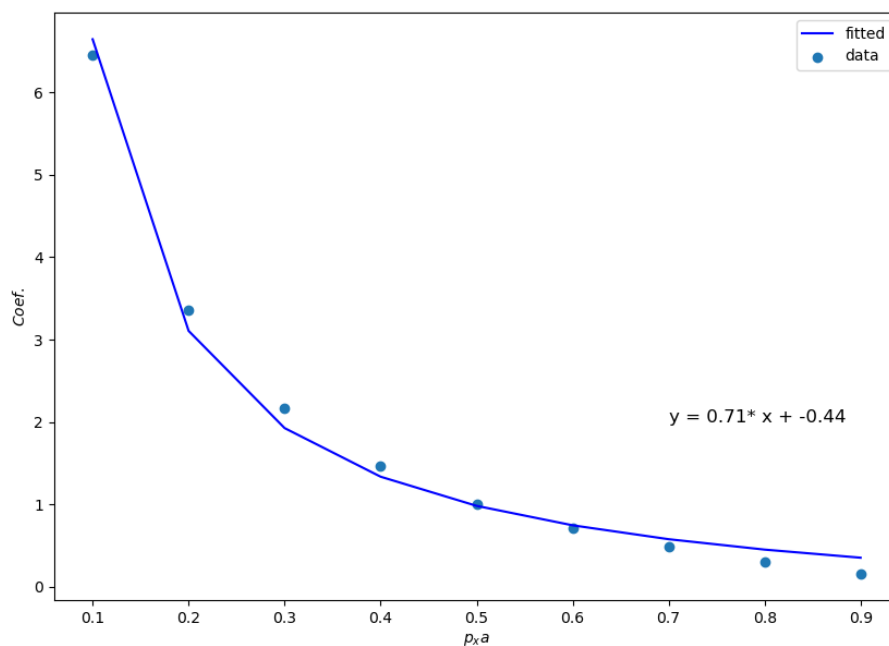


Рис. 4.12. Апроксимація $1/x$ оптимальної траєкторії для $k_{env} = 10^{-5}$

4.4. Висновки до розділу 4

Було продовжено пошук оптимальних параметрів для підвищення інформаційної ефективності молекулярної машини. Зокрема досліджено вплив зміщення точки рівноваги середовища на оптимальні параметри машини. Чисельно проаналізовано оптимальні траєкторії та систематизовано залежність цих траєкторій від параметру зміщення точки рівноваги.

ВИСНОВКИ

Було проведено аналіз поняття інформаційна ефективність молекулярної машини. Було проаналізовано існуючі моделі молекулярних машин та дослідження проведені на їх основі. Було відтворено результати іншої дослідницької групи та досліджено межі застосування наведеної моделі в рамках перевірки коректності її роботи.

Було отримано результати, що дозволили задати означення інформаційної ефективності. Надалі було проаналізовано тривіальні та нетривіальні шляхи її підвищення та зроблено висновки.

а) Літературні дослідження показали, що є зв'язок між термодинамічними величинами та величинами з теорії інформації, що дозволило обрахувати енергію що розсіюється молекулярною машину під час функціонування у середовищі.

б) Для такого дослідження було розроблено модель молекулярної машини. Збільшено її кількість ступенів свободи та додано можливість варіювати кількість станів.

в) Досліджено границь роботи симуляції та відтворено результати іншої дослідницької групи, таким чином показано правильність роботи симуляції

г) Досліджено що таке інформаційна ефективність молекулярної машини та способи її підвищення, спираючись на дослідження енергії, що розсіюється машиною та інформації, яку вона отримує.

д) Розглянуто як змінюються карти ефективності в залежності від основних параметрів k_{env} та $p_X a$. Розглянуто граничний випадок k_{env} при якому починає зникати різниця між зміщеною та незміщеною точками рівноваги.

е) Для стандартного та граничного випадків побудовано траєкторії, що забезпечують найбільший приріст ефективності та знайдено прямі, що їх описують для всіх досліджених $p_X a$.

ж) Отримані траєкторії було систематизовано шляхом апроксимації. Знайдено залежності між коефіцієнтами отриманих прямих та параметром середовища $p_X a$ для стандартного та граничного випадків. Оскільки було знайдено дві таких залежності наведено міркування стосовно кожної з них.

ПЕРЕЛІК ПОСИЛАНЬ

1. Anelli P. L., Spencer N., Stoddart J. F. A molecular shuttle. — 1991.
2. Media N. The Nobel Prize in Chemistry 2016. — 2016. — URL: <https://www.nobelprize.org/prizes/chemistry/2016/press-release/> (дата зверн. 2016).
3. David Hartich, Andre C. Barato, Udo Seifert. Efficiency of cellular information processing. — 2014.
4. Vicario J., Feringa A. M. B. L. Controlling the speed of rotation in molecular motors. — 2005.
5. Nanosized Molecular Propellers by Cyclodehydrogenation of Polyphenylene Dendrimers / C. D. Simpson [et al.]. — 2004.
6. Knipe P. C., Hamilton S. T. A. D. Ion-mediated conformational switches. — 2015.
7. Bissell R. A., Kaifer E. C. A. E., Stoddart J. F. A chemically and electrochemically switchable molecular shuttle. — 1994.
8. Directional Control in Thermally Driven Single-Molecule Nanocars / Y. Shirai [et al.]. — 2005.
9. STRAITSTIMES T. Nano-car race in France to be world's first. — 2017. — URL: <https://www.straitstimes.com/world/europe/nano-car-race-in-france-to-be-worlds-first> (дата зверн. 2017).
10. Mati I. K., Cockroft S. L. Molecular balances for quantifying non-covalent interactions. — 2010.
11. A DNA-fuelled molecular machine made of DNA / B. Yurke [et al.]. — 2000.
12. Klärner F.-G., Kahlert B. Molecular Tweezers and Clips as Synthetic Receptors. Molecular Recognition and Dynamics in Receptor–Substrate Complexes. — 2003.

13. Nanorobot architecture for medical target identification / A. Cavalcanti [et al.]. — 2007.
14. Silva A. P. de, McClenaghan N. D. Proof-of-Principle of Molecular-Scale Arithmetic. — 2000.
15. Stereodivergent synthesis with a programmable molecular machine / D. Leigh [et al.]. — 2017.
16. Kazem-Rostami M., Moghanian A. Hünlich base derivatives as photo-responsive Λ -shaped hinges. — 2017.
17. Matthew Quenneville, David A. Sivak. Energy Dissipation and Information Flow in Coupled Markovian Systems. — 2018.
18. Thermodynamics of Prediction / Susanne Still [et al.]. — 2012.
19. Rory A. Brittain, Nick S. Jones, Thomas E. Ouldrige. What we learn from the learning rate. — 2017.
20. David Hartich, Andre C. Barato, Udo Seifert. Sensory capacity: An information theoretical measure of the performance of a sensor. — 2016.
21. Horowitz J. M. E. M. Thermodynamics with Continuous Information Flow. — 2014.
22. Seifert U. Stochastic thermodynamics, fluctuation theorems and molecular machines. — 2012.
23. Thomas M. Cover, Joy A. Thomas. Elements of information theory. — New Jersey : A John Wiley Sons Inc., 2006.
24. В. Ю. Поляков Н. О. Гордійко D. S. Інформаційна ефективність молекулярних машин. — 2019.

ДОДАТОК А.

КОД СИМУЛЯЦІЇ

```

import numpy as np
import matplotlib.pyplot as plt
import itertools
from quantumsystemfunctions import *
np.set_printoptions(precision=20)
x = []
y = []
#coeflist = [0, 7.71, 3.83, 2.37, 1.58, 1, 0.68, 0.43, 0.26, 0.13]
coeflist = [0, 6.46, 3.36, 2.17, 1.47, 1.0, 0.71, 0.48, 0.3, 0.16]
for i in range(1, 10):
    print(i)
    q_x = i/10 #moves env distribution
    g = 1 #machine speed
    k_env = 10e-06
    #k_env = 1e-06*(1.26**(i))
    e = k_env * q_x #first env probability
    f = k_env - e #second env probability
    a = g * 1.0e-05
    c = g * 1.1e-05
    work_mat = np.array([[1 - e, f],
                          # = np.zeros((dim[0], dim[0]))
                          [e, 1 - f]])
    in_prob_env = np.array([0.99, 0.01])
    # in_prob_env = np.zeros(dim[0])
    dim = np.array([2, 2]) # dim = [env_states, system_states]
    n = 101
    val_Mat = np.zeros((n, n))
    val_Mat1 = np.zeros((n, n))
    val_Mat2 = np.zeros((n, n))
    val_Mat_l = np.zeros((n, n))

```

```

b_vect = []
d_vect = []
_p = 1.098
for i_a in range(n):
    b = 0.01*1e-05*(_p**i_a)
    b_vect.append(b/a)
for i_c in range(n):
    d = 0.01*1e-05*(_p**i_c)
    d_vect.append(d/c)
for i_a in range(n):
    b = 0.01*1e-05*(_p**i_a)
    for i_c in range(n):
        d = 0.01*1e-05*(_p**i_c)
        # t_max = 100000
        # relax_dists = np.zeros(list(dim) + [t_max])
        # work_dists = np.zeros(list(dim) + [t_max])
        rates = (a, b, c, d)
        relax_mat = get_relax_mat(rates)
        # relax_mat = np.zeros([dim[0], dim[1], dim[1]])
        # prob = get_initial_distribution(in_prob_env, relax_mat)
        # prob = np.zeros(dim)
        # for t in range(t_max):
        #     relax_dists[:, :, t] = prob[:, :]
        #     prob = work_step(prob, work_mat)
        #     work_dists[:, :, t] = prob[:, :]
        #     prob = relax_step(prob, relax_mat)
        # time = range(t_max)
        eq_prob = get_equilibrium_dist(dim, work_mat, relax_mat)
        val = mutual_information(work_step(eq_prob, work_mat))
        val1 = w_diss_one_step(dim, eq_prob, relax_mat,
                                work_mat)/k_env

        val2 = val/val1
        #x.append(b)
        #y.append(val2)
    val_Mat[i_a, i_c] = val

```

```

val_Mat1[i_a, i_c] = val1
val_Mat2[i_a, i_c] = val2
# val_Mat2_grad_x, val_Mat2_grad_y = np.gradient(val_Mat2)

plot_conts(val_Mat2, d_vect, b_vect, val_Mat1,
            '$I_{pred}/W_{disr}$ ',
            ('$d/c$', '$b/a$'), '$I_{pred}/W_{disr}$' +
            "($k_{env}$="+"{:.1e},p_Xa="+"{:.1f},"
            "a="+"{:.1e},c="+"{:.1e}$").format(k_env,
                                                q_x, a, c),
            'I_pred%W_disr_conts_log_{a}_{:.1e}'
            '_{:.1f}_{:.1e}_{:.1e}_ '
            'interp_rev'.format(float(i)/10,
                                k_env, q_x, a, c)+'.png',
            np.max(val_Mat2), np.min(val_Mat2),
            coeflist[i])

'''plot_conts(val_Mat, d_vect, b_vect, val_Mat1, '$I$',
            ('$d/c$', '$b/a$'), '$I_{nosr}/I$ ' +
            "($k_{env}$ = "+"{:.1e},p_Xa = {:.1f},
            a = {:.1e}, c = {:.1e}$").format(k_env,
                                                q_x, a, c),
            'I_nosr%I_conts_log_{a}_{:.1e}_{:.1f}'
            '_{:.1e}_{:.1e}_interp_rev'.format(float(i)/10,
                                                k_env, q_x, a, c)+'.png',
            np.max(val_Mat), np.min(val_Mat))'''

'''plot_conts(val_Mat1, d_vect, b_vect, val_Mat1, '$W_{disr}$ ',
            ('$d/c$', '$b/a$'), '$W_{disr}$ ' +
            "($k_{env}$ = " + "{:.1e},p_Xa = {:.1f},
            a = {:.1e}, c = {:.1e}$").format(k_env,
                                                q_x, a, c),
            'W_disr_conts_log_{a}_{:.1e}_{:.1f}_{:.1e}'
            '_{:.1e}_interp_rev'.format(float(i) / 10,
                                                k_env, q_x, a, c) +
            '.png',

```

```

np.max(val_Mat1), np.min(val_Mat1)) '''

'''plt.figure(figsize=(15, 15/1.61))
plt.plot(x, y)
plt.xlabel('d=2b')
plt.ylabel('$I_{pred}/W_{disr}$')
plt.show()'''

'''fig, ax1 = plt.subplots()

ax1.set_xlabel('a=b=d=c/1.01')
ax1.set_ylabel('$I_{pred}/W_{disr}$', color='b')
ax1.plot(x, y, color='b', linewidth=2)
ax1.tick_params(axis='y', labelcolor='b')

ax2 = ax1.twinx()
# instantiate a second axes that shares the same x-axis

ax2.set_ylabel('$I_{pred}$', color='black')
# we already handled the x-label with ax1
ax2.plot(x, yi, color='black')
ax2.tick_params(axis='y', labelcolor='black')

fig.tight_layout()
# otherwise the right y-label is slightly clipped
plt.show()'''

# Module

import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import scipy.interpolate as interp
import itertools
import sys

```

```

from matplotlib.colors import LogNorm
import os

def safe_log(argument):
    # Log base 2, setting log(0)=0
    with np.errstate(divide='ignore', invalid='ignore'):
        return np.ma.log(argument).filled(0.0)/np.log(2)

# Additional probabilities calculation
def marginal_prob_y(joint_prob): #  $p(y)$ 
    return np.sum(joint_prob, axis=0)

def marginal_prob_x(joint_prob): #  $p(x)$ 
    return np.sum(joint_prob, axis=1)

def conditional_prob_x(joint_prob): #  $p(y|x)$ 
    with np.errstate(divide='ignore', invalid='ignore'):
        marginal_prob = marginal_prob_x(joint_prob)
        conditional_prob = np.einsum('ij ..., i...->ij ...',
                                     joint_prob, 1/marginal_prob)
    return conditional_prob

def conditional_prob_y(joint_prob): #  $p(x|y)$ 
    with np.errstate(divide='ignore', invalid='ignore'):
        marginal_prob = marginal_prob_y(joint_prob)
        conditional_prob = joint_prob/marginal_prob
        conditional_prob[np.where(np.abs(
            marginal_prob -
            np.zeros(marginal_prob.shape)) < 10**(-14)))] = 0
    return conditional_prob

```

Entropies Calculation

```

def joint_entropy(joint_prob):  #  $H(x,y)$ 
    return -np.sum(joint_prob*safe_log(joint_prob),
                    axis=(0, 1))

def marginal_entropy_x(joint_prob):  #  $H(x)$  Checked
    marginal_prob = marginal_prob_x(joint_prob)
    return -np.sum(marginal_prob*safe_log(marginal_prob),
                    axis=0)

def marginal_entropy_y(joint_prob):  #  $H(y)$  Checked
    marginal_prob = marginal_prob_y(joint_prob)
    return -np.sum(marginal_prob*safe_log(marginal_prob),
                    axis=0)

def conditional_entropy_x(joint_prob):  #  $H(y|x)$  Checked
    return -np.sum(joint_prob*safe_log(conditional_prob_x(joint_prob)),
                    axis=(0, 1))

def conditional_entropy_y(joint_prob):  #  $H(x|y)$  Checked
    return -np.sum(joint_prob*safe_log(conditional_prob_y(joint_prob)),
                    axis=(0, 1))

def mutual_information(joint_prob):
    #  $I(x,y) = H(y)-H(y|x)$  Checked only nostalgia
    return marginal_entropy_y(joint_prob)-conditional_entropy_x(joint_prob)

```

```

# Free Energies

def kl_divergence(p, q): # Checked
    with np.errstate(divide='ignore'):
        return np.sum(p*safe_log(p/q), axis=1)

def f_neq_add(dim, joint_prob, relax): # Checked
    conditional_prob = conditional_prob_x(joint_prob)
    conditional_prob_eq = np.zeros(conditional_prob.shape)
    temp = get_sys_equilibrium_dist(dim, relax)
    try:
        n = len(conditional_prob[0, 0])
        for i in range(dim[1]):
            for j in range(dim[1]):
                conditional_prob_eq[i, j] = [temp[i, j]] * n
    except TypeError:
        conditional_prob_eq = temp

    return kl_divergence(conditional_prob, conditional_prob_eq)

def f_neq_add_expectation(dim, joint_prob_work, relax_mat):
    # Checked
    marginal_prob = marginal_prob_x(joint_prob_work)
    return np.einsum('i..., i...->...', marginal_prob,
                    f_neq_add(dim, joint_prob_work, relax_mat))

def delta_f_neq_add_expectation_one_step(dim, joint_prob_work, relax):
    # Checked
    joint_prob_after = relax_step(joint_prob_work, relax)
    f_neq_add_before = f_neq_add(dim, joint_prob_work, relax)
    f_neq_add_after = f_neq_add(dim, joint_prob_after, relax)
    delta_f_neq_add = f_neq_add_after - f_neq_add_before

```



```

marginal_prob = marginal_prob_x(joint_prob_work)
return np.einsum('i...,i...->...', marginal_prob,
                  delta_f_neq_add)

```

```

def delta_f_neq_add_expectation_timeseries(dim,
      joint_prob_relax, joint_prob_work, relax):  # Checked
    joint_prob_before = joint_prob_work[:, :, :-1]
    joint_prob_after = joint_prob_relax[:, :, 1:]
    f_neq_add_before = f_neq_add(dim, joint_prob_before, relax)
    f_neq_add_after = f_neq_add(dim, joint_prob_after, relax)
    delta_f_neq_add = f_neq_add_after - f_neq_add_before

    marginal_prob = marginal_prob_x(joint_prob_before)
    return np.einsum('i...,i...->...', marginal_prob,
                    delta_f_neq_add)

```

Other Quantities

```

def nostalgia_one_step(joint_prob, work_mat):  # Checked
    return mutual_information(joint_prob) - \
        mutual_information(work_step(joint_prob, work_mat))

def nostalgia_timeseries(joint_prob_relax, joint_prob_work):  # Checked
    return mutual_information(joint_prob_relax) - \
        mutual_information(joint_prob_work)

def w_diss_one_step(dim, joint_prob_relaxed, relax_mat, work_mat):
    # Checked
    return nostalgia_one_step(joint_prob_relaxed, work_mat) - \
        delta_f_neq_add_expectation_one_step(dim,
        work_step(joint_prob_relaxed, work_mat), relax_mat)

```

```

def w_diss_timeseries(dim, joint_prob_relax , joint_prob_work , relax_mat):
    # Checked
    return nostalgia_timeseries(joint_prob_relax ,
                                joint_prob_work)[:−1] − \
                                delta_f_neq_add_expectation_timeseries(dim,
                                joint_prob_relax , joint_prob_work , relax_mat)

def mu_one_step(dim, joint_prob_relaxed , relax_mat , work_mat):
    # Checked
    return nostalgia_one_step(joint_prob_relaxed , work_mat) / \
    w_diss_one_step(dim, joint_prob_relaxed ,
                    relax_mat , work_mat)

def mu_timeseries(dim, joint_prob_relax ,
                    joint_prob_work , relax_mat):
    # Checked
    return nostalgia_timeseries(joint_prob_relax ,
                                joint_prob_work)[:−1] / \
    w_diss_timeseries(dim, joint_prob_relax ,
                    joint_prob_work , relax_mat)

def ly_ent_one_step(joint_prob , relax_mat):
    # Checked with ly_one_step
    # return conditional_entropy_y(relax_step(work_step(joint_prob ,
    # work_mat), relax_mat)) − \
    # conditional_entropy_y(work_step(joint_prob , work_mat))
    return conditional_entropy_y(joint_prob) − \
    conditional_entropy_y(relax_step(joint_prob , relax_mat))

def hx_ent_one_step(joint_prob , work_mat):

```

```

# Checked with hx_one_step
return conditional_entropy_y(work_step(joint_prob ,
                                     work_mat)) - conditional_entropy_y(joint_prob)

def ly_one_step(joint_prob , relax_mat):
    # Checked with ly_ent_one_step
    a = np.einsum('ikj , ij -> ijk' , relax_mat , safe_log(joint_prob))
    b = np.einsum('ikj , ik -> ijk' , relax_mat , safe_log(joint_prob))
    return -np.einsum('ij , ijk ->' , joint_prob , a-b)

def ly_timeseries(joint_probs , relax_mat): # Checked with ly_one_step
    a = np.einsum('ikj , ijt -> ijkt' , relax_mat , safe_log(joint_probs))
    b = np.einsum('ikj , ikt -> ijkt' , relax_mat , safe_log(joint_probs))
    return -np.einsum('ijt , ijkt -> t' , joint_probs , a-b)

def hx_one_step(joint_prob , work_mat):
    # Checked with hx_ent_one_step
    a = np.einsum('ij , ki , ij ->' , joint_prob ,
                  work_mat , safe_log(joint_prob))
    b = np.einsum('ij , ki , kj ->' , joint_prob ,
                  work_mat , safe_log(joint_prob))
    return a-b

def hx_timeseries(joint_prob , work_mat):
    # Checked with hx_ent_one_step
    a = np.einsum('ijt , ki , ijt -> t' , joint_prob ,
                  work_mat , safe_log(joint_prob))
    b = np.einsum('ijt , ki , kjt -> t' , joint_prob ,
                  work_mat , safe_log(joint_prob))
    return a-b

```

Functions for evolving system

```
def work_step(joint_prob , work_mat_loc):
    return np.einsum('ij ,jk ', work_mat_loc , joint_prob)

def relax_step(joint_prob , relax_mat_loc):
    return np.einsum('ijk ,ik->ij ', relax_mat_loc , joint_prob)
```

Equilibrium distributions

```
def get_initial_distribution(initial_prob_x , relax):
    relax_mat_eq = np.einsum('i ,ijk ', initial_prob_x , relax)
    eigvals , eigvecs = np.linalg.eig(relax_mat_eq)
    eq = np.squeeze(eigvecs[:,
                        np.abs(np.sum(eigvecs , axis=0)) > 0.1])
    initial_prob = np.einsum('i ,j->ij ',
                              initial_prob_x , eq / np.sum(eq))

    assert abs(initial_prob.sum() - 1.) < 10 ** -15
    return initial_prob

def get_equilibrium_dist(dim , work , relax):
    relax_big = np.zeros([dim[0]*dim[1]]*2)
    work_big = np.zeros([dim[0]*dim[1]]*2)
    for i in range(dim[0]):
        relax_big[i*dim[1]:(i+1)*dim[1] ,
                  i*dim[1]:(i+1)*dim[1]] = relax[i]
        for j in range(dim[0]):
            work_big[i*dim[1]:(i+1)*dim[1] ,
                     j*dim[1]:(j+1)*dim[1]] = np.diag([work[i , j]]*dim[1])
    eq = np.zeros(dim)
    trans_mat = np.dot(relax_big , work_big)
    eigvals , eigvecs = np.linalg.eig(trans_mat)
```

```
eq = np.squeeze(eigvecs[:,
                    np.abs(np.sum(eigvecs, axis=0)) > 0.1])
return np.reshape(eq/np.sum(eq), dim)
```

```
def get_sys_equilibrium_dist(dim, relax):
    # Checked because f_neq_add Checked
    eq = np.zeros(dim)
    for i in range(dim[0]):
        eigvals, eigvecs = np.linalg.eig(relax[i])
        eq[i] = np.squeeze(eigvecs[:,
                                np.abs(np.sum(eigvecs, axis=0)) > 0.1])
    return np.transpose(np.transpose(eq)/np.sum(eq, axis=1))
```

```
def diagonal_parametrization(dim):
```

```
    work = np.diag([1]*dim[0])
    relax = np.diag([1]*dim[1])
```

```
    return work, relax
```

```
def plot_dists(dim, time, probs_loc, prob_eq, title, filename):
    plt.figure(figsize=(15, 15/1.61))
    ax = plt.subplot(1, 2, 1)
    plt.xlabel('$Time$', fontsize=15)
    plt.ylabel(r'$P(x,y)$', fontsize=15)
    plt.title(title, fontsize=18)
    for i in range(dim[0]):
        for j in range(dim[1]):
            ax.plot(time, probs_loc[i, j],
                    'C'+str(4*i+j), label=str((i, j)))
            ax.plot([0, time[-1]],
                    [prob_eq[i, j], prob_eq[i, j]], 'C'+str(4*i+j)+'--')
```

```
ax.legend()
```

```
ax1 = plt.subplot(2, 2, 2)
plt.ylabel(r'$P(x)$', fontsize=15)
for i in range(dim[0]):
    ax1.plot(time, marginal_prob_x(probs_loc)[i],
             'C'+str(i), label='env'+str(i))
    ax1.plot([0, time[-1]],
             [marginal_prob_x(prob_eq)[i],
              marginal_prob_x(prob_eq)[i]],
             'C'+str(i)+'--')
ax1.legend()
```

```
ax1 = plt.subplot(2, 2, 4)
plt.ylabel(r'$P(y)$', fontsize=15)
for i in range(dim[1]):
    ax1.plot(time, marginal_prob_y(probs_loc)[i],
             'C'+str(i), label='sys'+str(i))
    ax1.plot([0, time[-1]],
             [marginal_prob_y(prob_eq)[i],
              marginal_prob_y(prob_eq)[i]],
             'C'+str(i)+'--')
ax1.legend()
plt.savefig('./Vovaplots/Max_I_pred_to_W_diss_1/Var_New51/'+filename)
plt.close()
```

```
def plot_cond_dists(dim, time, probs_loc,
                    prob_eq1, prob_eq2, title, filename):
    plt.figure(figsize=(15, 15/1.61))
    ax = plt.subplot(1, 1, 1)
    plt.title(title, fontsize=18)
    plt.xlabel('$Time$', fontsize=15)
    plt.ylabel(r'$\frac{P(x,y)}{P(x)}$', fontsize=15)
    for i in range(dim[0]):
```

```

    for j in range(dim[1]):
        ax.plot(time, probs_loc[i, j],
                'C'+str(4*i+j), label=str((i, j)))
        ax.plot([0, time[-1]], [prob_eq1[i, j],
                                prob_eq1[i, j]], 'C'+str(4*i+j)+'--')
        ax.plot([0, time[-1]], [prob_eq2[i, j],
                                prob_eq2[i, j]], 'C'+str(4*i+j)+'-.')
ax.legend()

plt.savefig('./Vovaplots/quantities/'+filename)
plt.close()

def plot_timeseries(time, values, title, y, filename):
    plt.figure(figsize=(15, 15/1.61))
    plt.plot(time, values)
    plt.title(title, fontsize=18)
    plt.ylabel(y, fontsize=15)
    # plt.ylim((0, 10**(-6)))
    if y == 'mu':
        plt.ylim((0, 1))
    plt.xlabel('$Time$', fontsize=15)
    plt.savefig('./Vovaplots/quantities/'+filename)
    plt.close()

def plot_f(values, title, y, filename):
    plt.figure(figsize=(15, 15/1.61))
    for i in range(1, values.shape[0]):
        plt.plot(values[i], label=str(i))
    plt.legend()
    plt.title(title, fontsize=18)
    plt.ylabel(y, fontsize=15)
    # plt.ylim((0, 0.025))
    plt.xlabel('$Time$', fontsize=15)

```

```

plt.savefig(' ./Vovaplots/ Severally /'+filename)
plt.close()
plt.figure(figsize=(15, 15/1.61))
for i in range(1, values.shape[0]):
    plt.plot(safe_log(values[i]), label=str(i))
plt.legend()
plt.title(title , fontsize=18)
plt.ylabel(y, fontsize=15)
# plt.ylim((0, 0.025))
plt.xlabel('$Time$', fontsize=15)
plt.savefig(' ./Vovaplots/ Severally /Log_'+filename)
plt.close()

```

```

def plot_mu(values , title , y, filename):
    plt.figure(figsize=(15, 15/1.61))
    for i in range(1, values.shape[0]):
        plt.plot(values[i], label=str(i))
    plt.legend()
    plt.title(title , fontsize=18)
    plt.ylabel(y, fontsize=15)
    plt.ylim((0, 1))
    plt.xlabel('$Time$', fontsize=15)
    plt.savefig(' ./Vovaplots/ Severally /'+filename)
    plt.close()

```

From Matt

```

def get_pars():
    pars = [10**(-5), 0.25]
    return tuple(pars)

```

```

def get_rates(pars):

```



```

k = pars[0]
delta_e = pars[1]
k_minus = k*(1+np.exp(-delta_e))/2
k_plus = k_minus/np.exp(-delta_e)
return k_plus, k_minus, k_minus, k_plus

```

```

def get_relax_mat(pars):
    a_val, b_val, c_val, d_val = pars

    return np.array(
        [[[1-a_val, b_val],
          [a_val, 1-b_val]],
         [[1-c_val, d_val],
          [c_val, 1-d_val]]]
    )

```

```

def plot_heatmap(data, yticks, xticks, name, axes, title, filename,
                  maxval=None, minval=None):
    font = {'weight': 'normal', 'size': 16}
    matplotlib.rc('font', **font)

    n_grid = data.shape[0]
    units = ''

    color_map = plt.cm.seismic
    color_map.set_bad(color='k')
    if maxval is None:
        maxval = np.nanmax(data)
    if minval is None:
        minval = min(np.nanmin(data), 0)
    fig = plt.figure(figsize=(15, 15/1.61))

    plt.imshow(data, aspect='auto', interpolation='nearest',

```

```

        cmap=color_map , origin='lower')
# vmin=0, vmax=0.05

ax = plt.gca()
ticks_pos = list(range(n_grid))
ax.set_xticks(ticks_pos[::5])
ax.set_yticks(ticks_pos[::5])
ax.set_xticklabels(xticks[::5])
ax.set_yticklabels(yticks[::5])
plt.setp(ax.get_xticklabels(), rotation=45,
          ha="right", rotation_mode="anchor")
plt.xlabel(axes[1], size=16)
plt.ylabel(axes[0], size=16, labelpad=0)
cb = plt.colorbar()
cb.set_label(name + units , labelpad=10)
plt.suptitle(title)
fig.subplots_adjust(bottom=0.15, left=0.15)
plt.savefig('./Vovaplots/Max_I_pred_to_W_diss_1/'+filename)
plt.close('all')

def plot_conts(Z, x, y, Z1, name, axes,
               title, filename, max, min, coef):
    font = {'weight': 'normal', 'size': 16}
    matplotlib.rc('font', **font)
    X, Y = np.meshgrid(x, y)
    plt.figure(figsize=(12, 9))
    cmap = plt.cm.seismic
    levels = np.linspace(min, max, 50)
    cs = plt.contourf(X, Y, Z, levels[::1])
    contour_filled = plt.contourf(X, Y, Z, levels, cmap=cmap)
    plt.colorbar(contour_filled)
    #levels1 = np.linspace(np.max(Z1)/100+np.min(Z1), np.max(Z1), 20)
    #cs1 = plt.contour(X, Y, Z1, levels1)
    plt.title(title)
    plt.xlabel(axes[0])

```

```

plt.ylabel(axes[1])
_x = np.linspace(0, 100, 101)
#_irange = np.linspace(7, 11, 3)
#for i in _irange:
plt.plot(_x, _x * coef, 'black')
plt.ylim((0.01, 100))
plt.xlim((0.01, 100))
#plt.xscale('log')
#plt.yscale('log')
plt.savefig('./Vovaplots/Master/ShiftDependency2nonlog/' + filename)
plt.show()
#plt.close()

```

```

def plot_cont_grad(Z, x, y, U, V, name,
                    axes, title, filename, lmax, lmin):
    font = {'weight': 'normal', 'size': 16}
    matplotlib.rc('font', **font)
    print(name)
    X, Y = np.meshgrid(x, y)
    plt.figure(figsize=(14, 11))
    cmap = plt.cm.seismic
    levels = np.linspace(lmin, lmax, 40)
    cs = plt.contourf(X, Y, Z, levels[::5])
    contour_filled = plt.contourf(X, Y, Z, levels, cmap=cmap)
    plt.quiver(X, Y, V, U)
    plt.colorbar(contour_filled)
    plt.title(title)
    plt.scatter(1, 1)
    plt.xlabel(axes[0])
    plt.ylabel(axes[1])
    plt.ylim((0.001, 0.1))
    plt.xlim((10, 1000))
    plt.xscale('log')
    plt.yscale('log')

```

```
plt.savefig(' ./Vovaplots/Max_I_pred_to_W_diss_1/' + filename)
plt.show()
```

```
def plot_cont_interp(Z, x, y, U, V, name,
                    axes, title, filename, lmax, lmin):
    font = {'weight': 'normal', 'size': 16}
    matplotlib.rc('font', **font)
    print(name)
    xlim = [np.min(x), np.max(x)]
    ylim = [np.min(y), np.max(y)]
    X, Y = np.meshgrid(x, y)
    f = interp.interp2d(x, y, Z)
    x1 = np.linspace(xlim[0], xlim[1], 400)
    y1 = np.linspace(ylim[0], ylim[1], 400)
    X1, Y1 = np.meshgrid(x1, y1)
    Z1 = f(x1, y1)
    plt.figure(figsize=(14, 11))
    cmap = plt.cm.seismic
    cmaprev = plt.cm.seismic_r
    levels = np.linspace(lmin, lmax, 40)
    cs = plt.contourf(X1, Y1, Z1, levels[::5])
    contour_filled = plt.contourf(X1, Y1, Z1, levels, cmap=cmap)
    plt.quiver(X, Y, V, U)
    plt.colorbar(contour_filled)
    plt.title(title)
    plt.scatter(1, 1)
    plt.xlabel(axes[0])
    plt.ylabel(axes[1])
    plt.xlim((0.1, 10))
    plt.ylim((0.1, 10))
    plt.xscale('log')
    plt.yscale('log')
    x = 1
    y = 1
```

```

xline = []
yline = []
while xlim[0] < x < xlim[1] and ylim[0] < y < ylim[1]:
    dx = x/1000
    dy = y/1000
    grad_x = (f(x+dx, y)[0]-f(x-dx, y)[0])/(2*dx)
    grad_y = (f(x, y+dy)[0]-f(x, y-dy)[0])/(2*dy)
    x += (x/1000)*grad_x/abs((grad_x+grad_y))
    y += (y/1000)*grad_y/abs((grad_x+grad_y))
    xline.append(x)
    yline.append(y)
print(len(xline))
plt.plot(xline, yline, lw=1)
plt.savefig('./Vovaplots/Max_I_pred_to_W_diss_1/Var/' + filename)
plt.show()

```

```

def plot_big_heatmap(data, yticks, xticks,
                     name, axes, title, filename,
                     maxval=None, minval=None):
    font = {'weight': 'normal', 'size': 16}
    matplotlib.rc('font', **font)

    n_grid = data.shape[0]

    plt.figure(figsize=(15, 15/1.6))

    if maxval is None:
        maxval = np.nanmax(data)
    if minval is None:
        minval = np.nanmin(data)
    print(minval)
    print(maxval)

    for i in range(n_grid):

```

```

for j in range(n_grid):
    plt.subplot(n_grid, n_grid, (n_grid-i-1)*n_grid+j+1)
    plt.imshow(data[i, j], cmap=plt.cm.seismic,
               vmin=minval, vmax=maxval, origin='lower')
    ticks_pos = list(range(n_grid))
    plt.xticks([])
    plt.yticks([])
    # if j==0:
    #     plt.yticks(ticks_pos[::2], yticks[::2])
    # if i==n_grid-1:
    #     plt.xticks(ticks_pos[::2], xticks[::2], rotation=40)
plt.subplots_adjust(bottom=0.15, right=0.8, top=0.9)
cax = plt.axes([0.85, 0.15, 0.03, 0.8])
plt.colorbar(cax=cax)
# plt.savefig(' ./Vovaplots/Min_W_diss/' + filename)
plt.show()

```

```

import numpy

```

```

from scipy.optimize import curve_fit

```

```

import matplotlib.pyplot as plt

```

```

x = numpy.array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
#y = numpy.array([6.46, 3.36, 2.17, 1.47, 1.0, 0.71, 0.48, 0.3, 0.16])
y = numpy.array([7.71, 3.83, 2.37, 1.58, 1.00, 0.68, 0.43, 0.26, 0.13])
#y = numpy.log(y)
[a, b], res1 = curve_fit(lambda x1, a, b: a / x1 + b, x, y)

```

```

y1 = a / x + b

```

```

print(a, b)

```

```

print(numpy.sqrt(numpy.diag(res1)))

```

```

fig = plt.figure(figsize=(10, 7))

```

```

ax = fig.add_subplot()

```

```

ax.set_xlabel('$p_{xa}$')

```

```

ax.set_ylabel('$Coef$.')

```

```

ax.scatter(x, y, label='data')

```

```

ax.plot(x, y1, 'b', label='fitted')
ax.text(0.7, 2, 'y=□' + str(round(a,2)) + '*x+□' + str(round(b,2)),
        fontsize=12)
plt.legend()
plt.show()

```

```

import glob
import os

```

```

gif_name = 'finalgif'
file_list = glob.glob('*.png') # Get all the pngs in the current directory
#list.sort(file_list, key=lambda x: int(x.split('_')[1].split('.')[0]))
# Sort the images by #, this may need to be tweaked for your use case

```

```

with open('image_list.txt', 'w') as file:
    for item in file_list:
        file.write("%s\n" % item)

```

```

os.system('magick -delay 10 @image_list.txt {}.gif'.format(gif_name))

```