Харківський національний університет імені В.Н. Каразіна
Факультет математики і інформатики
Кафедра прикладної математики

Кваліфікаційна робота

рівень: ***магістр***

на тему ***«Математичне та комп'ютерне моделювання хвиль у мотузковій системі»***

Виконав: студент групи МП62 II курсу (другий магістерський рівень), спеціальності 113 "Прикладна математика" освітньо-професійної програми "Прикладна математика"
Собур М.С.
Керівник: кандидат фіз.-мат. наук доцент кафедри прикладної математики
Послався́кий С.О.
Рецензент: кандидат техн. наук, старший дослідник,
Духопельников С. В.

Харків — 2024 рік

Анотації

Собур Микита Станіславович. Математичне та комп'ютерне моделювання хвиль у мотузковій системі.

У роботі було розглянуто двовимірну дискретизовану мотузкову систему. Система складається з горизонтальної натягнутої мотузки (базова мотузка) та з вертикальної ненатягнутої динамічної мотузки з вантажем на вільному кінці (мотузка-поводок). Виведені диференціальні рівняння руху системи і провели досліждення розповсюдження поперечних та поздовжних хвиль. За допомогою мови програмування Python було побудовано комп'ютерну модель для проведення експериментів з механічною системою.

Ключові слова: двовимірна мотузкова система, в'язко-пружні моделі, комп'ютерне моделювання хвильових процесів, поздовжні та поперечні хвилі.

Sobur Mykyta. Mathematical and computer modeling of waves in the rope system.

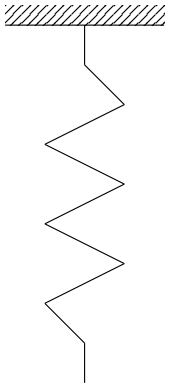
In this work, a two-dimensional discretized rope system was considered. The system consists of a horizontally tensioned rope (the base rope) and a vertically untensioned dynamic rope with a weight at its free end (the lead rope). The differential equations of motion for the system were derived, and the propagation of transverse and longitudinal waves was investigated. A computational model was developed using the Python programming language to conduct experiments with the mechanical system.

Keywords: two-dimensional rope system, viscoelastic models, computational modeling of wave processes, longitudinal and transverse waves.

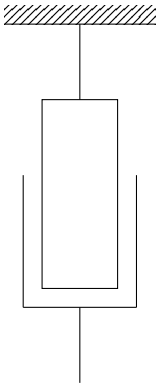
Зміст

Вступ

Сучасні мотузкові системи описуються за допомогою комбінації пружніх та в'язких елементів. За пружні властивості у реологічних моделях відповідає пружина з деяким показником жорсткості (або як ще називають тіло Гука), а за в'язкі – демпфер, що являє собою поршень із в'язкою речовиною (також відомий як тіло Ньютона). При різкому навантаженні такої системи пружні і в'язкі елементи проявляють себе одразу ж (тобто деформація виникає миттєво). Після зняття навантаження пружина прагне відновити свою колишню форму і розміри, а демпфер не відновлює свого минулого положення і таким чином зберігає деформацію незмінною (це так званий ефект пам'яті). Зазначимо, що коефіцієнти жорсткості і в'язкості у всіх рівняннях вважаємо постійними значеннями, що не залежать від часу. Нижче наведені схематичні зображення цих двох елементів.



(а) Схема пружини.



(б) Схема демпфера.

Рис. 1: Схематичні зображення пружини і демпфера.

Ідеальне пружнє тіло описується за допомогою лінійного закону Гука:

$$F = kx,$$

де x – деформація пружини, k – коефіцієнт жорсткості.

Демпфер описується за допомогою закону в'язкого тертя Ньютона:

$$F = \eta \dot{x},$$

де $\dot{x}$ – швидкість деформації, η – коефіцієнт в'язкості.

Ці два компоненти з'єднуються паралельно і послідовно, таким чином можна створювати більш складні та точні моделі, які будуть враховувати особливості поведінки реологічних систем.

Реологічні моделі

Наведемо деякі приклади реологічних моделей та рівняння, що описують їх поведінку при навантаженнях.

1. **Тіло Максвелла**

У цій моделі послідовно з'єднані пружина та демпфер.

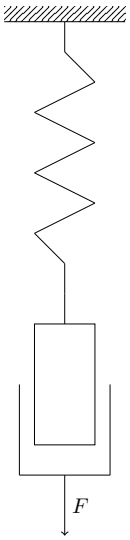


Рис. 2: Тіло Максвелла.

Рівняння для її опису у термінах реології має наступний вигляд:

$$\dot{\varepsilon} = \frac{1}{E}\dot{\sigma} + \frac{\sigma}{\eta},$$

де ε – загальна деформація системи, E – модуль пружності Юнга,

σ – загальна напруга у системі, η – коефіцієнт в'язкості речовини.

Похідна усюди береться за часом. Модель може бути узагальнена шляхом паралельного з'єднання деякої кількості таких елементів.

2. **Тіло Кельвіна-Фойгта**

На відміну від попередньї моделі, у цьому випадку пружину і демпфер з'єднано паралельно.

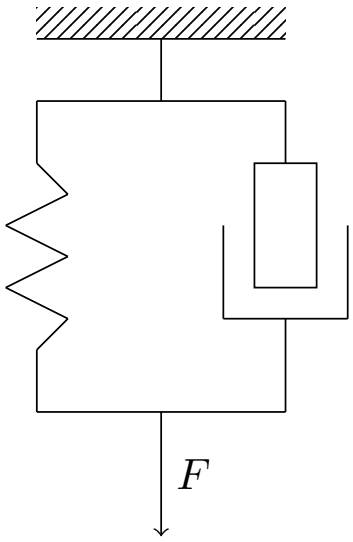


Рис. 3: Тіло Кельвіна-Фойгта.

Рівняння виглядає наступним чином:

$$\sigma = E\varepsilon + \eta\dot{\varepsilon}.$$

Модель Кельвіна-Фойгта також може бути узагальнена за рахунок послідовного з'єднання елементів.

3. Модель стандартного навантаженого лінійного тіла

Модель складається з паралельно з'єднаних пружини і тіла Максвелла.

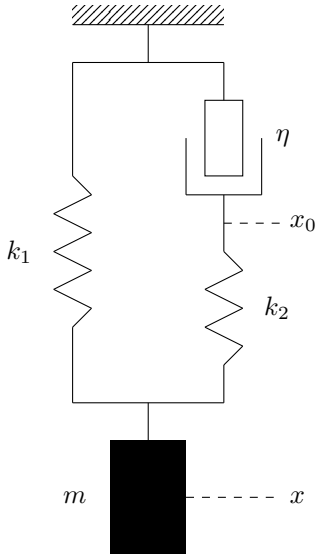


Рис. 4: Модель стандартного лінійного тіла (SLS).

Система диференціальних рівнянь для опису руху:

$$m\ddot{x} + k_1 x + k_2(x - x_0) = mg$$

$$\eta\dot{x}_0 + k_2(x_0 - x) = 0.$$

Тут x, x_0 – координати точок, m – маса вантажу, k_1, k_2 – коефіцієнти жорсткості пружин, η – коефіцієнт в'язкості демпфера, g – прискорення вільного падіння на Землі.

Ця модель і рівняння для неї були розглянуті у роботі [1]. Автор використовував її для опису властивостей альпіністських мотузок. У роботі [2] цього ж автора розглядається поняття фактору ривка (fall factor), яке дозволяє оцінити навантаження на систему страхування, що виникає при падінні альпініста. Чисельно це значення дорівнює відношенню висоти h, з якої падає людина до того як почнеться натягнення мотузки, до натуральної довжини мотузки L.

4. **Тіло Бюргерса**

Модель складається з послідовно з'єднаних тіла Максвелла та тіла Кельвіна-Фойгта. Диференціальне рівняння, що пов'язує загальну напругу і загальну де-

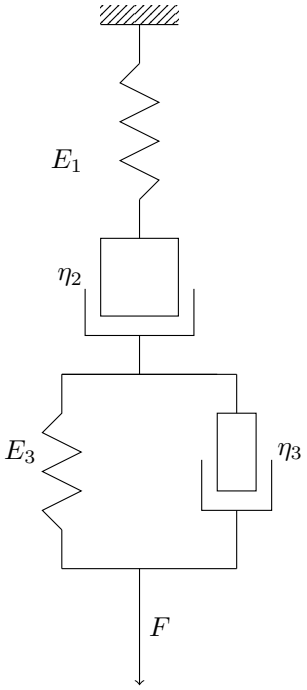


Рис. 5: Тіло Бюргерса.

формацію у системі:

$$\ddot{\sigma} + \left(\frac{E_1}{\eta_2} + \frac{E_1}{\eta_3} + \frac{E_3}{\eta_3}\right)\dot{\sigma} + \frac{E_1 E_3}{\eta_2 \eta_3}\sigma = E_1\ddot{\varepsilon} + \frac{E_1 E_3}{\eta_3}\dot{\varepsilon}.$$

Зазначимо, що усі розглянуті вище моделі є лінійними відносно деформацій та швидкостей деформацій. Але як відомо, закон пружності Гука допускає узагальнення. Воно полягає у тому, щоб окрім лінійного доданка (першого степеня деформації) у рівнянні розглядати також доданки більш високого порядку. Таким чином, можна врахувати нелінійну поведінку пружнього матеріалу при розтягуванні та стисненні. Тоді рівняння для пружньої сили буде мати наступний вигляд:

$$F = kx + ax^2 + bx^3,$$

k – коефіцієнт жорсткості, a, b – додаткові коефіцієнти, що визначають степінь нелінійності.

Ознайомитися з виводом рівнянь для наведених вище моделей і пов'язаними з цією областю задачами можна у відповідній літературі по теорії в'язко-пружності [3] та

реології [4]. Також такі моделі розглядаються у підручниках з механіки суцільних середовищ українських авторів (див. [5] с. 200-204, [6] с. 145).

Основна двовимірна модель, що буде розглянута у цій роботі також є лінійною у цьому сенсі, але у ній все рівно виникають нелінійності за рахунок саме геометрії системи.

Метою нашого дослідження є спостереження за розповсюдженням поздовжних та поперечних хвиль у мотузковій системі. Поперечні хвилі мають достатньо високу амплітуду і їх можна легко спостерігати неозброєним оком у житті. При комп'ютерному моделюванні їх можна помітити за рахунок зміни кута між фрагментами мотузки, що з'єднані точковими масами. Тобто мотузки змінюють форму, утворюючи ламані лінії при русі. У поздовжних хвилях частинки середовища, у якому відбувається хвильовий процес, коливаються паралельно до напряму поширення хвилі. У такій хвилі чергуються проміжки деформацій розтягування і стиснення. Ці хвилі достатньо важко помітити у реальних мотузкових системах, особливо при різкому навантаженні та великій швидкості руху. При комп'ютерному моделюванні цей вид хвиль можна спостерігати через зміну відстаней між точками, якими ми робимо дискретизацію. Для розповсюдження поперечних хвиль у мотузці повинен бути натяг і чим сильніше натягнута мотузка тим швидше у ній розповсюджуються хвилі. Швидкість розповсюдження поздовжних хвиль залежить від щільності мотузки та її коефіцієнту жорсткості.

Розділ 1. Двовимірна мотузкова система

У цьому розділі буде наведено детальний опис головної моделі у нашому дослідженні. Перед цим, введемо спочатку наступну систему позначень, якою ми будемо користуватися надалі.

1.1. Позначення

Нехай N – загальна кількість рухомих матеріальних точок системи. Позначимо через n_1, n_2 – кількості точок базової мотузки і поводка відповідно. Тобто маємо наступну рівність $N = n_1 + n_2$.

Позначимо через m_i точкову масу, з координатами (x_i, y_i), де x_i – горизонтальна координата точки з номером i, y_i – вертикальна координата з номером i.

Позначимо через L_1 та L_2 довжини базової мотузки та мотузки-поводка у ненатягненому стані відповідно. Тоді l_0^1 – довжина фрагменту базової мотузки між парою точкових мас у стані спокою, l_0^2 – довжина фрагменту мотузки-поводка у стані спокою. Позначимо через $l_{i,i+1}(t)$ довжину фрагмента мотузки між точковими масами m_i та m_{i+1} у момент часу t. Оскільки в процесі руху системи мотузки деформуються, то зрозуміло, що ця величина залежить від часу. Введемо позначення для характеристик мотузки:

k_1, η_1 – коефіцієнти жорсткості та в'язкості для всієї базової мотузки,

k_2, η_2 – коефіцієнти жорсткості та в'язкості для всієї мотузки-поводка.

Переходимо до позначень векторних величин. Перш за все наведемо їх для радіус-векторів координат, швидкостей і прискорень точки.

$\vec{r_i} = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$ – радіус-вектор точки m_i.

$\dot{\vec{r}}_i = \begin{pmatrix} \dot{x}_i \\ \dot{y}_i \end{pmatrix}$ – швидкість точки m_i.

$\ddot{\vec{r}}_i = \begin{pmatrix} \ddot{x}_i \\ \ddot{y}_i \end{pmatrix}$ – прискорення точки m_i.

Тепер опишемо сили та вектори, що пов'язані з ними.

$\vec{F}_i$ – сумарна сила, що діє на точкову масу m_i.

$\vec{F}_{i,i+1}$ – сила, що діє на точку m_i з боку точки m_{i+1}. Цей вектор напрямлений від точки m_i (має початок у цій точці) до m_{i+1}.

$\vec{e}_{i,i+1} = \dfrac{\vec{r}_{i+1} - \vec{r_i}}{\|\vec{r}_{i+1} - \vec{r_i}\|}$ – одиничний вектор, направлений вздовж вектора сили $\vec{F}_{i,i+1}$ (або в напрямку вектора, що з'єднує точки m_i та m_{i+1}).

$\vec{F}^{\text{пруж.}}_{i,i+1}$ – пружня складова сили $\vec{F}_{i,i+1}$.

$\vec{F}^{\text{в'яз.}}_{i,i+1}$ – в'язка складова сили $\vec{F}_{i,i+1}$.

1.2. Постановка задачі

1.2.1. Вступні зауваження

Розглянемо нашу основну механічну систему. Нехай **A**, **B** – ліва та права крайові нерухомі точки фіксації базової горизонтальної мотузки. Ці точки розташовані на одній прямій, яка паралельна до вісі Ox. Позначимо літерою **C** рухому точку цієї мотузки, розташовану посередині. До цієї точки прикріплюється друга мотузка (мотузка-поводок). На вільному кінці мотузки-поводка прикріплено вантаж **D**. Маса вантажа значно більша за кожну окрему матеріальну точку у обох мотузках.

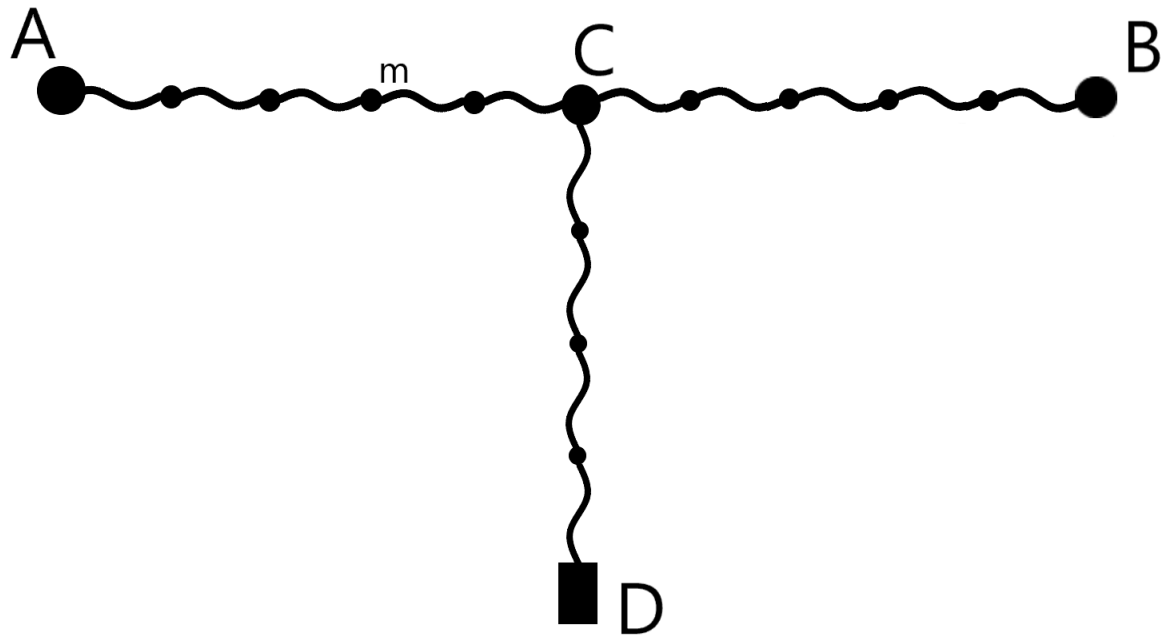


Рис. 1.1: Схематичне зображення системи.

Наша модель являє собою дискретне представлення реальної мотузкової системи. Це означає, що кожна з мотузок розглядається як ланцюжок з'єднаних між собою малих точкових мас m_i. Вважаємо, що маса мотузки розподілена між точками рівномірно, а також положення точок розподілене рівномірно. Останнє означає, що у стані спокою або при постійному рівні напруги між будь-якою парою точок зберігається постійна відстань. Зауважимо, що у реальній мотузці сили напруження виникають тільки під час розтягнення, а при вигині чи стисненні вона не створює жодного спротиву. Ця властива мотузкам поведінка відрізняє її від поведінки пружини, тому що у останній виникають сили напруги навіть під час деформації стиснення. Аналітично це можна виразити для деякого фрагмента мотузки між точковими масами як відсутність сили,

тобто:

$$\vec{F}_{i,i+1} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \text{ – якщо } l_{i,i+1}(t) - l_0^1 < 0.$$

Вважаємо, що у початковий момент часу базова мотузка має попередній натяг, а мотузка-поводок не має натягу, але вона витягнута вздовж вектикальної прямої на свою природну недеформовану довжину (тобто має довжину L_2). При цьому варто зазначити, що початкові швидкості точок бази дорівнюють нулю, а точки мотузки-поводка мають деяку початкову швидкість. Довільна точка останньої мотузки повинна мати тим більше значення швидкості, чим нижче вона знаходиться (відносно вертикальної осі координат).

Добре відомо, що будь-яка мотузка із закріпленими кінцями у полі сил тяжіння, що знаходиться у стані рівноваги має статичне провисання. Це означає, що навіть за відсутності сил, які змушують мотузку рухатися, у стані спокою вона має вигин, причому крива, яку у такому положенні описує мотузка має наступне рівняння у декартовій системі координат:

$$y = a \cosh\left(\frac{x}{a}\right).$$

Це так звана ланцюгова лінія або катенарія. У нашому випадку ми вважаємо, що за рахунок натягнення базової мотузки цього провисання немає (точніше, воно настільки мале, що ми їм нехтуємо) і вона має форму відрізка горизонтальної прямої, з кінцевими точками **A** і **B**. У рівняннях цей факт ми виразимо відсутністю доданка, що відповідає силі тяжіння для усіх точок базової мотузки. Тоді цей доданок буде додаватися тількі у рівняннях мотузки-поводка. У реальній мотузковій системі можна зустріти два види натягнення базової мотузки. У першому випадку натягнення відбувається у обох кінцевих точках в протилежних напрямках, а у другому – в одній з кінцевих точок, а в іншій точці мотузка просто закріплена. Зазвичай мотузку натягують за допомогою блока (котушки). Під час тертя мотузки з блоком при русі системи виникає сила дисипації (втрати енергії). У роботі [2] в рівняннях враховується таке тертя. Ця сила описується рівнянням Ейлера-Ейтельвейна (або рівнянням тертя ременя). Вивід рівняння можна знайти у літературі з класичної механіки (див. [7], с. 26–27) і з теоретичної механіки (див. [8], с. 45-47).

Також зробимо додаткове зауваження, що при русі системи ми нехтуємо силою тертя вантажа з повітрям. При достатньо великих значеннях t тертя тіла із середовищем може вносити суттєвий вклад і гальмувати швидкість руху. Зазвичай, зовнішнє тертя

враховують, додаючи до рівняння доданки, що залежать від швидкості руху вантажа (наприклад $\dot{x}^2$).

Додамо, що у реальних мотузкових системах зазвичай розглядається ситуація, коли точка кріплення мотузок є рухомою відносно базової мотузки. Це означає, що точка **C** є роликом, що рухається під час навантаження мотузки-поводка по базовій мотузці і таким чином сповільнює швидкий вертикальний рух вантажа.

Враховуючи вказані зауваження, приходимо до висновку, що на систему накладені тільки геометричні в'язі.

1.2.2. Виведення рівнянь руху

Кожний фрагмент обох мотузок ми розглядаємо як паралельно з'єднані пружину та демпфер. Як відомо, при такому з'єднанні деформація однакова для обох елементів, а загальна напруга дорівнює сумі напруг на окремих елементах. Це означає, що сила яка діє між парою точок дорівнює сумі пружньої та в'язкої складових:

$$\vec{F}_{i,i+1} = \vec{F}^{\text{пруж.}}_{i,i+1} + \vec{F}^{\text{в'яз.}}_{i,i+1}. \tag{1.1}$$

Знайдемо рівняння для пружньої та в'язкої сил для фрагмента мотузки між точками m_i та m_{i+1}. Запишемо їх для випадку фрагмента базової мотузки, оскільки для фрагментів мотузки-поводка вони виглядатимуть аналогічно. Згідно з лінійним законом пружності Гука у двовимірному випадку пружина подовжується у напрямку, що задається одиничним вектором $\vec{e}_{i,i+1}$. Тому отримуємо наступний вираз для сили пружності:

$$\vec{F}^{\text{пруж.}}_{i,i+1} = \left(\frac{l_{i,i+1}(t) - l^1_0}{l^1_0}\right) k_1 \vec{e}_{i,i+1}, \tag{1.2}$$

$\left(\frac{l_{i,i+1}(t) - l^1_0}{l^1_0}\right)$, k_1 – відносне подовження пружини та її жорсткість відповідно. Згідно з нашими позначеннями та зауваженням наведеним вище щодо розподілу точок по мотузці отримуємо наступне значення для довжини фрагмента бази у стані спокою: $l^1_0 = \dfrac{L_1}{n_1 + 1}$.

В'язка складова також повинна бути напрямлена вздовж вектора $\vec{e}_{i,i+1}$. Введемо додаткове позначення $\vec{f}_{i,i+1} = \eta_1(\dot{\vec{r}}_{i+1} - \dot{\vec{r}}_i)$. Проектуємо вектор $\vec{f}_{i,i+1}$ на напрямок одиничного вектора та ділимо на довжину нерозтягненого фрагмента:

$$\vec{F}^{\text{в'яз.}}_{i,i+1} = \frac{(\vec{e}_{i,i+1} \cdot \vec{f}_{i,i+1})}{l^1_0} \vec{e}_{i,i+1}. \tag{1.3}$$

Вираз у дужках вище означає скалярний добуток векторів. Відстань між точками (тобто довжина фрагмента мотузки у момент часу t) розраховується як звичайна евклідова норма вектора:

$$l_{i,i+1}(t) = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} = \|\vec{r}_{i+1} - \vec{r_i}\|. \tag{1.4}$$

Таким чином, знаходимо вектор сили $\vec{F}_{i,i+1}$ як суму виведених вище пружньої на в'язкої сил. Аналогічно знаходиться вираз для сили $\vec{F}_{i,i-1}$, яка діє на точкову масу m_i з боку точки m_{i-1}.

На малюнку нижче можна побачити схематичне зображення двох фрагментів мотузки, що сполучені точкою m_i. До цієї точки прикладено дві сили, кожна з яких направлена вздовж відповідних фрагментів.

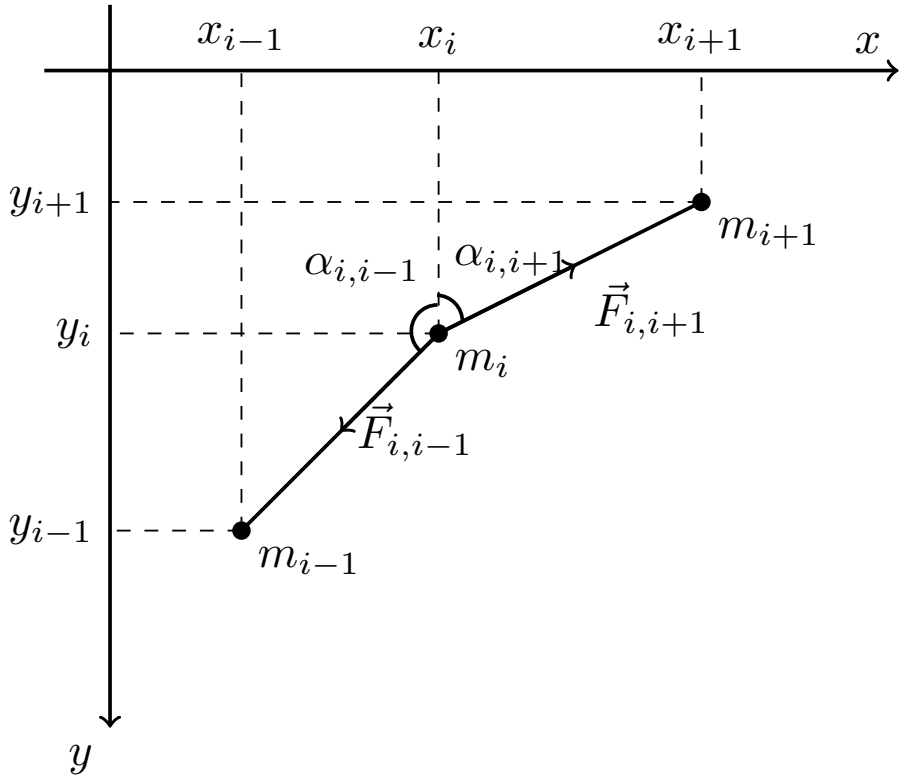


Рис. 1.2: Два сполучені фрагменти мотузкової системи розглянуті в процесі навантаження.

Тепер ми можемо знайти вектор загальної сили $\vec{F}_i$, що діє на точку:

$$\vec{F}_i = \vec{F}_{i,i+1} + \vec{F}_{i,i-1} = \vec{F}^{\text{пруж.}}_{i,i+1} + \vec{F}^{\text{в'яз.}}_{i,i+1} + \vec{F}^{\text{пруж.}}_{i,i-1} + \vec{F}^{\text{в'яз.}}_{i,i-1}. \tag{1.5}$$

Скористуємось другим законом Ньютона і отримаємо остаточне рівняння у векторному вигляді:

$$m_i \ddot{\vec{r}}_i = \vec{F}_i. \tag{1.6}$$

Тепер можна спроектувати вектор сили на осі координат і таким чином отримаємо систему рівнянь для окремих координат точки:

$$\begin{aligned} m_i \ddot{x}_i &= F_i^x, \\ m_i \ddot{y}_i &= F_i^y. \end{aligned} \tag{1.7}$$

Згідно з третім законом Ньютона справедлива наступна рівність для будь-яких пар

точок: $\vec{F}_{i,i+1} = -\vec{F}_{i+1,i}$.

Виведені вище рівняння руху є справедливими для усіх точок базової мотузки. Для точок мотузки-поводка окрім точки з вантажем **D** та точки кріплення мотузок **C** рівняння мають наступний вигляд:

$$\begin{aligned} m_i\ddot{x}_i &= F_i^x, \\ m_i\ddot{y}_i &= F_i^y + m_i\|\vec{g}\|. \end{aligned} \tag{1.8}$$

.

У рівняннях для опису сили змінюються відповідно коефіцієнти жорсткості (k_2 замість k_1), в'язкості (η_2) та довжина фрагмента (l_0^2). Додамо, що для точок мотузки-поводка взагалі кажучи не повинні змінюватися координати точок по осі Ox. З фізичної точки зору це означає, що у цій мотузці не розповсюджуються поперечні хвилі, а тільки поздовжні. Наведемо уточнення для вигляду рівнянь у зазначених точках.

Розглянемо спочатку точку **C**. Оскільки ми вважаємо, що ця точка розташована у центрі базової мотузки, то для дискретизованої моделі це означає, що зліва та справа від неї однакова кількість точок, з чого випливає, що n_1 є непарним числом. Таким чином індекс цієї точки дорівнює $C_{indx} = \left\lfloor \frac{n_1}{2} \right\rfloor$. Тоді для сусідніх точок, які знаходяться зліва і справа від неї маємо індекси $C_{indx-1} = \left\lfloor \frac{n_1}{2} \right\rfloor - 1$ та $C_{indx+1} = \left\lfloor \frac{n_1}{2} \right\rfloor + 1$. Тоді індекс точки мотузки-поводка з'єднаної з точкою **C** дорівнює $n_1 + 1$.

Точка **C** відрізняється від усіх інших точок системи саме тим, що до неї прикладено три сили замість двох. Згідно із третім законом Ньютона вони повинні врівноважувати себе. На малюнку нижче зображені ці три сили.

Також рівняння відрізняються для точки **D**. У цьому випадку немає сили, що діє на точку знизу, тому що ця точка є останнью у ланцюжку.

Наведемо рівняння для цих двох точок нижче.

$$m_C\ddot{\vec{r}}_C = \vec{F}_C = \vec{F}_{\lfloor \frac{n_1}{2} \rfloor, \lfloor \frac{n_1}{2} \rfloor + 1} + \vec{F}_{\lfloor \frac{n_1}{2} \rfloor, \lfloor \frac{n_1}{2} \rfloor - 1} + \vec{F}_{\lfloor \frac{n_1}{2} \rfloor, n_1+1}. \tag{1.9}$$

Замінимо для краткості відповідні індекси точок на $C, C-1, C+1$ і запишемо рівняння для кожної з сил:

$$\vec{F}_{\lfloor \frac{n_1}{2} \rfloor, \lfloor \frac{n_1}{2} \rfloor - 1} = \left(\frac{(l_{C,C-1} - l_0^1)k_1 + \eta_1(\vec{e}_{C,C-1} \cdot (\dot{\vec{r}}_{C-1} - \dot{\vec{r}}_C))}{l_0^1} \right) \vec{e}_{C,C-1}, \tag{1.10}$$

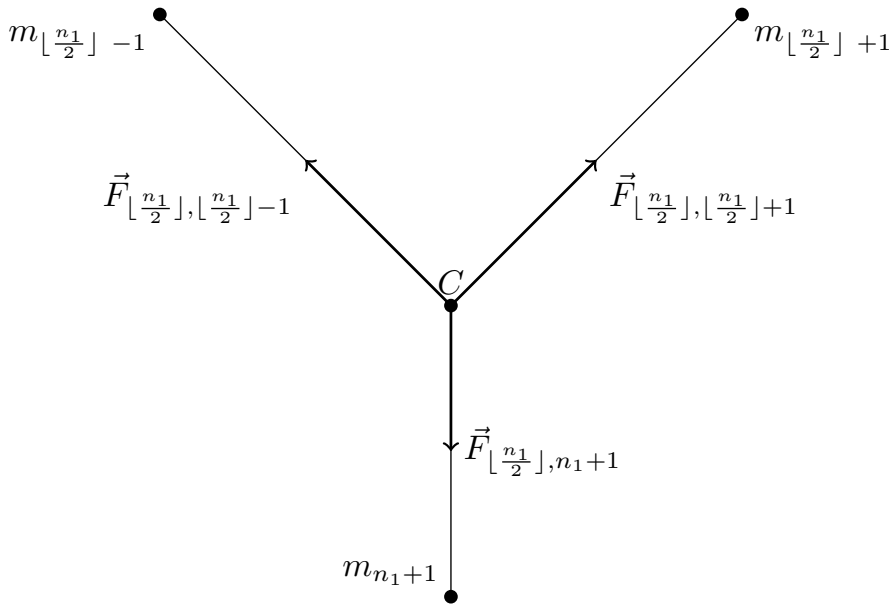


Рис. 1.3: Сили прикладені до точки кріплення мотузок.

$$\vec{F}_{\lfloor \frac{n_1}{2} \rfloor, \lfloor \frac{n_1}{2} \rfloor +1} = \left(\frac{(l_{C,C+1} - l_0^1)k_1 + \eta_1(\vec{e}_{C,C+1} \cdot (\dot{\vec{r}}_{C+1} - \dot{\vec{r}}_C))}{l_0^1} \right) \vec{e}_{C,C+1}, \tag{1.11}$$

$$\vec{F}_{\lfloor \frac{n_1}{2} \rfloor, n_1+1} = \left(\frac{(l_{C,n_1+1} - l_0^2)k_2 + \eta_2(\vec{e}_{C,n_1+1} \cdot (\dot{\vec{r}}_{n_1+1} - \dot{\vec{r}}_C))}{l_0^2} \right) \vec{e}_{C,n_1+1}. \tag{1.12}$$

Рівняння для точки **D**:

$$m_D \ddot{\vec{r}}_D = \vec{F}_D = \vec{F}_{N,N-1}, \tag{1.13}$$

$$\vec{F}_{N,N-1} = \left(\frac{(l_{N,N-1} - l_0^2)k_2 + \eta_2(\vec{e}_{N,N-1} \cdot (\dot{\vec{r}}_{N-1} - \dot{\vec{r}}_N))}{l_0^2} \right) \vec{e}_{N,N-1}. \tag{1.14}$$

1.3. Дослідження мотузкових систем

У цьому підрозділі наведемо порівняльний аналіз досліджень мотузкових систем у різних задачах.

У вже згадуваній роботі [1] розглядалась модель одновимірного вертикально направленого ланцюжка точкових мас, з'єднаних між собою пружинами із вантажем на вільному кінці. Ця механічна система є схожою на нашу з декількома відмінностями. По-перше, у цій системі мотузка описується тільки пружинами без врахування паралельно під'єднаних демпферів, а по-друге, у нашій моделі присутня друга мотузка, що сповільює вертикальний рух першої. Фактично, якщо не враховувати демпфер (тобто прирівняти коефіцієнт в'язкості до нуля), то зазначена у авторів модель є мотузкою-поводком у нашій моделі з поправкою, що у нашій моделі рухомими є обидва кінці.

У роботі [9] було розглянуто тросову систему, яка використовується для вирішення задачі захоплення у повітрі літаком іншого літаючого об'єкта (в даному випадку крилатої ракети). Трос у цій задачі одним кінцем прикріплений до літака-буксира, а на вільному кінці до нього закріплений спеціальний керований пристрій, який з'єднується з ракетою. Схожість задачі з нашим випадком полягає у тому, що при русі нашої системи, обидва кінці мотузки-поводка рухаються і до одного з них також прикріплено масивний (відносно маси мотузки) вантаж. Відмінність полягає в тому, що у цій роботі автори розглядають трос як ланцюг з'єднаних жорстких тіл і у точках з'єднання містяться спіральні обертальні пружини. Тобто у такій системі спротив виникає під час вигину (або скручування) троса. Це суттєво відрізняється від поведінки звичайної мотузки, яка зазвичай складається зі сплетіння тонших ниток більш еластичних матеріалів, таких як поліестр та поліаміди.

У роботах [10], [11] також моделюється поведінка мотузкових систем. У першій з них розглядається модель ланцюга (або мотузки) як набір послідовно з'єднаних жорстких сегментів. Один кінець цього ланцюга фіксований, а інший вільно падає під дією сили тяжіння. Досліджується динаміка руху вільно падаючого кінця та передача енергії по ланцюгу. Були проведені лабораторні експерименти та чисельне моделювання. У цій роботі автори намагалися пояснити парадоксальне прискорення вільного кінця ланцюга, яке досягає значення більшого ніж прискорення вільного падіння. Результати отримані у цій роботі та постановка задачі можуть бути корисними для наших досліджень, оскільки у нашому випадку також розглядається вільне падіння мотузки-поводка з вантажем. Відмінність з нашим випадком полягає в тому, що у нас мотузка

витягнута вертикально у початковий момент часу, а у роботі авторів мотузка займає деяке положення і має форму вже згадуваної катенарії. Таким чином, постановку задачі Коші у нашому випадку також можна змінювати, задаючи положення вільного кінця мотузки-поводка аналогічно до того, як це було зроблено у згадуваній роботі. Це дозволяє розширити та узагальнити наше дослідження на випадок іншого положення вантажа і врахувати провисання мотузки-поводка у початовий момент часу. У реальних роупджампінгових системах площадка, з якої падає людина зазвичай знаходиться дещо збоку відносно точки кріплення мотузок. Також у авторів один з кінців фіксований (а у нас обидва кінці рухомі), а тому якщо змінити початкові умови в нашій задачі, то можна врахувати більш складну поведінку системи. У другій роботі мотузка також розглядається як дискретизований ланцюжок жорстких тіл з шарнірними з'єднаннями. Проводяться експерименти з двома типами обмежень. У першому з них, один кінець фіксований, а у другому випадку цей кінець рухається за деякою заданою траєкторією (наприклад, періодичні коливання з заданою частотою та амплітудою). Другий кінець у цих експериментах може бути вільним, але не обов'язково. Автори провели комп'ютерне моделювання поведінки такої системи з порівнянням різних чисельних методів. Приділяється увага порівнянню динаміки руху при різній кількості точок дискретизації. Досліджується вплив руху закріпленого кінця, який призводить до виникнення складних хвильових процесів у мотузці, включаючи вимушені коливання та хаотичні ефекти. Система відрізняється від нашої тим, що ланцюжок складається з твердих недеформівних тіл і один з кінців рухається за деяким відомим наперед законом. Спільним є те, що автори також у чисельних експериментах розглядають випадки різної кількості точок у системі та досліджують розповсюдження хвиль.

Отже, ми зробили огляд сучасних наукових робіт по темі і порівняли їх з нашою моделлю. У нашому випадку система складається з пружньо-в'язких елементів, тому що на нашу думку мотузка має як пружні так і в'язкі властивості. У наведених роботах мотузки розглядаються як послідовність з'єднаних твердих тіл, що більше підходить для опису металевих тросів або ланцюгів.

1.4. Опис моделі у термінах рівнянь з частинними похідними

Наша модель може бути розглянута з точки зору рівнянь у частинних похідних. Зауважимо, що такий спосіб опису динаміки руху допускається в припущенні, що базова мотузка здійснює **малі** коливання. У цьому підрозділі ми лише наведемо можливі приклади постановки початково-крайових умов у задачі, але не будемо наводити аналітичний або чисельний розв'язок.

З курсу рівнянь математичної фізики або диференціальних рівнянь у частинних похідних добре відомим є хвильове рівняння. Це класичне рівняння розглядається для опису розповсюдження поперечних хвиль у таких середовищах, як наприклад, струна або мембрана у багатовимірному випадку. Вивід цього рівняння і методи його розв'язання можна знайти у літературі по рівнянням з частинними похідними (рекомендуємо подивитися на чудову книгу [12], у якій також наведено випадок, коли є супротив до руху мотузки, тобто коли мотузка рухається, наприклад у рідині).

Введемо деякі позначення. Нехай $u_1(x,t)$ – функція, яка означає відхилення від горизонтального положення точки на базовій мотузці з координатою x у момент часу t. В залежності від моменту часу для кожної точки це відхилення змінюється. Тоді

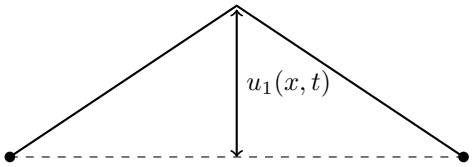


Рис. 1.4: Функція відхилення базової мотузки.

хвильове рівняння для базової мотузки виглядає наступним чином:

$$\frac{\partial^2 u_1}{\partial t^2} = c_1^2 \frac{\partial^2 u_1}{\partial x^2}, \tag{1.15}$$

де c_1 – коефіцієнт, що має сенс швидкості розповсюдження хвилі.

Тепер перейдемо до рівняння, що описує рух вертикально направленої мотузки-поводка. Рівняння для цієї мотузки буде виглядати аналогічним чином, але сенс функції u_2 змінюється. У цьому випадку функція $u_2(y,t)$ характеризує поздовжні хвилі, тобто описує вертикальні відхилення точок. Для виводу цього рівняння поглянемо на нашу мотузку знову як на ланцюжок малих точкових мас, з'єднаних пружними тілами. Тоді рівняння, що описують рух такої системи мають наступний вигляд (ця система рівнянь

також наведена у роботі [1], ми наведемо загальний вигляд рівняння для i-тої маси):

$$m_i\ddot{u}_i = -k(u_i - u_{i-1}) - k(u_i - u_{i+1}) + m_i g, \tag{1.16}$$

де u_i – позначає зміщення тіла m_i з положення рівноваги, k – жорсткість пружинок. Вважаємо, як і раніше, що для всіх точок значення m_i однакове.

Поділивши на масу і перегруповуючи доданки, отримаємо вигляд:

$$\ddot{u}_i = \frac{k}{m_i}(u_{i+1} - 2u_i + u_{i-1}) + g. \tag{1.17}$$

Нехай $l_0 = \Delta y$ – довжина кожної з пружинок (відстань між тілами) у стані спокою. Вважаємо, що зміщення тіл u_i з положення рівноваги є меншою за відстань l_0. Тоді при прямуванні кількості точкових мас до нескінченності (і відповідно, прямуванні $l_0 \to 0$) відносне подовження приблизно дорівнює похідній по координаті:

$$\frac{u_i - u_{i-1}}{l_0} \approx \frac{u_{i+1} - u_i}{l_0} \approx \frac{\partial u}{\partial y}.$$

З цього отримуємо вираз для похідної другого порядку:

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{l_0^2} \approx \frac{\partial^2 u}{\partial y^2}.$$

Звертаючи увагу на вираз у правій частині рівняння (1.17), зробивши вказаний граничний перехід і підставляючи у це рівняння наведену апроксимацію, отримуємо одновимірне хвильове рівняння поздовжних хвиль у вертикальній мотузці-поводку:

$$\frac{\partial^2 u_2}{\partial t^2} = c_2^2 \frac{\partial^2 u_2}{\partial y^2} + g, \tag{1.18}$$

де $u_2(y,t)$ – функція поздовжного відхилення, що залежить від вертикальної координати та часу, $c_2 = \sqrt{\dfrac{k}{m}} l_0$, g – прискорення вільного падіння на Землі.

Також це рівняння застосовують для описання поздовжних хвиль, що виникають у еластичному стрижні (наприклад, у балці). Якщо розглянути мотузку з точки зору циліндричного стрижня з високою пружністю, то його можна вивести також на основі другого закону Ньютона. На малюнку нижче стрілками позначені напрямки дії напруження у середовищі, два поперечних перерізи. Тоді елемент середовища між цими перерізами має масу:

$$m = \rho V = \rho S dx,$$

де V – об'єм середовища між перерізами, S – площа поперечного перерізу, ρ – щільність середовища, $dx = \Delta x$ – відстань між перерізами.

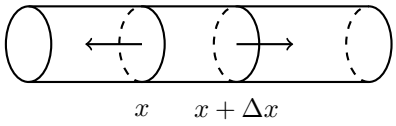


Рис. 1.5: Циліндричне пружнє середовище.

Механічне напруження σ дорівнює відношенню сили F, що діє на переріз до площі цього перерізу S. А силу, можна виразити як суму напружень на перерізах x та $x+\Delta x$, які направлені в протилежних напрямках:

$$S(\sigma(x + \Delta x) - \sigma(x)).$$

Тепер, скористуємось наступним виразом:

$$\sigma = \varepsilon E,$$

де E – модуль Юнга, ε – відносне подовження. Виразимо відносне подовження через функцію $u(x,t)$, що означає зміщення точки, яка мала у стані спокою координату x у момент часу t:

$$\varepsilon = \frac{\partial u}{\partial x}.$$

І тепер отримуємо рівняння:

$$\rho S dx \frac{\partial^2 u}{\partial t^2} = SE\left(\frac{\partial u(x + \Delta x)}{\partial x} - \frac{\partial u(x)}{\partial x}\right),$$

а це рівняння простими алгебраїчними перетвореннями зводиться до хвильового:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \tag{1.19}$$

де коефіцієнт має наступний вигляд $c = \sqrt{\dfrac{E}{\rho}}$.

Повернемось до нашої системи рівнянь (1.15), (1.18). Для її розв'язку необхідно задати початкові та граничні умови. Наведемо спочатку ці умови для базової мотузки. Вважаємо, що в початковий момент часу вона є недеформованою та нерухомою, з цього

отримуємо пару початкових умов:

$$u_1(x,0) = 0, \quad \frac{\partial u_1}{\partial t}(x,0) = 0. \tag{1.20}$$

Опис нашої моделі у такий спосіб дозволяє нам також врахувати і статичне провисання, про яке було написано раніше. Це можна виразити у початковій умові на функцію:

$$u_1(x,0) = a\cosh\left(\frac{x}{a}\right),$$

де a – константа.

Оскільки кінці мотузки є фіксованими на одному рівні, тому граничні умови мають вигляд:

$$u_1(0,t) = u_1(L_1,t) = 0. \tag{1.21}$$

Узагальнити граничні умови можна на випадок, коли один із фіксованих кінців знаходиться вище іншого:

$$u_1(0,t) = A > 0, \quad u_1(L_1,t) = 0 \quad \text{або} \quad u_1(0,t) = 0, \quad u_1(L_1,t) = B > 0.$$

Тепер запишемо початкові та граничні умови для хвильового рівняння мотузки-поводка. Ми вважаємо, що мотузка витягнута на свою природну довжину вертикально, але натягу немає. При цьому ми також вважаємо, що тіло в початковий момент часу вже має деяку ненульову швидкість направлену вертикально вниз. Цю швидкість можна вважати рівною швидкості вільно падаючого тіла:

$$v_0 = \sqrt{2gh},$$

h – висота падіння, g – прискорення вільного падіння. Таким чином, початкові умови мають вигляд:

$$u_2(y,0) = 0, \quad \frac{\partial u_2}{\partial t}(y,0) = \sqrt{2gh}. \tag{1.22}$$

Оскільки початкові умови є функціями лише від координати y, то можна вважати, що в правій частині умови на швидкість $h = y$. Тоді швидкість зміщення деякої точки мотузки з координатою y тим більша, чим нижче знаходиться ця точка. Тобто рівність $h = y$ означає, що початкова висота, з якої падає окрема точка мотузки має значення симетричне до її координати в момент часу $t = 0$ (або що мотузка була витягнута вгору

на натуральну довжину і відпущена, а момент відліку часу починається тоді, коли швидкість вільного падіння дорівнює v_0). Крайові умови для цієї мотузки виглядають дещо складніше, тому що обидва кінці тут є рухомими. По-перше, повинна виконуватися умова узгодженості положення верхнього закріпленого кінця цієї мотузки з серединою базової мотузки (тому що це одна і та сама точка). Отже, отримуємо:

$$u_2(0,t) = u_1\left(\frac{L_1}{2}, t\right). \tag{1.23}$$

Крайову умову на вільному кінці мотузки можна розкласти на суму двох складових. Одна з них, це зміщення закріпленого кінця мотузки, а друга – функція, що описує рух вантажа на кінці мотузки без врахування зміщення (тобто якщо верній кінець закріплений нерухомо):

$$u_2(L_2,t) = u_1\left(\frac{L_1}{2}, t\right) + f(t). \tag{1.24}$$

Функція $f(t)$ взагалі кажучи залежить (точніше залежить її амплітуда та частота коливань) від маси вантажа M, властивостей мотузки та висоти вільного падіння h. Ця функція задовільняє звичайному диференціальному рівнянню руху вантажа:

$$\ddot{y}_M(t) + \frac{k}{M} y_M(t) = \frac{k}{M} L_2 + g.$$

Це рівняння є неоднорідним рівнянням другого порядку зі сталими коефіцієнтами, а тому його розв'язок має вигляд:

$$y_M(t) = \alpha_1 \cos(\omega t) + \alpha_2 \sin(\omega t) + \beta,$$

де $\alpha_1, \alpha_2, \omega, \beta$ – константи, що визначаються з початкових умов та на основі властивостей мотузки. Отже, остаточний вид крайової умови на вільному кінці мотузки-поводка має наступний вигляд:

$$u_2(L_2,t) = u_1\left(\frac{L_1}{2}, t\right) + \alpha_1 \cos(\omega t) + \alpha_2 \sin(\omega t) + \beta. \tag{1.25}$$

Отже, ми навели загальний вигляд системи рівнянь у частинних похідних (1.15), (1.18), що описує рух нашої мотузкової системи і задали початково-крайові умови для функції $u_1(x,t)$ (1.20), (1.21) та для функції $u_2(y,t)$ (1.22), (1.23), (1.25). Підкреслимо, що при такому описі нашої системи для базової мотузки ми описуємо лише поперечні коливання, але не враховуємо поздовжні, а для мотузки-поводка, ми враховуємо тільки поздовжні.

Зауважимо, що такий опис підходить лише для малих коливань у системі. Виділимо, що позначення у цьому пункті не пов'язані з тими, що будуть введені у наступному розділі. Для опису одночасно і поперечних, і поздовжних хвиль у базовій мотузці у термінах рівнянь математичної фізики потрібно розглядати не лише нормальну складову напруження (напрямлену вздовж мотузки), що діє на її поперечний переріз, а і дотичну складову, що призведе до розгляду такого поняття як тензор напружень. У зв'язку з цим виникне ще більш складна система диференціальних рівнянь з частинними похідними. Схожі до такої системи зазвичай розглядаються у курсах механіки суцільних середовищ. Побудова такої загальної системи рівнянь, її вивчення, та пошук чисельних або аналітичних розв'язків можуть слугувати предметом майбутніх досліджень.

Розділ 2. Комп'ютерне моделювання

2.1. Постановка комп'ютерних експериментів

У цьому розділі буде детально описано комп'ютерне моделювання руху нашої системи та чисельні експерименти. Код проекту було написано на мові програмування Python версії 3.12 [13]. При побудові програми були використані бібліотеки зі середовища цієї мови, такі як Numpy [14] для матрично-векторних операцій, Matplotlib [15] для візуалізації графіків, Pygame [16] для малювання системи у кожен окремий момент часу на екрані, Sciру [17] для чисельного інтегрування системи диференціальних рівнянь (див. друге посилання на документацію відповідної функції).

Зазначимо, що чисельні методи для розв'язання систем звичайних диференціальних рівнянь розроблені для розв'язання саме рівнянь першого порядку. У нашому ж випадку (як і в інших задачах механіки) завдяки використанню другого закону Ньютона ми отримали систему другого порядку. Таку систему ми не можемо напряму подавати на вхід функції-розв'язувачу. Тому необхідно перевести кожне рівняння вихідної системи, яка була розглянута у попередньому розділі, у вигляд системи рівнянь першого порядку. Таким чином, якщо ми маємо N – загальну кількість точок у моделі, тоді отримуємо $2N$ рівнянь другого порядку, тобто по два рівняння для однієї точки (по одному рівнянню для кожної з координат). При заміні змінних отримаємо систему з $4N$ рівнянь першого порядку.

Сучасні чисельні методи будуються на основі ідеї добре відомого методу Ейлера. Розглянемо цей метод для випадку задачі Коші для одного рівняння. Нехай задане рівняння з початковою умовою наступного вигляду:

$$\dot{x} = f(x,t)$$
$$x(t_0) = x_0.$$

Функція $f(x,t)$ визначена на деякій множині $D \subset R^2$. Розв'язок вказаного рівняння шукатимемо на деякому проміжку часу $(t_0, T]$. Задамо деяке розбиття цього напівінтервала: $t_0 < t_1 < \cdots < t_n \leq T$. Виразивши наближене значення похідної функції $x(t)$ як відношення приросту значень функції в точках до приросту значень аргументу t:

$$\frac{x(t_{i+1}) - x(t_i)}{t_{i+1} - t_i} \approx f(x_i, t_i),$$

отримаємо ітеративну формулу для знаходження лінійно-кускового наближення розв'язку рівняння:

$$x_{i+1} = x(t_{i+1}) = x(t_i) + (t_{i+1} - t_i) f(x_i, t_i), \quad i = 1, 2, \ldots, n-1.$$

Таким чином, на першому кроці алгоритму ми використаємо значення з початкової умови для знаходження значення відомої функції $f(x,t)$, а далі, використовуючи вже знайдені за домогою цього алгоритму значення x_i, ми знайдемо послідовно наступні значення шуканої функції у точках.

Цей метод є простим та інтуїтивно зрозумілим, а також дає відносно непогану точність на малих проміжках часу з великою кількістю точок розбиття інтервалу. Але при достатньо великих t та складному вигляді функції правої частини $f(x,t)$ цей метод має велику похибку, а тому при чисельному моделюванні фізичних явищ, що описуються диференціальними рівняннями використовують складніші методи більш високого порядку точності. Основним сімейством чисельних методів, що дають більшу точність є так звані методи Рунге-Кутти. Класичний метод Ейлера, а також детальне обґрунтування інших методів можна знайти у відповідній літературі з чисельних методів (див. главу 25 у книзі [18], главу 5 у підручнику [19]). Також одними з найпоширеніших проблем чисельних методів є їхня нестійкість та висока чутливість до початкових умов. Це все може погіршувати точність розв'язку та призводити до неочікуваної поведінки системи. Тому варто обирати високоточні та стійкі методи, які добре апроксимують розв'язок рівняння навіть при великих значеннях t. У наших експериментах було обрано два методи: метод Рунге-Кутти порядку 5(4) [20] та Radau (посилання на цей метод можна знайти у документації бібліотеки Scipy [17]). Вибір методів регулюється при передачі відповідних аргументів 'RK45' або 'Radau' до функції `solve_ivp` зазначеної

бібліотеки.

Перейдемо до заміни змінних у вихідній системі. Введемо наступні позначення для деякої точкової маси m_i з координатами (x_i, y_i):

$$u_{1,i} = x_i$$
$$v_{1,i} = \dot{x}_i$$
$$u_{2,i} = y_i$$
$$v_{2,i} = \dot{y}_i.$$

Тоді зробивши таку заміну змінних, ми переходимо від системи (1.7) другого порядку до системи наступного вигляду:

$$\begin{cases} \dot{u}_{1,i} = v_{1,i} \\ \dot{v}_{1,i} = \dfrac{F_i^{u_{1,i}}}{m_i} \\ \dot{u}_{2,i} = v_{2,i} \\ \dot{v}_{2,i} = \dfrac{F_i^{u_{2,i}}}{m_i}, \end{cases} \tag{2.1}$$

розв'язавши наступну систему рівнянь, ми отримаємо вектор значень координат та швидкостей точки. Загальна система складається з таких підсистем з 4 рівнянь першого порядку для кожної точки. Тоді на виході з функції-розв'язувача на основі чисельного методу отримаємо вектор довжини $4N$.

Для отримання чисельного розв'язку нам необхідно задати початкові умови. Кожна точка описується набором з чотирьох початкових умов: дві для координат та дві для швидкостей. Для точок базової мотузки у коді програми положення визначались рівномірним розподіленням вздовж горизонтального відрізка з кінцями у фіксованих точках. Позначивши за (x_A, y_A) та (x_B, y_B) координати лівої та правої фіксованих точок відповідно, та вважаючи, що горизонтальна вісь координат проходить вздовж цього відрізка отримуємо початкові умови на координати точок базової мотузки:

$$u_{1,i} = x_A + i\frac{x_B - x_A}{n_1 + 1}, \quad u_{2,i} = 0, \quad i = 1, \ldots, n_1. \tag{2.2}$$

Оскільки у початковий момент часу базова мотузка є нерухомою, то початкові умови на швидкості мають вигляд:

$$v_{1,i} = 0, \quad v_{2,i} = 0, \quad i = 1, \ldots, n_1. \tag{2.3}$$

Оскільки у нашій задачі Коші вважаємо, що мотузка-поводок витягнута на натуральну довжину вздовж осі Oy і відстані між парами сусідніх точок є однаковими, то початкові умови на координати для цієї мотузки мають вигляд:

$$u_{1,i} = \frac{x_A + x_B}{2}, \quad u_{2,i} = i\frac{L_2}{n_2}, \quad i = 1, \ldots, n_2. \tag{2.4}$$

Швидкості для точок мотузки-поводка ми робили лінійно зростаючими зі значеннями від 0 для точки кріплення мотузок до $\sqrt{2gh}$ для точки-вантажа. Горизонтальну складову швидкостей покладено рівними нулю. Тому отримуємо умову на похідні:

$$v_{1,i} = 0, \quad v_{2,i} = i\frac{\sqrt{2gh}}{n_2}, \quad i = 1, \ldots, n_2, \tag{2.5}$$

де h – значення висоти, з якої падає вантаж, що також задається у програмі.

При моделюванні руху ми проводили два види експериментів. У першому з них, задавши початкові умови, ми багаторазово розв'язували задачу Коші до тих пір, поки не буде закрите вікно для відображення руху системи. Тобто система розв'язувалася для деякого інтервалу часу (брали одну секунду і розбивали інтервал на 60 частин, таким чином відображували 60 кадрів з положенням точок системи у секунду). Потім для другого інтервалу часу також розбитого на 60 частин, ми знову знаходили розв'язок задачі Коші, взявши за початкові умови вектор значень координат і швидкостей, отриманих з розв'язку на попередньому інтервалі для останнього моменту часу. Таким чином ітеративно ми могли розв'язувати задачу Коші для довільного проміжку часу, розбивши його на менші проміжки довжиною в 1 секунду, і для кожного з них розв'язавши часткову задачу Коші. У цьому експерименті ми спостерігали еволюцію системи на великому проміжку часу, а отже бачили поступове затухання коливань. У другому експерименті, ми наперед зафіксували кількість секунд, протягом яких нас цікавив розв'язок однієї задачі Коші, а також кількість часових кроків `time_steps_per_second`, на які розбивався інтервал в одну секунду. При відображенні розв'язку такої задачі Коші ми робили затримку, яка дорівнює $\frac{1000}{\texttt{time\_steps\_per\_second}}$. Ми брали значенння для усього інтервалу 30 секунд та 500 кроків, а тому отримували затримку у 2 мілісекунди. Такий підхід дозволяє чітко прив'язати часові рамки експерименту до комп'ютерної моделі, а також описати реальну еволюцію хвильових процесів у часі. Це дозволяє спостерігати без затримок та неперервно за рухом усіх точок. Але цей підхід має недолік, який полягає в тому, що для достаньо великих проміжку часу, кількості кроків та кількості точок дискретизації, програма потребує більшого часу для отримання повного розв'язку на

всьому інтервалі. Але проводячи есперименти ми помітили, що чисельний метод Radau давав розв'язок швидше за метод RK45.

Одним з критеріїв перевірки коректності моделі у цьому випадку є узгодженість експериментів. Тобто в цілому ми повинні отримувати в обох випадках схожу поведінку системи та відповідні графіки сил, що виникають у фрагментах мотузок, координат і швидкостей точок в залежності від часу. Також збільшення точок дискретизації повинне призводити до уточнення руху мотузок та не повинне протирічити поведінці системи при меншій кількості точок. Зауважимо, для відображення руху системи на екрані необхідно задавати відстані між точками у термінах пікселів. Але при розв'язку системи диференціальних рівнянь ми шукаємо у метрах, час вимірюється у секундах, маси в кілограмах, тобто працюємо у системі одиниць СІ.

Під час експериментів з кодом програми ми взяли відстань між точками фіксації рівною 650 пікселів і вважали, що ця відстань відповідає 100 метрам у реальній відстані. Отримали коефіцієнт масштабу 6.5 пікселів на метр. Цей масштаб використовувався для переведення відстаней між точками та швидкостей руху. Додамо, що у піксельній системі координат ліва фіксована точка має координати $(100, 30)$, а у метричній системі – $(0, 0)$. Тому при переведенні з піксельної у метричну систему координат ми спочатку віднімали вектор $(100, 30)$, а потім ділили на значення масштабу.

Нижче наведено характеристики мотузки та дані експериментів:

$$\begin{gathered}
\texttt{horizontal_stiffness_coeff} = 3 \cdot 10^4\ H, \\
\texttt{horizontal_viscous_coeff} = 2 \cdot 10^4\ Hc, \\
\texttt{vertical_stiffness_coeff} = \frac{\texttt{horizontal_stiffness_coeff}}{1.5}\ H, \\
\texttt{vertical_viscous_coeff} = 2 \cdot 10^4\ Hc, \\
\texttt{weighted_point_mass} = 90\ kg, \\
\texttt{vertical_rope_rest_len} = 23.07\ m, \\
\texttt{ropes_mass} = 12\ kg, \texttt{initial_tension_force} = 800H.
\end{gathered} \tag{2.6}$$

Значення початкового натягу базової мотузки ми також змінювали від 600 до 800 Н. Висоту падіння вантажа було покладено 20 м.

2.2. Результати моделювання

Нижче наведено знімки форми мотузки у різні моменти часу.

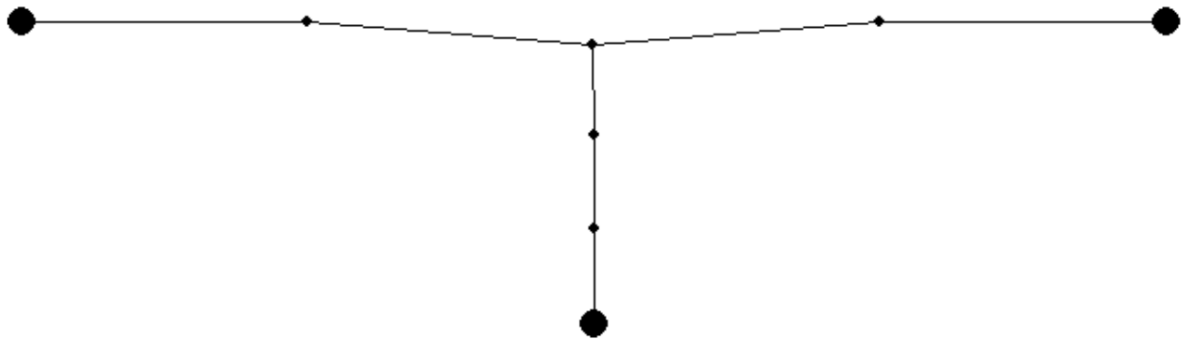


Рис. 2.1: Початок руху, 10-й момент часу ($t = 0.1694$).

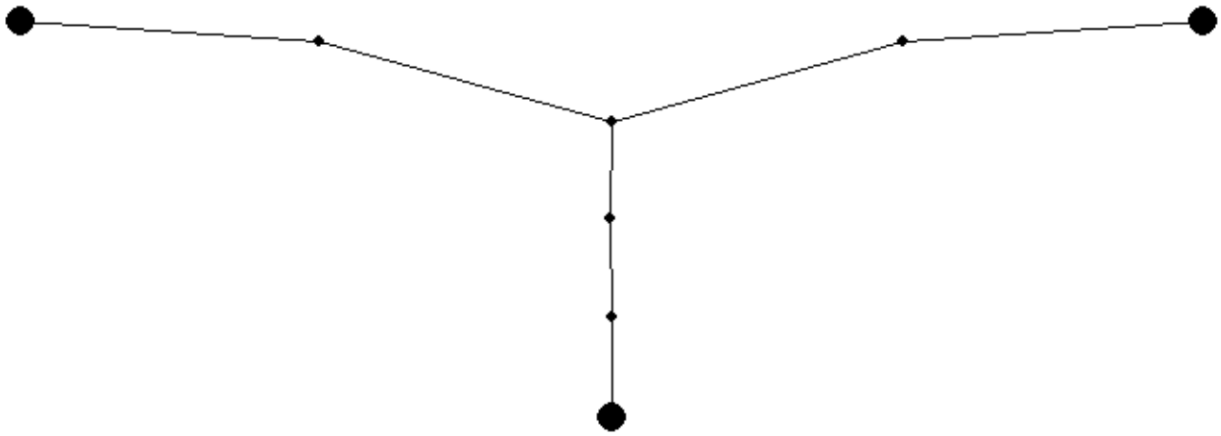


Рис. 2.2: Поступове розтягнення мотузок, 30-й момент часу ($t = 0.5084$).

На цих двох знімках можна спостерігати характерну клювоподібну форму вигину базової мотузки в околі центральної точки. Ми бачимо зростання швидкості падіння, а з ним і поширення поперечних та поздовжних хвиль у системі.

З графіків можна побачити пікову швидкість руху, а відповідно і співпадаючі за часом максимальне відхилення вантажа та максимальне значення модуля сили. Також бачимо з результатів, що при даній простій конфігурації системи приріст координати вантажа має порядок 60 метрів. Схожі графіки було отримано для більш складної конфігурації з більшою кількістю точок дискретизації. По горизонтальній вісі відкладено номери моментів часу. Графіки для інших конфігурацій можна побачити у додадках.

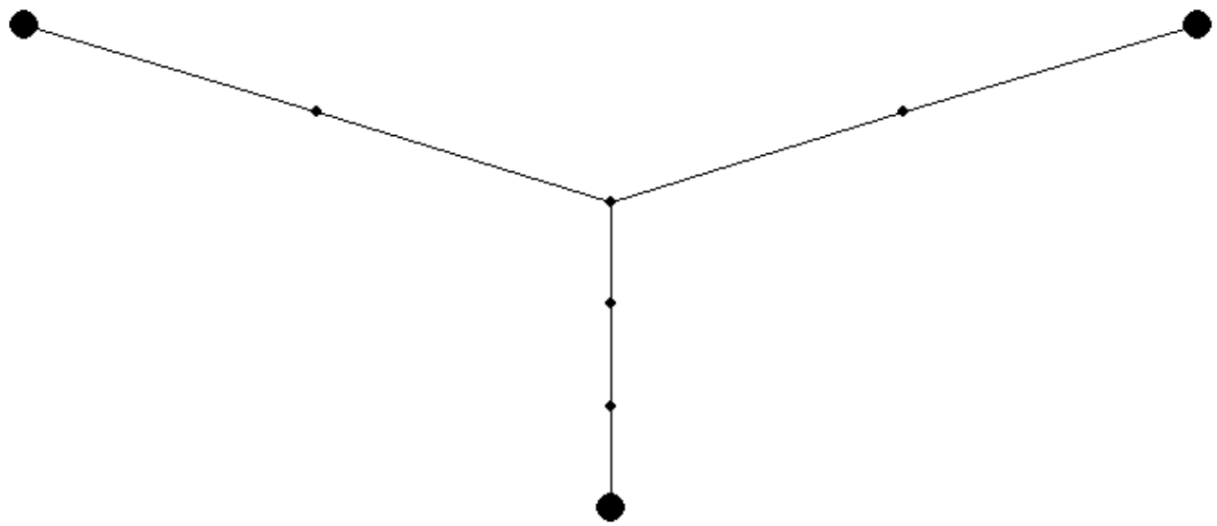


Рис. 2.3: Зростання швидкості, 50-й момент часу.

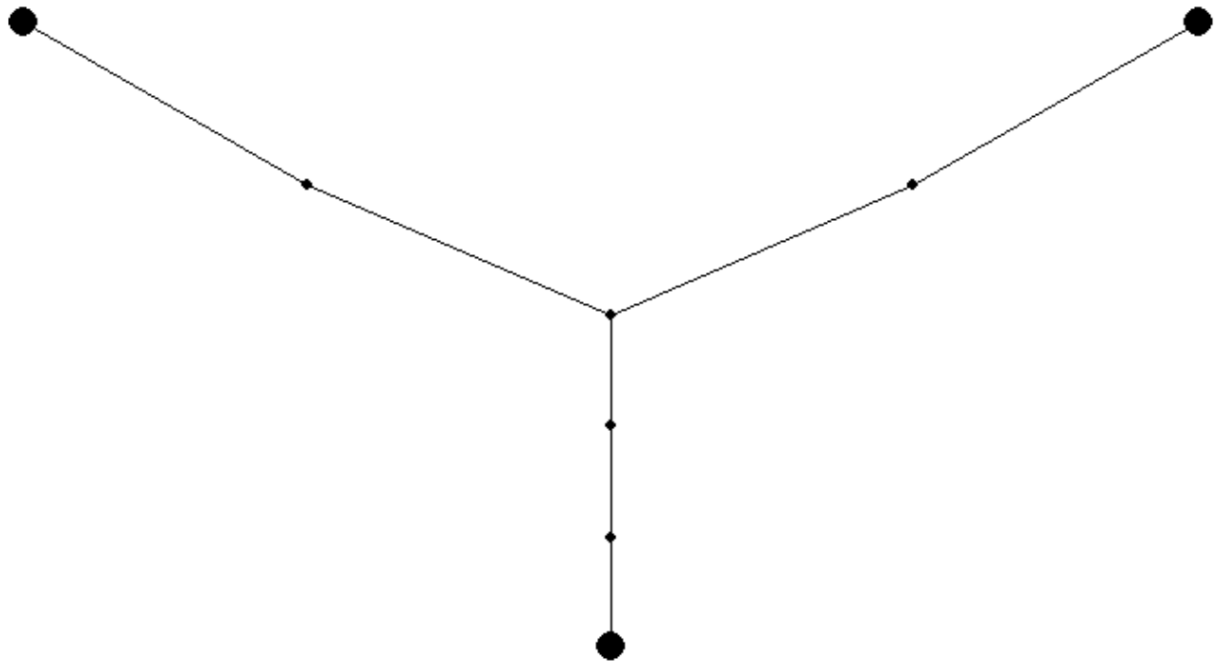


Рис. 2.4: Поширення поперечних хвиль у базовій мотузці, 80-й момент часу.

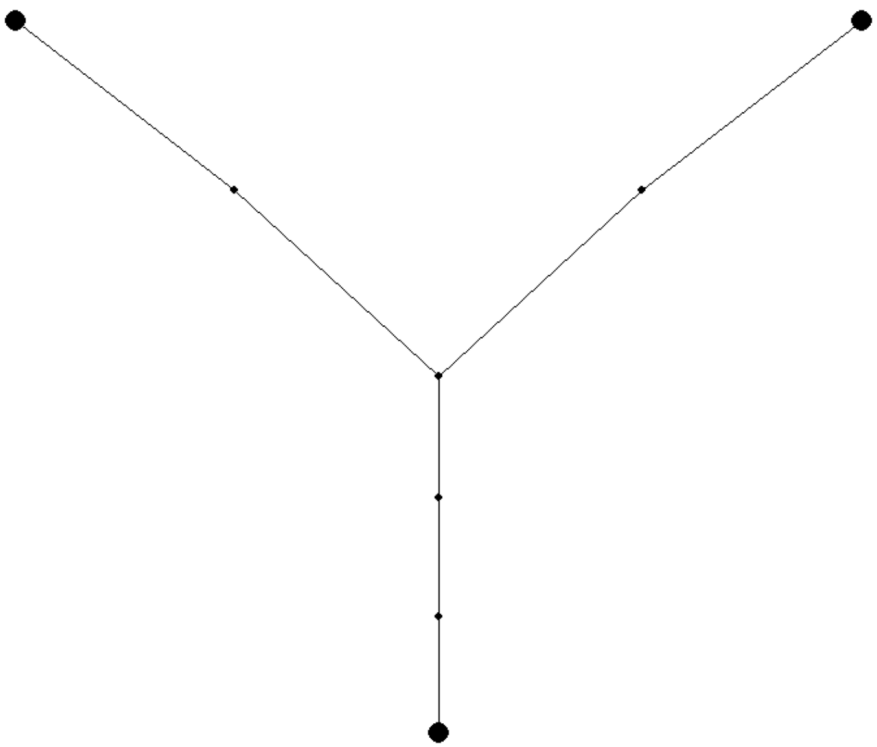


Рис. 2.5: Момент часу, близький до максимального розтягнення мотузки-поводка.

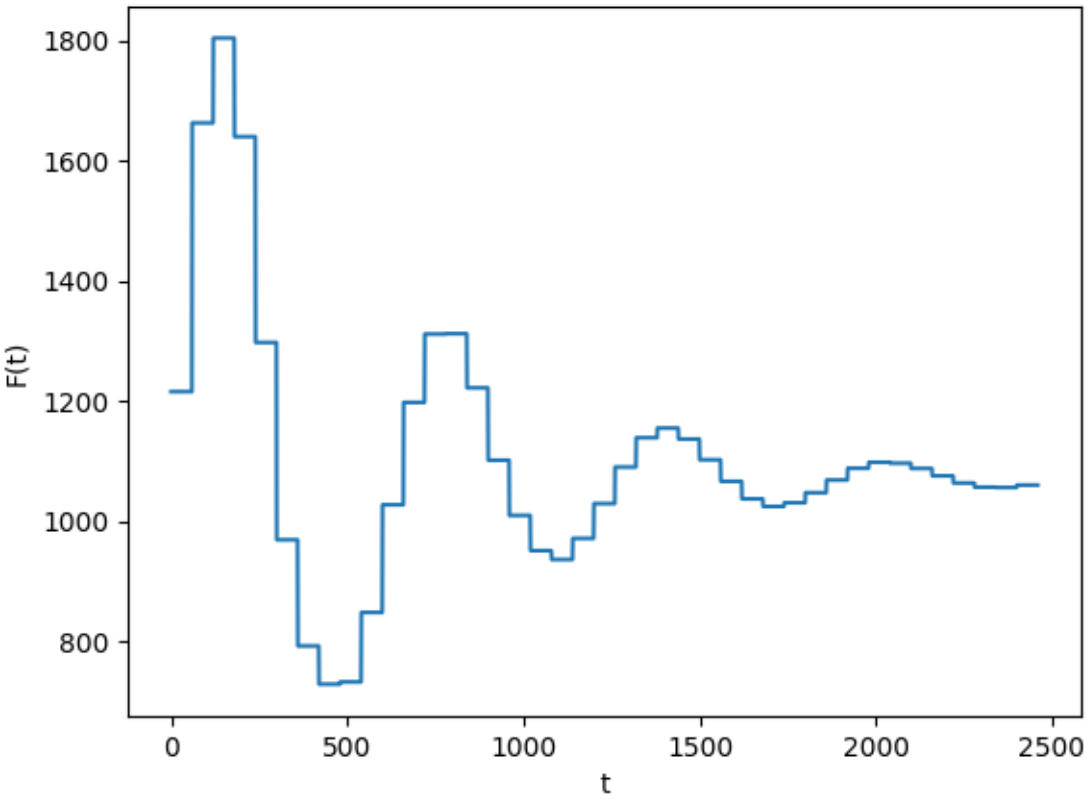


Рис. 2.6: Графік залежності сили у правому крайовому фрагменті мотузки від часу.

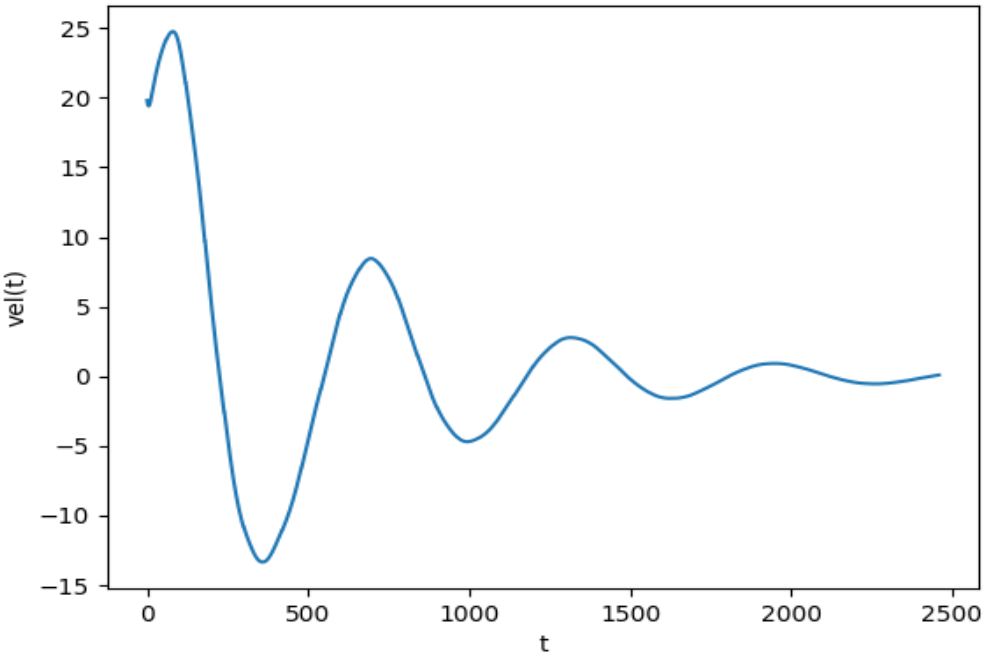


Рис. 2.7: Графік залежності швидкості руху вантажа від часу.

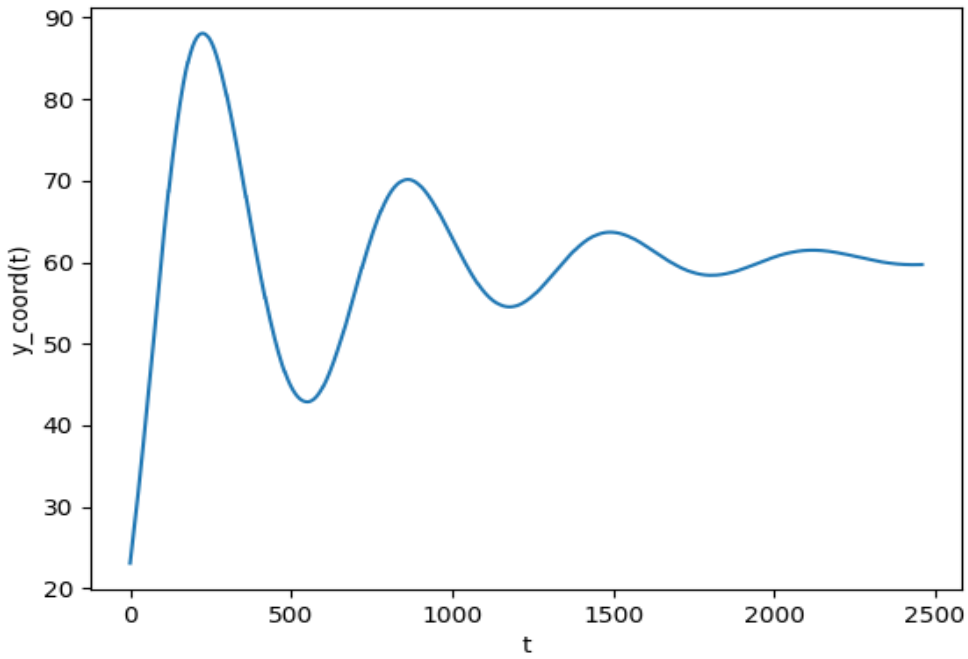


Рис. 2.8: Графік залежності координати вантажа від часу.

Розділ 3. Лістинги програм

У цьому розділі наведено лістинги програм.

Проектування розглянутої системи зручно робити на основі принципів ООП (Об'єктно-орієнтованого програмування). Код програми розбито на класи. Кожен клас у коді описує деяку сутність, яка містить поля або властивості (деякі характеристики об'єкта цього класу) і методи (функції), які певним чином обробляють дані під час роботи програми. Методи призначені для відображення об'єктів у вікні, оновлення положення точок у процесі ітерацій на основі отриманого чисельного розв'язку, тощо.

Лістинг 3.1: Завантаження необхідних бібліотек

```
import pygame as pg
import numpy as np
from scipy.integrate import solve_ivp
from math import sqrt
import matplotlib.pyplot as plt
import numpy as np
```

Лістинг 3.2: Імплементація абстрактного класу точки

```
class Point:
    '''
    Base point class. FixedPoint and MassPoint classes inherits
       methods from this class
    '''


    def __init__(self):
        self.point_color = 'black'
        self.point_data = {'coordinates': [], 'radius': None}
```

```
def draw_point(self):
    pg.draw.circle(self.surface, self.point_color,
                   (self.x, self.y), self.radius)


def set_surface(self, surface):
    self.surface = surface


def set_color(self, color):
    self.point_color = color


def set_radius(self, radius):
    self.radius = radius
    self.point_data['radius'] = self.radius


def set_coords(self, coords):
    self.x, self.y = coords
    self.point_data['coordinates'] = np.array([self.x, self.
      y], dtype = 'd')
```

Код класу, що описує рухому точкову масу m_i. Цей клас наслідується від попереднього класу, але крім цього до нього додаються методи, властиві саме для рухомих точок: методи для оновлення координат точки і її компонент вектора швидкості, метод для встановлення значення маси.

Лістинг 3.3: Клас рухомої точки

```
class MassPoint(Point):
    '''
        Class for weighted moving point of the horizontal and vertical ropes
    '''

    def __init__(self, id, mass, radius, start_coords, start_velocity):
        super().__init__()
        self.id = id
        self.set_coords(start_coords)
        self.vel_x, self.vel_y = start_velocity
        self.mass = mass
        self.set_radius(radius)
        self.point_color = 'black'
        self.point_data = {'id': self.id,
            'coordinates': np.array([self.x, self.y]),
            'velocities': np.array([self.vel_x, self.vel_y]),
            'radius': self.radius,
            'mass': self.mass}
        self.si_point_data = {'coordinates': None, 'velocities': None}

    def update_coords(self, updated_x, updated_y):
        self.si_point_data['coordinates'] = np.array([updated_x, updated_y])
        self.x, self.y = self.meters_to_pixels()
        self.point_data['coordinates'] = np.array([self.x, self.y])


    def update_velocities(self, updated_vel_x, updated_vel_y):
        self.si_point_data['velocities'] = np.array([
```

```
            updated_vel_x, updated_vel_y])
        self.vel_x, self.vel_y = self.meters_to_pixels(mode = '
            vel')
        self.point_data['velocities'] = np.array([self.vel_x,
            self.vel_y])


    def get_data(self): return self.point_data


    def meters_to_pixels(self, mode = 'coord'):
        if mode == 'coord':
            pix_coords = self.si_point_data['coordinates'] *
                self.scale
            pix_coords += self.displace_vector
        elif mode == 'vel':
            pix_coords = self.si_point_data['velocities'] * self
                .scale
        return pix_coords


    def set_mass(self, mass):
        if mass > 0:
            self.mass = mass
            self.point_data['mass'] = self.mass
        else:
            raise ValueError('Mass value of the point must be a
                positive real number!')
```

Код класу фіксованої точки. Цей клас також наслідується від батьківського класу Point.

Лістинг 3.4: Клас фіксованої точки

```
class FixedPoint(Point):
    '''
```

```
        Class for fixed points on the edges!
    '''
    def __init__(self, x_coord, y_coord, radius, point_color = '
        black'):
        self.radius = radius
        self.point_color = point_color
        self.__velocities = [0, 0]
        self.point_data = {'coordinates': [],
                           'velocities': self.__velocities,
                           'radius': self.radius,
                           'color': self.point_color}
        self.set_coords((x_coord, y_coord))
        self.si_point_data = {'coordinates': None, 'velocities':
            self.__velocities}


    def set_convert_data(self, scale, displace_vector):
        self.scale, self.displace_vector = scale,
            displace_vector
        if self.si_point_data['coordinates'] is None:
            self.si_point_data['coordinates'] = (self.point_data
                ['coordinates'] - self.displace_vector) / self.
                scale
```

Нижче наведено код класу, що описує сутність фрагменту мотузки. Саме з конкретних представників (екземплярів або об'єктів) цього класу складається кожна мотузка у нашій механічній системі. Тобто кожна з мотузок – це ланцюжок послідовно з'єднаних екземплярів класу RopeFragment, таких, що для одного з них точкова маса m_i є лівою крайовою точкою, а для другого – правою. Цей клас містить методи для малювання фрагменту за допомогою лінії, методи для отримання кортежів координат та швидкостей точок, метод для знаходження координат вектора напрямленого між точками, а також для знаходження приросту швидкостей. Також клас містить метод для розрахунку довжини фрагмента мотузки (тобто норму вектора, початком і кінцем яких є точкові маси) і метод для розрахунку сили, що діє в цьому фрагменті мотузки на

точку.

Лістинг 3.5: Клас фрагменту мотузки

```
class RopeFragment:
    '''
    Class for rope fragment between two neighboring points!
    '''
    def __init__(self, start_point, end_point, rest_len,
        stiffness, viscosity):
        self.rest_len = rest_len
        self.stiffness = stiffness
        self.viscosity = viscosity
        self.start_point, self.end_point = start_point,
            end_point
        self.surface = None
        self.color = 'black'
        self.width = 1
        self.max_len = rest_len * 5


    def draw_fragment(self):
        '''
        Method for drawing rope fragment with using Pygame
        '''
        left_point_coords, right_point_coords = self.
            get_points_coords()
        pg.draw.line(self.surface, self.color, left_point_coords
            , right_point_coords, self.width)


    def get_points_coords(self, mode = 'si'):
        if mode == 'si':
            left_coords = self.start_point.si_point_data['
                coordinates']
            right_coords = self.end_point.si_point_data['
```

```
            coordinates']
        elif mode == 'pix':
            left_coords = self.start_point.point_data['
                coordinates']
            right_coords = self.end_point.point_data['
                coordinates']
        return (left_coords, right_coords)


    def get_points_vels(self):
        left_vels = self.start_point.si_point_data['velocities']
        right_vels = self.end_point.si_point_data['velocities']
        return (left_vels, right_vels)


    def vector_coords(self, opposite_vector = False):
        left_coords, right_coords = self.get_points_coords()
        if opposite_vector:
            vector_coords = [left_coords[0] - right_coords[0],
                left_coords[1] - right_coords[1]]
        else:
            vector_coords = [right_coords[0] - left_coords[0],
                right_coords[1] - left_coords[1]]
        return np.array(vector_coords)


    def vel_vector_coords(self, opposite_vector = False):
        left_vels, right_vels = self.get_points_vels()
        if opposite_vector:
            vector_vels = [left_vels[0] - right_vels[0],
                left_vels[1] - right_vels[1]]
        else:
            vector_vels = [right_vels[0] - left_vels[0],
                right_vels[1] - left_vels[1]]
```

```
        return np.array(vector_vels)


    def rope_fragment_len(self):
        vector_coords = self.vector_coords()
        dist = np.linalg.norm(vector_coords)
        if dist >= self.max_len:
            raise ValueError("Rope lenght can't be greater than maximum length!")
        return dist



    def force(self, main_point = 'left'):
        if main_point == 'left':
            vector, vel_vector = self.vector_coords(), self.vel_vector_coords()
        elif main_point == 'right':
            vector, vel_vector = self.vector_coords(opposite_vector = True), self.vel_vector_coords(opposite_vector = True)
        else:
            raise ValueError("Incorrect value of main_point parameter! Must be 'left' or 'right'.")


        vector_norm = self.rope_fragment_len()
        if vector_norm < self.rest_len:
            return np.array([0.0, 0.0])

        vector /= vector_norm
        elastic = self.stiffness * (vector_norm - self.rest_len)
        viscous = self.viscosity * np.dot(vector, vel_vector)
        self.force_vec = (elastic + viscous) * vector / self.
```

```
            rest_len
        self.force_val = np.linalg.norm(self.force_vec)
        return self.force_vec
```

Лістинг 3.6: Основний клас мотузкової системи

```
class RopeModel:
    '''
    General rope model (contains method for horizontal rope and vertical rope)
    '''
    def __init__(self, surface, hor_rope_data):
        self.surface = surface
        self.fixed_point_data = hor_rope_data['fixed_point_data']
        self.hor_moving_point_data = hor_rope_data['moving_point_data']
        self.hor_rope_mass = hor_rope_data['rope_mass']
        self.hor_rope_stiffness = hor_rope_data['rope_stiffness']
        self.hor_rope_viscosity = hor_rope_data['rope_viscosity']
        self.hor_rope_tension = hor_rope_data['tension_force']
        self.fall_height = 20
        self.set_convert_coeff()
        self.set_hor_rest_len()
        self.objects = {'fixed_points': [],
                        'rope_fragments':[],
                        'moving_points':[],
                        'surface': self.surface}



    def create_fixed_points(self):
        for i in range(len(self.fixed_point_data['x_coords'])):
```

```
            fixed_point = FixedPoint(self.fixed_point_data['
                x_coords'][i],
                                     self.fixed_point_data['y_coords'
                                         ][i],
                                     self.fixed_point_data['radius'])
            disp_vec = fixed_point.point_data['coordinates'] if
                i == 0 else disp_vec
            fixed_point.set_convert_data(scale = self.
                pix_per_metr, displace_vector = disp_vec)
            fixed_point.set_surface(self.surface)
            self.objects['fixed_points'].append(fixed_point)


    def create_free_end_rope(self, attach_point_id,
        points_velocities = None):
        self.attach_point_id = attach_point_id
        top_point = self.objects['moving_points'][self.
            attach_point_id]
        points_num = self.ver_rope_point_num
        x_coord = top_point.point_data['coordinates'][0]
        fragment_rest_len = self.get_rope_fragment_rest_len(
            rope_type = 'ver')
        start_coord = top_point.point_data['coordinates'][1]
        end_coord = start_coord + self.ver_rope_rest_len
        vertical_coords = np.linspace(start_coord, end_coord,
            points_num + 1)[1:]
        vertical_coords = [(x_coord, y_coord) for y_coord in
            vertical_coords]
        stiffness = self.get_rope_fragment_property(self.
            ver_rope_stiffness, points_num)
        viscosity = self.get_rope_fragment_property(self.
            ver_rope_viscosity, points_num)
        mass = self.get_moving_point_mass(rope_type = 'ver')
        radius = self.ver_rope_point_radius
```

```
        if points_velocities is None:
            y_vels = self.get_linear_vels_distribution(
                points_num + 1, 0, sqrt(2 * 9.81 * self.
                fall_height))[1:]
            points_velocities = [(0, y_vel * self.pix_per_metr)
                for y_vel in y_vels]

        for i in range(points_num):
            if i == points_num - 1:
                mass = self.weighted_point_mass
                radius = self.weighted_point_radius

            if i != 0:
                top_point = self.objects['moving_points'][-1]

            self.create_moving_point(id =
                        len(self.objects['moving_points']) + 1,
                        mass = mass, radius = radius,
                        coords = vertical_coords[i],
                        velocities = points_velocities[i])
            bottom_point = self.objects['moving_points'][-1]
            self.create_rope_fragment(top_point, bottom_point,
                stiffness, viscosity, fragment_rest_len)


        self.general_points_num = len(self.objects['
            moving_points'])


    def create_rope_fragment(self, left_endpoint, right_endpoint
        , stiffness = None, viscosity = None, rest_len = None):
        if rest_len is None:
            rest_len = self.get_rope_fragment_rest_len()
```

```
        if stiffness is None:
            stiffness = self.get_rope_fragment_property(
                        self.hor_rope_stiffness,
                        self.hor_moving_point_data['
                            moving_points_num'])

        if viscosity is None:
            viscosity = self.get_rope_fragment_property(
                self.hor_rope_viscosity,
                self.hor_moving_point_data['moving_points_num'])

        rope_fragment = RopeFragment(left_endpoint,
                                     right_endpoint,
                                     rest_len,
                                     stiffness,
                                     viscosity)
        rope_fragment.surface = self.surface
        self.objects['rope_fragments'].append(rope_fragment)


    def get_linear_vels_distribution(self, point_num, start_vel,
        end_vel):
        vels = np.linspace(start_vel, end_vel, point_num)
        return vels


    def create_moving_point(self, id, mass, radius, coords,
        velocities):
        moving_point = MassPoint(id,
                                 mass,
                                 radius,
                                 (coords[0], coords[1]),
                                 (velocities[0], velocities[1]))
        moving_point.set_surface(self.surface)
```

```
        self.objects['moving_points'].append(moving_point)


    def create_model(self):
        self.create_fixed_points()
        hor_coords = np.linspace(
                self.objects['fixed_points'][0].x,
                self.objects['fixed_points'][1].x,
            self.hor_moving_point_data['moving_points_num'] + 2)
                [1 :-1 :]

        vertical_coord = self.fixed_point_data['y_coords'][0]
        self.set_moving_point_mass(self.hor_rope_mass,
            self.hor_moving_point_data['moving_points_num'])

        for i in range(self.hor_moving_point_data['
            moving_points_num']):
            self.create_moving_point(i,
                self.hor_moving_point_mass,
                self.hor_moving_point_data['moving_points_radius
                    '],
                (hor_coords[i], vertical_coord),
                self.hor_moving_point_data['
                    moving_points_velocities'])


        left_point = self.objects['fixed_points'][0]
        right_point = self.objects['moving_points'][0]
        self.create_rope_fragment(left_point, right_point)

        for j in range(self.hor_moving_point_data['
            moving_points_num'] - 1):
            left_point = self.objects['moving_points'][j]
            right_point = self.objects['moving_points'][j + 1]
```

```
            self.create_rope_fragment(left_point, right_point)

        left_point = self.objects['moving_points'][-1]
        right_point = self.objects['fixed_points'][1]
        self.create_rope_fragment(left_point, right_point)



def draw_model(self):
    for fixed_point in self.objects['fixed_points']:
        fixed_point.draw_point()


    for moving_point in self.objects['moving_points']:
        moving_point.draw_point()


    for rope_fragment in self.objects['rope_fragments']:
        rope_fragment.draw_fragment()


def set_moving_point_mass(self, rope_mass: float,
    moving_points_num: int, rope_type = 'hor'):
    if rope_type == 'hor':
        self.hor_moving_point_mass = rope_mass / (
            moving_points_num)
    elif rope_type == 'ver':
        self.ver_moving_point_mass = rope_mass / (
            moving_points_num - 1)


def get_moving_point_mass(self, rope_type = 'hor'):
    if rope_type == 'hor':
        return self.hor_moving_point_mass
```

```
        elif rope_type == 'ver':
            return self.ver_moving_point_mass


    def add_mass(self, mass: float, point_indx: int):
        masspoint = self.objects['moving_points'][point_indx]
        masspoint.mass += mass if mass >= 0 else 0


    def get_fixed_points_distanse(self):
        x_coords = self.fixed_point_data['x_coords']
        dist = x_coords[1] - x_coords[0]
        return dist


    def set_hor_rest_len(self):
        stretch_rope_len = self.get_fixed_points_distanse() /
            self.pix_per_metr
        hor_rest_len = stretch_rope_len / (self.hor_rope_tension
            / self.hor_rope_stiffness + 1)
        if hor_rest_len > 0:
            self.hor_rest_len = hor_rest_len
        else:
            print(f"""Horizontal rest len: {hor_rest_len},\n
                    Stretched rope len: {stretch_rope_len},\n
                    Horizontal rope stiffness: {self.
                        hor_rope_stiffness}""")
            raise ValueError("Rest len of the horizontal rope
                must be a positive float!")


    def set_convert_coeff(self, real_meter_dist = 100):
        dist = self.get_fixed_points_distanse()
        self.pix_per_metr = dist / real_meter_dist
```

```
def get_rope_fragment_rest_len(self, rope_type = 'hor'):
    if rope_type == 'hor':
        rope_fragments_num = self.hor_moving_point_data['
            moving_points_num'] + 1
        hor_fragment_rest_len = self.hor_rest_len /
            rope_fragments_num
        return hor_fragment_rest_len

    elif rope_type == 'ver':
        rope_fragments_num = self.ver_rope_point_num
        ver_fragment_rest_len = self.ver_rope_rest_len /
            rope_fragments_num
        return ver_fragment_rest_len


def get_rope_fragment_property(self, property_value: float,
   points_num: int) -> float:
    '''
    Function returns values of stiffness or viscosity of the
        rope fragment by
    by the corresponding values of stiffness or viscosity of
        the whole rope
    (it does't matter horizontal or vertical).
    property_value is a value of stiffness or viscosity of
       the whole rope.
    '''
    return property_value * points_num


def set_ver_rope_data(self, ver_rope_data):
    self.ver_rope_rest_len = ver_rope_data['rest_len'] /
       self.pix_per_metr
```

```
        self.ver_rope_mass = ver_rope_data['rope_mass']
        self.ver_rope_stiffness = ver_rope_data['rope_stiffness'
            ]
        self.ver_rope_viscosity = ver_rope_data['rope_viscosity'
            ]
        self.ver_rope_point_num = ver_rope_data['point_num']
        self.ver_rope_point_radius = ver_rope_data['
            ver_rope_point_radius']
        self.weighted_point_mass = ver_rope_data['
            weighted_point_mass']
        self.weighted_point_radius = ver_rope_data['
            weighted_point_radius']
        self.set_moving_point_mass(self.ver_rope_mass, self.
            ver_rope_point_num, rope_type = 'ver')


    def split_data(self, data_list):
        '''
        Splitting the data_list containing info about
            coordinates and velocities of each point in one
        general 1d-list into the following format:
        [data_1, data_2, ..., data_point_num], where data_i is a
            data list [x, dx, y, dy]
        for i-th point of the system.
        '''
        splitted_data = []
        for i in range(0, len(data_list), 4):
            splitted_data.append(data_list[i : i + 4])

        return splitted_data


    def system(self, t, data_list):
        '''
```

```
        A method for solving the system of differential equations.
        This function is passed as an argument into solve_ivp from Scipy.
        '''
        if self.first_num_step:
            self.first_num_step = False
            return self.init_right_parts

        self.update_system_state(data_list)
        splitted_data = self.split_data(data_list)
        updated_right_parts = []
        for point_num in range(self.general_points_num):
            '''
            This loop generates 4 diff equations for each point of the system:
            2 equations for x and x' (x coordinate and x velocity (the first derivative of x by time))
            2 equations for y and y'
            '''
            if point_num <= self.hor_moving_point_data['moving_points_num'] - 1:
                #Horizontal rope points
                left_rope = self.objects['rope_fragments'][point_num]
                right_rope = self.objects['rope_fragments'][point_num + 1]

                if point_num == self.attach_point_id:
                    #Attachment point case
                    indx = self.hor_moving_point_data['moving_points_num'] + 1
                    vert_rope = self.objects['rope_fragments'][indx]
```

```
elif self.hor_moving_point_data['moving_points_num'] - 1 < point_num:
    #Vertical rope points case
    top_rope = self.objects['rope_fragments'][point_num + 1]
    if point_num < self.general_points_num - 1:
        #Vertical rope points (case of unweighted point)
        down_rope = self.objects['rope_fragments'][point_num + 2]

'''
Definitions:
w  = x -- x coord of the point in the previous time step
dw = w' = z = x' -- velocity of the x coord
dz = z' = x'' -- acceleration of the x coord
u  = y -- y coord of the point in the previous time step
du = u' = v = y' -- velocity of the y coord
dv = v' = y'' -- acceleration of the y coord
'''
w, z, u, v = splitted_data[point_num]

if point_num < self.general_points_num - 1:
    if point_num == self.attach_point_id: # attachment point case
        left_rope_force = left_rope.force(main_point = 'right')
        right_rope_force = right_rope.force()
        if self.general_points_num == self.hor_moving_point_data['moving_points_num']
```

```
                ]:
                    vert_rope_force = np.array([0.0, 0.0])
                else:
                    vert_rope_force = vert_rope.force()
                dz = (left_rope_force[0] + right_rope_force
                    [0] + vert_rope_force[0]) / point_mass
                dv = (left_rope_force[1] + right_rope_force
                    [1] + vert_rope_force[1]) / point_mass


            elif point_num > self.hor_moving_point_data['
                moving_points_num'] - 1:
                top_rope_force = top_rope.force(main_point =
                    'right')
                down_rope_force = down_rope.force()
                dz = (top_rope_force[0] + down_rope_force
                    [0]) / point_mass
                dv = (top_rope_force[1] + down_rope_force
                    [1]) / point_mass + 9.81


            else:
                left_rope_force = left_rope.force(main_point
                    = 'right')
                right_rope_force = right_rope.force()
                dz = (left_rope_force[0] + right_rope_force
                    [0]) / point_mass
                dv = (left_rope_force[1] + right_rope_force
                    [1]) / point_mass


        else: #weighted point case

            top_rope_force = top_rope.force(main_point = '
```

```
                right')
            dz = top_rope_force[0] / point_mass
            dv = top_rope_force[1] / point_mass + 9.81
        #Diff eqs for mass point with number point_num

        dw = z
        du = v
        updated_data_list += [dw, dz, du, dv]

    return updated_data_list

def get_ivp_data(self):
    '''
    Inverse function (in some sence) to split_data function.
    This function converts coordinates and velocities of each separate point into the general
    list, which contains all coords and vels.
    '''
    data = []
    for point in self.objects['moving_points']:
        coords = point.si_point_data['coordinates']
        vels = point.si_point_data['velocities']
        data += [coords[0], vels[0], coords[1], vels[1]]
    return data


def initial_right_parts(self):
    self.init_right_parts = []
    self.first_num_step = True
    for i, point in enumerate(self.objects['moving_points']):
        mass, vels = point.point_data['mass'], point.si_point_data['velocities']
        if i < self.hor_moving_point_data['moving_points_num
```

```
            ']:
                right_part_point_data = [vels[0], self.
                    hor_rope_tension / mass, vels[1], 0]
            else:
                right_part_point_data = [vels[0], 0, vels[1],
                    9.81]
            self.init_right_parts += right_part_point_data


    def update_system_state(self, new_state_list):
        state = self.split_data(new_state_list)
        for i, point in enumerate(
                self.objects['moving_points']):
            new_point_state = state[i]
            point.update_coords(new_point_state[0],
                new_point_state[2])
            point.update_velocities(new_point_state[1],
                new_point_state[3])


    def solve_ivp(self, initial_cond, duration = 1, start_time =
        0, fps = 60, method = 'Radau'): #RK45
        '''
        A method for solving a system of differential equations
            using a numerical method at each step of the game
            cycle.
        '''
        fps = fps
        duration = duration # 1 sec
        t_span = (start_time, start_time + duration) #(0, 1)
        t_eval = np.linspace(t_span[0], t_span[1], fps) #(0,
            0.1, 0.2, 1)
        solution = solve_ivp(self.system, method = method,
            t_span = t_span, y0 = initial_cond, t_eval = t_eval)
```

```
        return solution


    def set_weighted_point_data(self, radius, mass):
        weighted_point = self.objects['moving_points'][-1]
        weighted_point.set_mass(mass)
        weighted_point.set_radius(radius)
```

Вище наведено код основного класу. Цей клас містить методи для побудови фіксованих точок, обох мотузок, метод для розв'язання системи диференціальних рівнянь, розрахунок сил у правих частинах цієї системи та оновлення стану механічної системи (оновлення положень точок та їх швидкостей). Окрім цього є багато інших службових методів, таких як функція для малювання системи на екрані, розбиття вектора розв'язку системи рівнянь та отримання даних задачі Коші (початкових положень та швидкостей).

Висновки

У представленій роботі було досліджено двовимірну дискретизовану мотузкову систему. Кожен з фрагментів обох мотузок розглядався як паралельне з'єднання пружнього елемента (пружини) та в'язкого елемента (демпфера). Механічна система складалась з двох мотузок, одна з яких горизонтально розміщена та мала початковий натяг (базова мотузка), а інша – мотузка-поводок, одним кінцем прикріплена посередині до базової мотузки, а на іншому вільному кінці закріплено вантаж. Були виведені диференціальні рівняння, що описують рух системи під навантаженням. Ми сформулювали задачу Коші для вільно падаючого об'єкта на кінці мотузки-поводка. Систему було розглянуто з точки зору диференціальних рівнянь з частинними похідними. Наведені система з двох хвильових рівнянь, постановки початково-крайових задач. У роботі за допомогою мови програмування Python та бібліотек до неї було побудовано комп'ютерну модель системи і проведено велику кількість експериментів. Ми намагалися відтворити поведінку реальної мотузкової системи задаючи відповідні параметри жорсткості, в'язкості, маси мотузок та кількість точок дискретизації. При дослідженні подібних систем важливим елементом є оцінка максимальних навантажень, що виникають у системі особливо у точках кріплення мотузок. З цією метою наведені графіки модулів сил, швидкості руху вантажа та зміна його вертикальної координати з плином часу. Було проведено аналіз і порівняння з нашою моделлю сучасних наукових робіт по темі. Наведено великий список використаних джерел. В останньому розділі наведено код програми. Вивчення властивостей подібних до розглянутої мотузкових систем, а також побудова високоточних аналітичних моделей є важливим кроком для розробки безпечних та надійних систем страхування.

Додаток А. Знімки руху системи.

Нижче наведено знімки для двох конфігурацій системи з більшою кількістю точок.

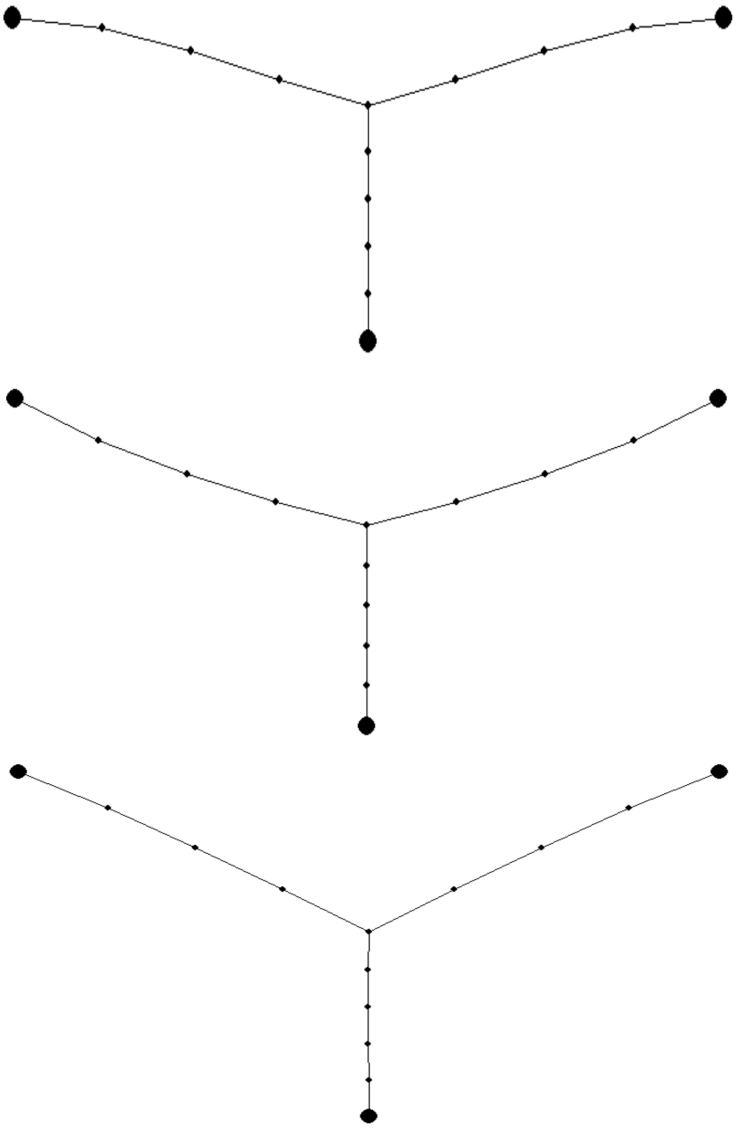


Рис. 3.1: Конфігурація: 7 точок бази і 5 точок мотузки-поводка (початковий натяг 800 Ньютонів).

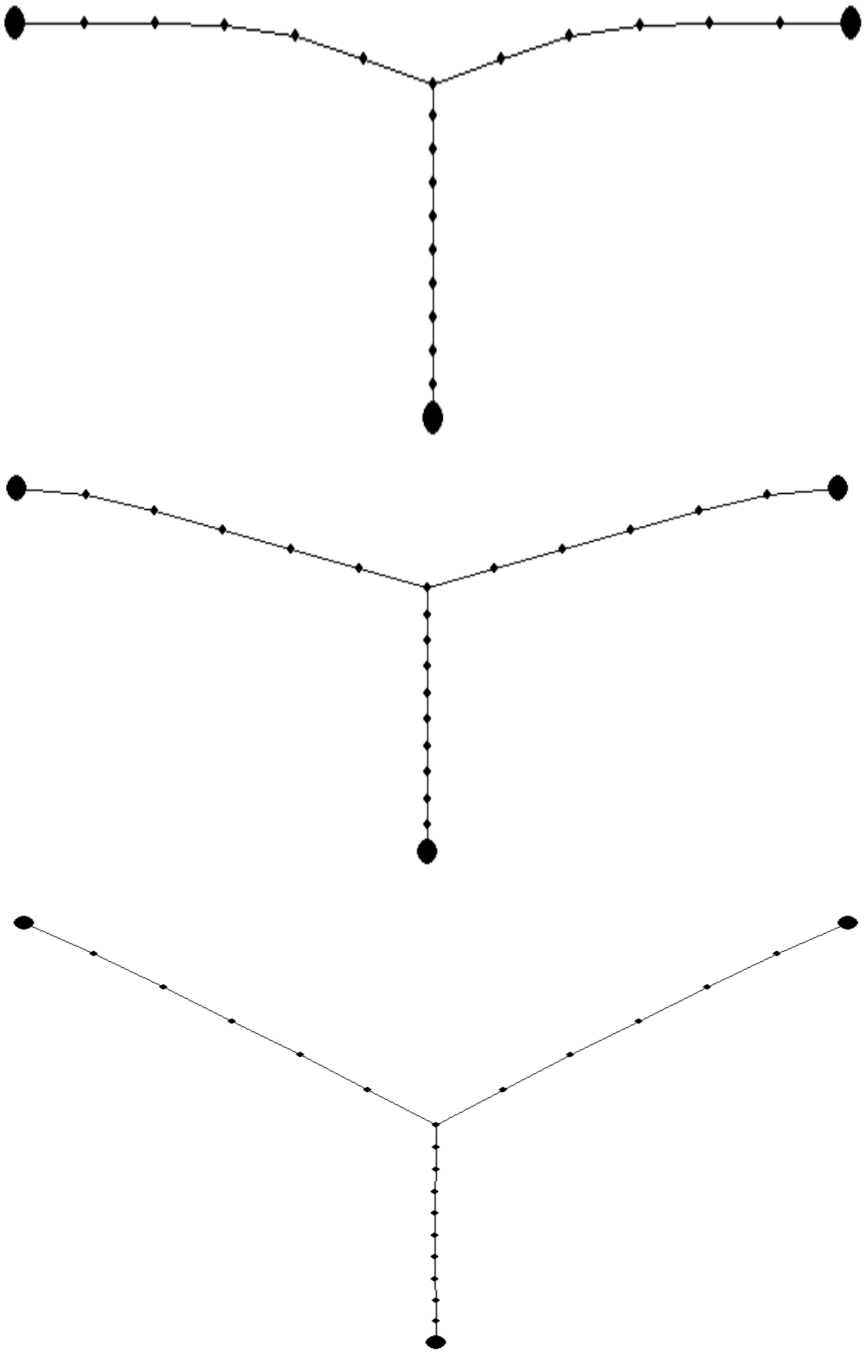


Рис. 3.2: Конфігурація: 11 точок бази і 10 точок мотузки-поводка.

Додаток Б. Графіки.

На зображених нижче графіках по осі часу відкладено інтервал в 30 секунд.

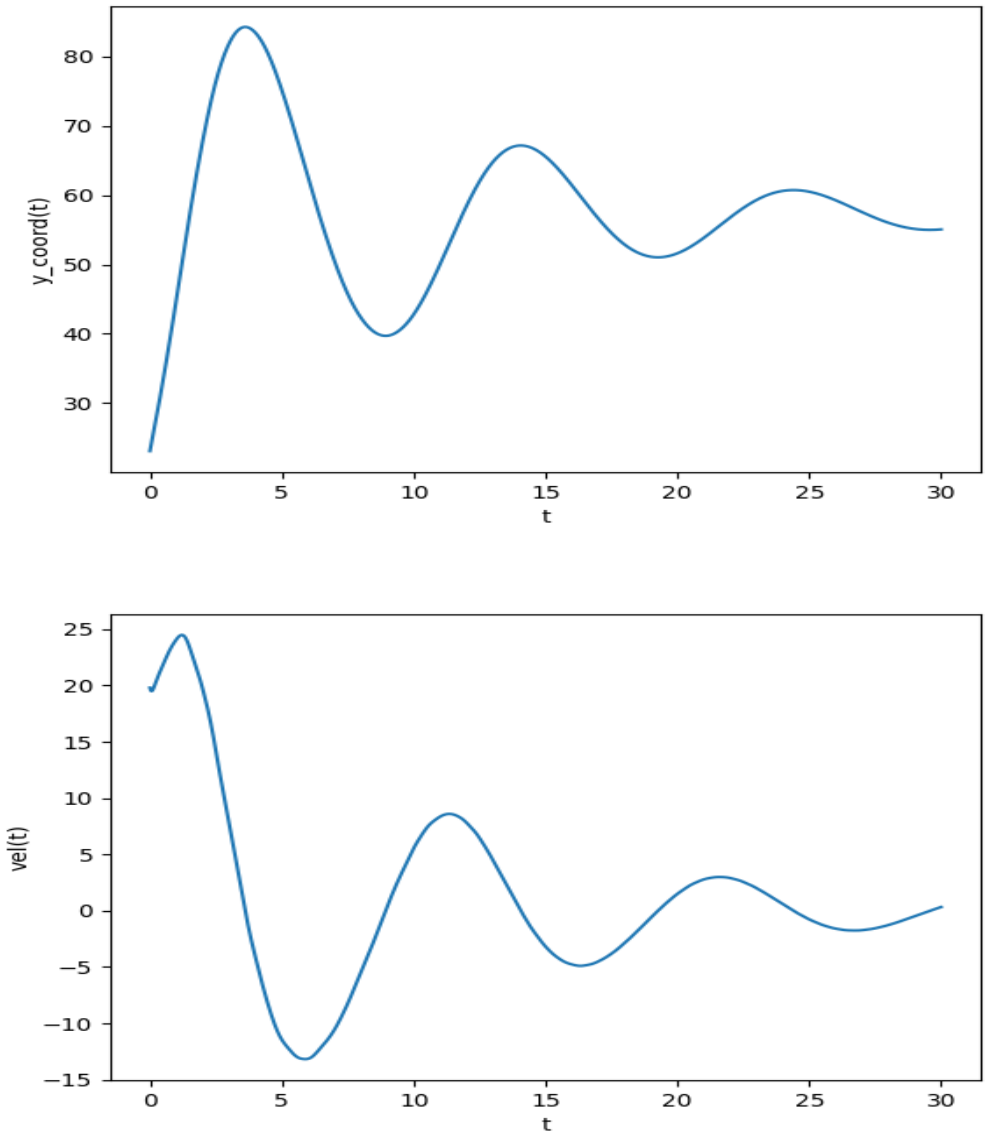


Рис. 3.3: Графіки координати та швидкості для конфігурації 3.1.

На малюнках нижче задача розглядалася на інтервалі в 10 секунд.

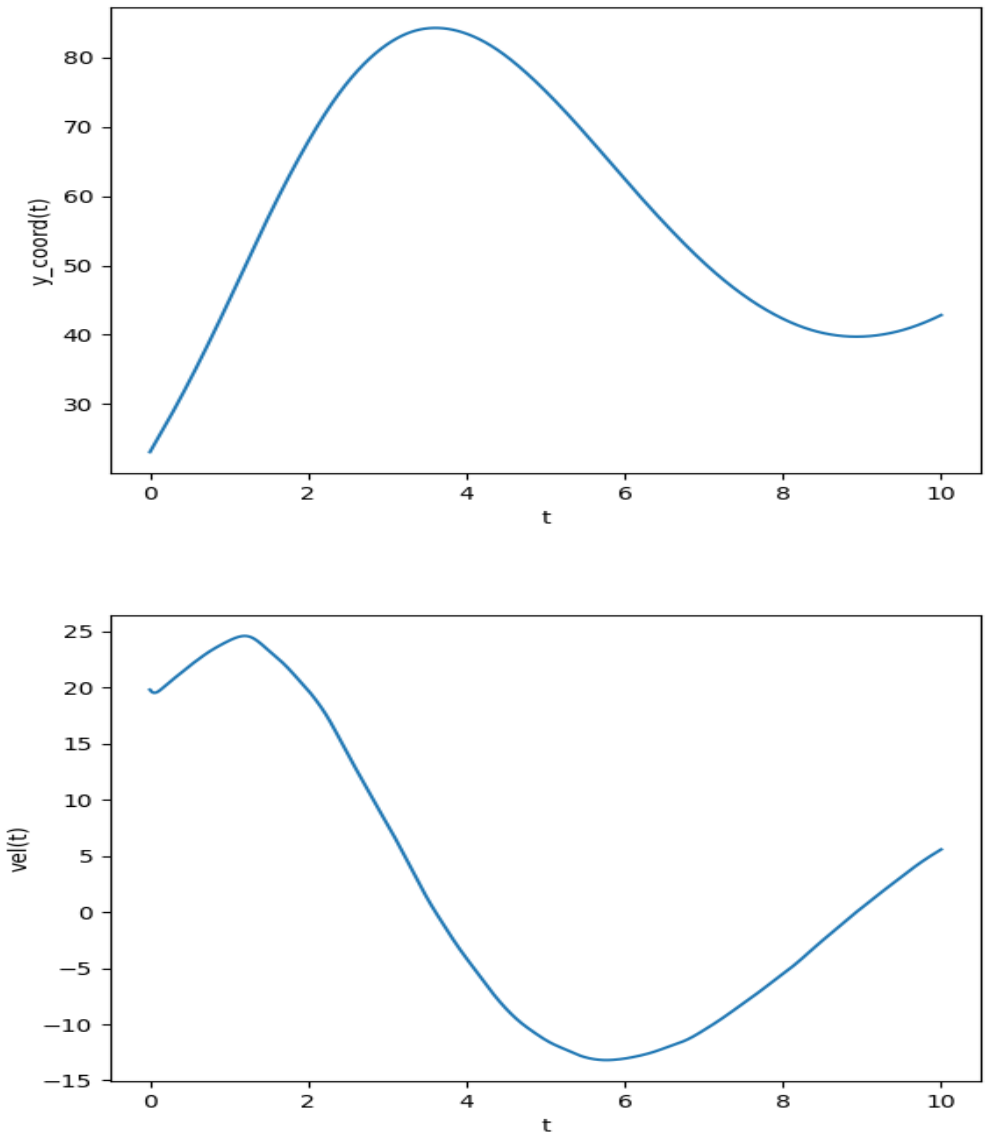


Рис. 3.4: Графіки координати та швидкості для конфігурації 3.2.

Додаток В. Код функцій для моделювання.

Лістинг 3.7: Імпорт бібліотек та класу моделі

```
import pygame as pg
from rope_model import RopeModel
import matplotlib.pyplot as plt
import numpy as np
```

Лістинг 3.8: Функція для малювання графіків

```
def draw_force_graph(force_vals, y_label = 'F(t)', x_label = 't'
    , time_steps = None):
    plt.ylabel(y_label)
    plt.xlabel(x_label)
    if time_steps is None:
        plt.plot(force_vals)
    else:
        plt.plot(time_steps, force_vals)
    plt.show()
```

Лістинг 3.9: Функція для ітеративної задачі Коші

```
def pg_simulation(initial_cond, running, clock, FPS):
    force_vals = []
    speed_vals = []
    coord_vals = []
    rope_indx = rope.hor_moving_point_data['moving_points_num']
    while running:
```

```
        sol = rope.solve_ivp(initial_cond, fps = FPS, method = '
            Radau')
        initial_cond = sol.y[:, -1]

        for i in range(FPS):
            updated_coords = sol.y[:, i]
            rope.update_system_state(updated_coords)
            surface.fill(white)
            rope.draw_model()
            pg.display.flip()
            clock.tick(FPS)
            force_vals.append(rope.objects['rope_fragments'][
                rope_indx].force_val)
            coord_vals.append(rope.objects['moving_points'][-1].
                si_point_data['coordinates'][1])
            speed_vals.append(rope.objects['moving_points'][-1].
                si_point_data['velocities'][1])
    pg.quit()

    draw_force_graph(force_vals)
    draw_force_graph(speed_vals, y_label = 'vel(t)')
    draw_force_graph(coord_vals, y_label = 'y_coord(t)')
```

Лістинг 3.10: Допоміжна функція для розбиття часового інтервалу

```
def find_time_steps(sec_num = 180, time_steps_per_second = 100):
    return {'general_time_steps':
                    sec_num * time_steps_per_second,
                    'delay': int(1000 / time_steps_per_second)}
```

Лістинг 3.11: Функція для однієї задачі Коші на всьому інтервалі

```
def time_step_simulation(initial_cond, duration, time_steps_num,
    delay):
    sol = rope.solve_ivp(initial_cond, duration = duration, fps
        = time_steps_num)
```

```
rope_indx = rope.hor_moving_point_data['moving_points_num']
force_vals, speed_vals, coord_vals = [], [], []
for t_step in range(time_steps_num):
    updated_state = sol.y[:, t_step]
    rope.update_system_state(updated_state)
    surface.fill(white)
    rope.draw_model()
    pg.display.flip()
    force_vals.append(rope.objects['rope_fragments'][
        rope_indx].force_val)
    coord_vals.append(rope.objects['moving_points'][-1].
        si_point_data['coordinates'][1])
    speed_vals.append(rope.objects['moving_points'][-1].
        si_point_data['velocities'][1])
    pg.time.delay(delay)
    for event in pg.event.get():
        if event.type == pg.QUIT:
            pg.quit()

draw_force_graph(force_vals, time_steps = sol.t)
draw_force_graph(speed_vals, time_steps = sol.t, y_label = '
    vel(t)')
draw_force_graph(coord_vals, time_steps = sol.t, y_label = '
    y_coord(t)')
```

Список використаних джерел

[1] Leuthäusser U. The physics of a climbing rope under a heavy dynamic load. *Proceedings of the Institution of Mechanical Engineers, Part P: Journal of Sports Engineering and Technology.*, 231(2):125–135, 2017.

[2] Leuthäusser U. Physics of a climbing ropes: impact forces, fall factors and rope drag, part 2, version 3 (12.7.2016).

[3] R.M. Christensen. *Theory of Viscoelasticity.* Civil, Mechanical and Other Engineering Series. Dover Publications, 2003.

[4] M. Reiner. *An Elementary Introduction to Theoretical Rheology.* H.K. Lewis, 1949.

[5] Карвацький А. Я. *Механіка суцільних середовищ [Електронний ресурс]: навч. посіб.* КПІ ім. Ігоря Сікорського, 2016.

[6] Жук Я.О. Будак В.Д. *Механіка суцільних середовищ: навчальний посібник.* Миколаїв: Іліон, 2011.

[7] D. Morin. *Introduction to Classical Mechanics: With Problems and Solutions.* Cambridge University Press, 2008.

[8] Robert A. Becker. *Introduction to Theoretical Mechanics.* McGraw-Hill, 1954.

[9] Sunayna Singh and Matteo Mastrogiuseppe. Multibody modelling of tether and capture system for dynamic simulations of in-air capturing. *Acta Astronautica*, 218:59–69, 2024.

[10] Waldemar Tomaszewski and Piotr Pieranski. Dynamics of ropes and chains: I. the fall of the folded chain. *New Journal of Physics*, 7(1):45, feb 2005.

[11] Pawel Fritzkowski and Henryk Kaminski. Dynamics of a rope as a rigid multibody system. *Journal of Mechanics of Materials and Structures*, 3:1059–1075, 08 2008.

[12] N.H. Asmar. *Partial Differential Equations with Fourier Series and Boundary Value Problems: Third Edition*. Pearson Prentice Hall, 2005.

[13] Python Software Foundation. Python programming language. https://www.python.org.

[14] Numpy library. https://numpy.org/.

[15] Matplotlib library. https://matplotlib.org/.

[16] Pygame library. https://www.pygame.org/news.

[17] Scipy library. https://scipy.org/ https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html.

[18] Chapra S. Canale R. *Numerical Methods for Engineers*. McGraw-Hill Education, 2014.

[19] R.L. Burden and J.D. Faires. *Numerical Analysis*. Cengage Learning, 2010.

[20] J.R. Dormand and P.J. Prince. A family of embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980.