Міністерство освіти та науки України
Національний університет водного господарства та природокористування
Кафедра екології, технології захисту навколишнього середовища та лісового господарства

03-06-170М

МЕТОДИЧНІ ВКАЗІВКИ

до виконання практичних робіт з навчальної дисципліни «Проблемно-орієнтовані методи аналізу та оптимізація процесів» для здобувачів вищої освіти другого (магістерського) рівня за освітньою програмою «Біотехнології, біоробототехніка та біоенергетика» спеціальності 162 «Біотехнології та біоінженерія» денної форми навчання

Рекомендовано
науково-методичною радою з якості
НІІ будівництва та архітектури
Протокол №5 від 11 лютого 2025 р.

Рівне – 2025

Методичні вказівки до виконання практичних робіт з навчальної дисципліни «Проблемно-орієнтовані методи аналізу та оптимізація процесів» для здобувачів вищої освіти другого (магістерського) рівня за освітньою програмою «Біотехнології, біоробототехніка та біоенергетика» спеціальності 162 «Біотехнології та біоінженерія» денної форми навчання [Електронне видання] / Буднік З. М. – Рівне : НУВГП, 2025. – 36 с.

Укладачі: Буднік З. М. – кандидат сільськогосподарських наук, доцент кафедри екології, технології захисту навколишнього середовища та лісового господарства.

Відповідальний за випуск: Мартинов С. Ю., доктор технічних наук, професор, завідувач кафедри водопостачання, водовідведення та бурової справи.

Керівник групи забезпечення освітньо-професійної програми першого (освітньо-професійного) рівня за освітньою програмою «Біотехнології, біоробототехніка табіоенергетика» спеціальності 162 «Біотехнології та біоінженерія» к.т.н., доцент Грицина О. О.

Зміст

ВСТУП

Сучасний розвиток науки і техніки вимагає від фахівців у галузі біотехнологій, біоробототехніки та біоенергетики глибоких знань у сфері аналізу та оптимізації біотехнологічних процесів. Дисципліна «Проблемно-орієнтовані методи аналізу та оптимізація процесів» спрямована на формування у здобувачів вищої освіти другого (магістерського) рівня навичок системного аналізу, моделювання та оптимізації біотехнологічних процесів з використанням сучасних методів і технологій.

Методичні вказівки до виконання лабораторних робіт розроблено для студентів денної форми навчання освітньої програми «Біотехнології, біоробототехніка та біоенергетика» спеціальності 162 «Біотехнології та біоінженерія». Вони містять теоретичні основи, алгоритми виконання лабораторних робіт, рекомендації щодо аналізу отриманих результатів та приклади розв’язання типових завдань.

Метою лабораторних робіт є набуття практичних навичок у застосуванні методів оптимізації та аналізу процесів у біотехнологічних системах. Виконання лабораторних завдань сприятиме розвитку критичного мислення, уміння ухвалювати обґрунтовані рішення та використовувати сучасні інформаційні технології у наукових дослідженнях.

Запропоновані методичні вказівки допоможуть студентам ефективно засвоїти матеріал дисципліни та підготуватися до

подальшої професійної діяльності у сфері біотехнологій, біоробототехніки та біоенергетики.

Лабораторні роботи виконуються згідно з навчальним планом та проводяться у спеціалізованих лабораторіях з використанням сучасного програмного забезпечення та обладнання. Кожна лабораторна робота складається з таких основних етапів:

1. Ознайомлення з теоретичними основами – вивчення методів аналізу та оптимізації процесів, розгляд алгоритмів та підходів.
2. Практичне виконання завдання – використання спеціалізованого програмного забезпечення та інструментів для моделювання й аналізу.
3. Обробка результатів – аналіз отриманих даних, порівняння з теоретичними очікуваннями, побудова висновків.
4. Оформлення звіту – узагальнення проведених досліджень у формі звіту відповідно до встановлених вимог.

Виконання лабораторних робіт здійснюється під керівництвом викладача, який контролює дотримання методичних рекомендацій, допомагає у розв'язанні складних завдань та оцінює результати роботи студентів.

Наступні розділи містять детальні описи кожної лабораторної роботи, включаючи постановку завдань, методичні рекомендації та приклади розв'язання типових задач.

ПРАКТИЧНА РОБОТА 1
МОДЕЛЮВАННЯ ФЕРМЕНТАЦІЙНОГО ПРОЦЕСУ У MATLAB АБО PYTHON

Мета: Навчитися створювати математичні моделі біотехнологічних процесів (на прикладі ферментації) з використанням програмних засобів MATLAB або Python. Основна увага приділяється розумінню ключових параметрів ферментаційного процесу та їхньому аналізу.

Теоретичні основи:

Ферментація — це процес перетворення органічних сполук під дією мікроорганізмів або ферментів. Він широко застосовується в біотехнології для виробництва різних продуктів, таких як етанол, антибіотики, ферменти, біопаливо тощо.

Основні стадії ферментації:

1. *Лаг-фаза* — адаптація мікроорганізмів до середовища.
2. *Лог-фаза* — швидке розмноження мікроорганізмів.
3. *Стаціонарна фаза* — рівновага між ростом і загибеллю мікроорганізмів.
4. *Фаза загибелі* — виснаження поживних речовин і зниження активності мікроорганізмів.

Ключові параметри для моделювання:

- X(t) – концентрація мікроорганізмів (клітин), г/л.
- S(t) – концентрація субстрату (поживної речовини), г/л.
- P(t) – концентрація продукту ферментації, г/л.
- μ – специфічна швидкість росту мікроорганізмів.
- Yx/s – коефіцієнт перетворення субстрату в біомасу.
- Ks – константа насичення (субстрату).

Модель Монода:

$$\mu = \mu_{max} \frac{S}{K_s + S}$$

де μ_{max} — максимальна специфічна швидкість росту.

Завдання:

1. Розробити математичну модель ферментації, враховуючи динаміку зміни концентрації мікроорганізмів, субстрату та продукту у часі на основі рівнянь Монода.
2. Реалізувати модель у MATLAB або Python та візуалізувати результати для аналізу поведінки системи в різні моменти часу.

Хід виконання:

Крок 1: Огляд математичних моделей ферментації

На основі моделі Монода та рівнянь балансу речовин складаються три основні рівняння для концентрацій:

- Для біомаси (мікроорганізмів):

$$\frac{dX}{dt} = \mu X$$

- Для субстрату:

$$\frac{dS}{dt} = -\frac{1}{Y_{x/s}} \mu X$$

- Для продукту:

$$\frac{dP}{dt} = Y^{p}/_{x}\,\mu X$$

Крок 2: Введення початкових умов та параметрів

Необхідно задати початкові значення концентрацій біомаси X_0, субстрату S_0, та продукту P_0. Вибрати значення параметрів, таких як μ_{max}, Ks, $Y_{x/s}$ та $Y_{p/s}$.

Крок 3: Реалізація моделі в MATLAB або Python

MATLAB:

1. Введіть початкові параметри:

```matlab
% Параметри
mu_max = 0.4; % Максимальна швидкість росту (год^-1)
Ks = 0.1; % Константа насичення (г/л)
Yxs = 0.5; % Коефіцієнт перетворення субстрату
Ypx = 0.2; % Коефіцієнт утворення продукту
X0 = 0.1; % Початкова концентрація біомаси (г/л)
S0 = 5; % Початкова концентрація субстрату (г/л)
P0 = 0; % Початкова концентрація продукту (г/л)
tspan = [0 50]; % Часовий інтервал (год)
```

2. Складіть функцію для рівнянь Монода:

```matlab
function dYdt = fermentation_model(t, Y, mu_max, Ks, Yxs, Ypx)
    X = Y(1);
    S = Y(2);
    P = Y(3);
    mu = mu_max * S / (Ks + S);
    dXdt = mu * X;
    dSdt = -1/Yxs * mu * X;
    dPdt = Ypx * mu * X;
    dYdt = [dXdt; dSdt; dPdt];
end
```

3. Використайте ode45 для розв’язку завдань:

```matlab
[T, Y] = ode45(@(t, Y) fermentation_model(t, Y, mu_max, Ks, Yxs, Ypx), tspan, [X0, S0,
```

4. Побудуйте графіки

matlab Копіювати код

```matlab
plot(T, Y(:,1), 'r', T, Y(:,2), 'g', T, Y(:,3), 'b');
legend('Біомаса (X)', 'Субстрат (S)', 'Продукт (P)');
xlabel('Час (год)');
ylabel('Концентрація (г/л)');
title('Моделювання ферментаційного процесу');
```

Python (з використанням SciPy та Matplotlib):

1. Введіть початкові параметри:

python Копіювати код

```python
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Параметри
mu_max = 0.4
Ks = 0.1
Yxs = 0.5
Ypx = 0.2
X0 = 0.1
S0 = 5
P0 = 0
```

2. Складіть функцію рівнянь:

python Копіювати код

```python
def fermentation_model(Y, t, mu_max, Ks, Yxs, Ypx):
    X, S, P = Y
    mu = mu_max * S / (Ks + S)
    dXdt = mu * X
    dSdt = -1/Yxs * mu * X
    dPdt = Ypx * mu * X
    return [dXdt, dSdt, dPdt]
```

3. Використайте ***odient*** для розв’язку рівнянь:

python Копіювати код

```python
t = np.linspace(0, 50, 100)
Y0 = [X0, S0, P0]
result = odeint(fermentation_model, Y0, t, args=(mu_max, Ks, Yxs, Ypx))
```

4. Побудуйте графіки:

```python
plt.plot(t, result[:, 0], 'r', label='Біомаса (X)')
plt.plot(t, result[:, 1], 'g', label='Субстрат (S)')
plt.plot(t, result[:, 2], 'b', label='Продукт (P)')
plt.legend()
plt.xlabel('Час (год)')
plt.ylabel('Концентрація (г/л)')
plt.title('Моделювання ферментаційного процесу')
plt.show()
```

Крок 4: Аналіз результатів

На основі отриманих графіків студенти повинні проаналізувати, як змінюються концентрації біомаси, субстрату та продукту з часом. Особливу увагу слід приділити фазам росту мікроорганізмів і оптимальному часу для максимального виходу продукту.

Висновок: В результаті практичної роботи студенти навчилися будувати математичні моделі ферментаційних процесів і проводити їхню оптимізацію за допомогою програм MATLAB або Python. Отримані навички дозволять їм краще розуміти процеси, які відбуваються під час біотехнологічних виробництв.

ПРАКТИЧНА РОБОТА 2.
ОПТИМІЗАЦІЯ БІОТЕХНОЛОГІЧНОГО ПРОЦЕСУ МЕТОДОМ ГРАДІЄНТНОГО СПУСКУ

Мета: Ознайомитися з методом градієнтного спуску та навчитися використовувати його для оптимізації параметрів біотехнологічного процесу. Основна задача полягає в оптимізації швидкості росту мікроорганізмів шляхом коригування параметрів процесу, таких як концентрація субстрату та температура.

Теоретичні основи:

Градієнтний спуск — це один з основних методів оптимізації, який використовується для мінімізації (або максимізації) функцій. Метод полягає в поступовому зміненні змінних, при якому значення цільової функції зменшується або збільшується в напрямку антиградієнта.

Основні кроки алгоритму градієнтного спуску:

1. Вибір початкових значень змінних (наприклад, параметрів процесу).
2. Обчислення градієнта цільової функції за цими змінними.
3. Оновлення змінних в напрямку, протилежному градієнту (для мінімізації) або в напрямку градієнта (для максимізації).
4. Повторення процесу до досягнення оптимуму (або принаймні значення, близького до нього).

Формула оновлення змінних:

$$\theta_{new} = \theta_{old} - \alpha \cdot \nabla f(\theta)$$

Де: θ – оптимізовані змінні (наприклад, температура або концентрація субстрату), α – крок (швидкість навчання), $f(\theta)$ – градієнт цільової функції по змінним θ.

Цільова функція:

Для цієї практичної роботи цільовою функцією буде швидкість росту мікроорганізмів μ, яка залежить від концентрації субстрату *S* та температури *T*. Функція швидкості росту може мати вигляд:

$$\mu(S,T) = \mu_{max} \cdot \frac{S}{K_S + S} \cdot e^{-\frac{(T-T_{opt})^2}{2\sigma_T^2}}$$

де: μ_{max} — максимальна швидкість росту, S – концентрація субстрату, K_s – константа насичення субстрату, T – температура, T_{opt} – оптимальна температура, σ_T – параметр, який визначає ширину діапазону температур, при якому спостерігається максимальна швидкість росту.

Мета — знайти такі значення S і T, які максимізують швидкість росту мікроорганізмів.

Завдання:

1. Реалізувати цільову функцію швидкості росту μ(S,T).
2. Реалізувати метод градієнтного спуску для оптимізації концентрації субстрату S та температури T.
3. Проаналізувати результати та побудувати графіки зміни швидкості росту залежно від параметрів.

Хід роботи

Крок 1: Визначення цільової функції

Напишемо функцію для розрахунку швидкості росту мікроорганізмів μ(S,T). Ми задамо фіксовані значення для μ_{max}, K_s, T_{opt} і σ_T.

```python
import numpy as np

# Параметри
mu_max = 0.5  # Максимальна швидкість росту
Ks = 0.1  # Константа насичення субстрату
T_opt = 37  # Оптимальна температура
sigma_T = 5  # Діапазон оптимальної температури

# Цільова функція - швидкість росту
def growth_rate(S, T):
    return mu_max * (S / (Ks + S)) * np.exp(-((T - T_opt)**2) / (2 * sigma_T**2))
```

Крок 2: Реалізація градієнтного спуску

1. **Функція градієнтів:** Для використання методу градієнтного спуску нам потрібно обчислювати частинні похідні цільової функції по змінних SSS і TTT. Для цього можна використати чисельне обчислення градієнта.
2. **Алгоритм градієнтного спуску:** Введемо основну функцію, яка буде оновлювати значення SSS та TTT на кожній ітерації на основі градієнтів.

```python
# Функція для чисельного обчислення градієнтів
def compute_gradients(S, T, h=1e-5):
    dS = (growth_rate(S + h, T) - growth_rate(S, T)) / h  # Частинна похідна по S
    dT = (growth_rate(S, T + h) - growth_rate(S, T)) / h  # Частинна похідна по T
    return dS, dT

# Градієнтний спуск
def gradient_descent(S_init, T_init, alpha, tol, max_iter):
    S, T = S_init, T_init
    for i in range(max_iter):
        dS, dT = compute_gradients(S, T)  # Обчислюємо градієнти
        S_new = S + alpha * dS  # Оновлюємо S
        T_new = T + alpha * dT  # Оновлюємо T

        # Перевірка умови збіжності
        if np.abs(S_new - S) < tol and np.abs(T_new - T) < tol:
            break
```

```
        S, T = S_new, T_new  # Оновлюємо змінні

    return S, T
```

Крок 3: Оптимізація швидкості росту

Задамо початкові значення концентрації субстрату S і температури T, а також параметри градієнтного спуску, такі як швидкість навчання α, допустима похибка tol і максимальна кількість ітерацій.

```python
import matplotlib.pyplot as plt

# Створюємо сітку значень S та T для побудови графіка
S_vals = np.linspace(0, 2, 100)
T_vals = np.linspace(20, 50, 100)
S_grid, T_grid = np.meshgrid(S_vals, T_vals)
mu_grid = growth_rate(S_grid, T_grid)

# Побудова графіка
plt.figure(figsize=(8, 6))
plt.contourf(S_grid, T_grid, mu_grid, levels=20, cmap='viridis')
plt.colorbar(label='Швидкість росту μ')
plt.scatter(S_opt, T_opt, color='red', label='Оптимум', marker='x')
plt.xlabel('Концентрація субстрату (S)')
plt.ylabel('Температура (T)')
plt.title('Оптимізація швидкості росту мікроорганізмів')
plt.legend()
plt.show()
```

Результати:

1. **Оптимальна концентрація субстрату та температура:** Ви отримаєте значення SSS та TTT, які максимізують швидкість росту мікроорганізмів, наприклад:

```
Оптимальна концентрація субстрату: 1.234
Оптимальна температура: 37.5
```

1. **Графік залежності швидкості росту від концентрації субстрату і температури:** Ви зможете візуально оцінити, як зміни параметрів впливають на швидкість росту.

Висновок: У цій практичній роботі студенти навчаться використовувати метод градієнтного спуску для оптимізації параметрів біотехнологічного процесу. Вони також дізнаються, як моделювати залежності швидкості росту мікроорганізмів від основних змінних (концентрації субстрату та температури), і зможуть застосовувати ці знання для вирішення реальних задач у біотехнології.

ПРАКТИЧНА РОБОТА 3
АНАЛІЗ І ОПТИМІЗАЦІЯ ПРОЦЕСУ БІОРЕАКТОРА В СЕРЕДОВИЩІ SIMULINK

Мета: ознайомитися з методами моделювання біотехнологічних процесів у біореакторах за допомогою середовища *Simulink* (MATLAB). Навчитися проводити аналіз роботи біореактора та виконувати оптимізацію процесу за різними критеріями, такими як максимізація виходу продукту, мінімізація витрат енергії та часу.

Теоретичні основи

Біореактор — це спеціальний апарат, у якому відбуваються біохімічні реакції за участі живих організмів або біологічних каталізаторів (ферментів). Процес у біореакторі залежить від багатьох параметрів: концентрації субстрату, кількості мікроорганізмів, температури, швидкості аерації, pH тощо.

Основні фази процесу:

1. *Лаг-фаза* — адаптація мікроорганізмів до середовища.
2. *Лог-фаза* — активне зростання клітин і перетворення субстрату на продукт.
3. *Стаціонарна фаза* — досягнення рівноваги між народженням і смертю клітин.
4. *Фаза загибелі* — вичерпання субстрату та зниження швидкості росту.

Математична модель біореактора базується на рівняннях Монода для швидкості росту клітин, рівняннях балансу для субстрату та продукту.

Основні рівняння:

- Для біомаси (мікроорганізмів):

$$\frac{dX}{dt} = \mu X$$

- Для субстрату:

$$\frac{dS}{dt} = -\frac{1}{Y_{x/s}}\mu X$$

- Для продукту:

$$\frac{dP}{dt} = Y^{p}/_{x}\,\mu X$$

- Модель Монода:

$$\mu = \mu_{max}\frac{S}{K_s + S}$$

- $Y_{X/S}$ — коефіцієнт перетворення субстрату в біомасу,
- $Y_{P/X}$ — коефіцієнт утворення продукту з біомаси.

Завдання:

1. Створити модель біореактора в середовищі Simulink, що описує динаміку зміни концентрацій біомаси, субстрату і продукту.
2. Провести симуляцію процесу біореактора для аналізу його динаміки.
3. Виконати оптимізацію процесу за певними критеріями (наприклад, максимізація виходу продукту, мінімізація витрат часу або субстрату).

Хід роботи

Крок 1: Створення математичної моделі біореактора в Simulink

1. **Запуск Simulink:** Відкрийте MATLAB та запустіть Simulink, набравши в командному рядку:
   matlab
   Копіювати код
   simulink
2. **Створення нової моделі:** Створіть нову модель і підготуйте робоче поле для побудови блочної схеми.
3. **Моделювання рівняння для концентрації біомаси (X):** Додайте блок **Integrator** (інтегратор) для моделювання рівняння

$$\frac{dX}{dt} = \mu X$$

   - Додайте блок **Gain** для множення μ на X.
   - Під'єднайте до цього блоку рівняння Монода для обчислення μ через блоки **Divide** (ділення) та **Sum** (сума).

- Введіть константи для μ та Ks через блоки **Constant**.

4. **Моделювання рівнянь для субстрату (S) та продукту (P):**
   Додайте ще два блоки **Integrator** для моделювання рівнянь зміни субстрату та продукту. Для кожного рівняння додайте відповідні блоки **Gain** для врахування коефіцієнтів $Y_{X/S}$, $Y_{P/X}$.

5. **Створення блоку для моделювання динаміки швидкості росту (μ):** Використовуйте блоки **Product**, **Divide**, та **Sum** для побудови рівняння швидкості росту: $\mu=\mu_{max}\cdot S/K_s+S$. Підключіть цей блок до інших рівнянь для коректного обчислення швидкості росту в системі.
6. **Налаштування початкових умов:** Використайте блоки **Constant** для задання початкових умов для концентрацій X_0, S_0, і P_0.
7. **Блок для виведення результатів:** Додайте блоки **Scope** або **XY Graph** для візуалізації змін концентрації біомаси, субстрату та продукту під час симуляції.

Крок 2: Запуск симуляції та аналіз результатів

1. **Налаштування параметрів симуляції:** Встановіть час симуляції (наприклад, 0–50 годин) та виберіть крок дискретизації (наприклад, 0.1 години).
2. **Запуск моделі:** Натисніть на кнопку запуску (Play) для виконання симуляції. Спостерігайте за результатами на графіках **Scope**, де буде показано зміни концентрацій X, S та P.
3. **Аналіз отриманих графіків:**

- **Біомаса (X)**: Зазвичай біомаса зростає на початкових етапах, потім стабілізується.
- **Субстрат (S)**: Його концентрація поступово зменшується у зв'язку з перетворенням на біомасу та продукт.
- **Продукт (P)**: Збільшується пропорційно до зростання біомаси.

Крок 3: Оптимізація процесу біореактора

1. Цільова функція для оптимізації: Виберіть критерії для оптимізації процесу. Це може бути:

- *Максимізація виходу продукту* — досягнення максимальної концентрації продукту за мінімальний час.

- *Мінімізація витрат субстрату* — зменшення кількості використаного субстрату для досягнення певної концентрації продукту.
- *Оптимізація часу процесу* — мінімізація часу досягнення стаціонарної фази.

2. **Налаштування параметрів оптимізації:** Використайте параметри, такі як μ_{max}, K_s, початкові концентрації субстрату або аерації, як змінні для оптимізації. У MATLAB ви можете використовувати функції оптимізації, такі як **fmincon** або **ga** (генетичні алгоритми), для пошуку оптимальних параметрів.
3. **Запуск симуляції з оптимізованими параметрами:** Після виконання оптимізації повторно запустіть симуляцію з новими параметрами та оцініть результати.

Крок 4: Візуалізація та звітність

1. Графіки результатів: Створіть графіки концентрацій біомаси, субстрату та продукту до і після оптимізації. Вони допоможуть наочно побачити ефект оптимізації.
2. Звітність: Оформіть результати симуляцій та оптимізації у вигляді звіту. Зробіть висновки про ефективність роботи біореактора та можливі шляхи подальшого покращення процесу.

Висновки:

1. Аналіз моделі: В результаті виконання практичної роботи ви побудуєте модель біореактора у середовищі Simulink і проаналізуєте її поведінку на основі зміни ключових параметрів.
2. Оптимізація: Ви навчитеся оптимізувати процеси в біореакторі за допомогою алгоритмів MATLAB, що дозволить підвищити ефективність роботи біотехнологічних процесів.
3. Практичне застосування: Отримані результати можна застосовувати для підвищення продуктивності біотехнологічних установок та мінімізації витрат ресурсів.

ПРАКТИЧНА РОБОТА 4.
ВИКОРИСТАННЯ ГЕНЕТИЧНИХ АЛГОРИТМІВ ДЛЯ ОПТИМІЗАЦІЇ ФЕРМЕНТАЦІЙНОГО ПРОЦЕСУ

Мета: Навчитися використовувати генетичні алгоритми (GA) для оптимізації параметрів ферментаційного процесу, таких як

концентрація субстрату, температура або pH, з метою максимізації виходу продукту або швидкості росту мікроорганізмів.

Теоретичні основи:

Генетичні алгоритми (GA) — це еволюційний підхід до оптимізації, заснований на ідеях природного добору та генетики. Основними етапами GA є:

1. Ініціалізація популяції: створення початкової групи рішень (хромосом).
2. Оцінка: обчислення цільової функції для кожної хромосоми.
3. Селекція: вибір хромосом для створення нового покоління на основі їхньої "пристосованості".
4. Кросовер (рекомбінація): обмін генетичною інформацією між хромосомами.
5. Мутація: випадкові зміни в хромосомах для забезпечення різноманітності.
6. Завершення: алгоритм завершується, коли досягнуто оптимального рішення або після заданої кількості поколінь.

Цільова функція: У ферментаційному процесі цільовою функцією може бути максимізація концентрації продукту P, що визначається такими рівняннями:

- Для біомаси (мікроорганізмів):

$$\frac{dX}{dt} = \mu X$$

- Для субстрату:

$$\frac{dS}{dt} = -\frac{1}{Y_{x/s}}\mu X$$

- Для продукту:

$$\frac{dP}{dt} = Y^{p}/_{x}\,\mu X$$

- Модель Монода:

$$\mu = \mu_{max}\frac{S}{K_s + S}$$

$Y_{X/S}$ — коефіцієнт перетворення субстрату в біомасу, $Y_{P/X}$ — коефіцієнт утворення продукту з біомаси.

Хід роботи

1. Реалізувати генетичний алгоритм для оптимізації параметрів ферментації (наприклад, μ_{max}, Ks, $Y_{X/S}$, T, pH).
2. Використати цільову функцію, яка максимізує концентрацію продукту P.
3. Провести симуляцію та проаналізувати результати.

Хід виконання:

Крок 1: Визначення моделі ферментаційного процесу

1. Реалізуйте модель ферментаційного процесу за допомогою системи диференціальних рівнянь.
2. Створіть функцію для обчислення концентрації продукту P у кінцевий момент часу.

```python
import numpy as np
from scipy.integrate import odeint

# Параметри ферментації
def fermentation_model(Y, t, mu_max, Ks, Yxs, Ypx):
    X, S, P = Y
    mu = mu_max * S / (Ks + S)  # Швидкість росту за Монодом
    dXdt = mu * X  # Зміна біомаси
    dSdt = -1 / Yxs * mu * X  # Зміна субстрату
    dPdt = Ypx * mu * X  # Зміна продукту
    return [dXdt, dSdt, dPdt]

# Функція для симуляції процесу
def simulate_fermentation(mu_max, Ks, Yxs, Ypx, X0, S0, P0, t):
    Y0 = [X0, S0, P0]  # Початкові умови
    result = odeint(fermentation_model, Y0, t, args=(mu_max, Ks, Yxs, Ypx))
    return result[:, 2]  # Концентрація продукту (P)
```

Крок 2: Реалізація генетичного алгоритму

- **Кодування рішень (хромосом):** Параметри μ_{max}, Ks, $Y_{X/S}$, $Y_{P/X}$ кодуються у вигляді векторів. Кожна хромосома представляє один набір значень цих параметрів.
- **Цільова функція:** Максимізація концентрації продукту P у кінцевий момент часу.

```python
# Цільова функція
def objective_function(params):
    mu_max, Ks, Yxs, Ypx = params
    X0, S0, P0 = 0.1, 5, 0  # Початкові умови
    t = np.linspace(0, 50, 100)  # Час симуляції
    P = simulate_fermentation(mu_max, Ks, Yxs, Ypx, X0, S0, P0, t)
    return -P[-1]  # Негативне значення, оскільки GA мінімізує функцію
```

3. Ініціалізація популяції: Створіть початкову групу хромосом із випадковим значенням параметрів.

```python
# Ініціалізація популяції
def initialize_population(pop_size, bounds):
    return np.random.uniform(bounds[:, 0], bounds[:, 1], (pop_size, bounds.shape[0]))
```

4. Оцінка пристосованості: Обчисліть цільову функцію для кожної хромосоми.

```python
# Оцінка пристосованості
def evaluate_population(population):
    fitness = []
    for individual in population:
        fitness.append(objective_function(individual))
    return np.array(fitness)
```

5. Селекція, кросовер і мутація: Реалізуйте етапи селекції, схрещування та мутації.

```python
# Селекція
def select_parents(population, fitness, num_parents):
    parents_idx = np.argsort(fitness)[:num_parents]
    return population[parents_idx]

# Кросовер
def crossover(parents, offspring_size):
    offspring = np.empty((offspring_size, parents.shape[1]))
    for k in range(offspring_size):
        p1_idx, p2_idx = np.random.choice(parents.shape[0], 2, replace=False)
        crossover_point = np.random.randint(1, parents.shape[1])
        offspring[k, :crossover_point] = parents[p1_idx, :crossover_point]
        offspring[k, crossover_point:] = parents[p2_idx, crossover_point:]
    return offspring

# Мутація
def mutate(offspring, bounds, mutation_rate=0.1):
    for idx in range(offspring.shape[0]):
        if np.random.rand() < mutation_rate:
            gene_idx = np.random.randint(0, offspring.shape[1])
            offspring[idx, gene_idx] = np.random.uniform(bounds[gene_idx, 0], bounds[gene_idx
    return offspring
```

6. Алгоритм GA:

```python
# Генетичний алгоритм
def genetic_algorithm(bounds, pop_size, num_generations, mutation_rate):
    population = initialize_population(pop_size, bounds)
    for generation in range(num_generations):
        fitness = evaluate_population(population)
        parents = select_parents(population, fitness, num_parents=pop_size // 2)
        offspring_crossover = crossover(parents, offspring_size=pop_size - parents.shape[0])
        offspring_mutation = mutate(offspring_crossover, bounds, mutation_rate)
        population[:parents.shape[0]] = parents
        population[parents.shape[0]:] = offspring_mutation
        print(f"Generation {generation}: Best fitness = {-np.min(fitness)}")
    best_solution_idx = np.argmin(fitness)
    return population[best_solution_idx], -fitness[best_solution_idx]
```

Крок 3. Виконання оптимізації.

1. Задайте межі параметрів

```python
bounds = np.array([
    [0.1, 1.0],  # mu_max
    [0.01, 1.0], # Ks
    [0.1, 1.0],  # Yxs
    [0.1, 0.5]   # Ypx
])
```

2. Запустіть генетичний алгоритм

```python
best_params, best_fitness = genetic_algorithm(bounds, pop_size=20, num_generations=50,
print(f"Оптимальні параметри: {best_params}")
print(f"Максимальний вихід продукту: {best_fitness}")
```

Аналіз результатів

1. Отримані параметри: Оптимальні значення як μu_{max}, Ks, $Y_{x/s}$ та $Y_{p/s}$ дозволяють досягти концентрації продукту.
2. Графік результатів: Побудуйте графіки для порівняння динаміки ферментів до і після оптимізації.

```python
# Побудова графіків
t = np.linspace(0, 50, 100)
X0, S0, P0 = 0.1, 5, 0
P_before = simulate_fermentation(0.5, 0.1, 0.5, 0.2, X0, S0, P0, t)
P_after = simulate_fermentation(*best_params, X0, S0, P0, t)

import matplotlib.pyplot as plt
plt.plot(t, P_before, label='До оптимізації')
plt.plot(t, P_after, label='Після оптимізації')
plt.legend()
plt.xlabel('Час (год)')
plt.ylabel('Концентрація продукту (г/л)')
plt.title('Результати оптимізації')
plt.show()
```

Висновок: У результаті роботи студенти дізналися, як застосовувати генетичні алгоритми для оптимізації параметрів ферментації. Вони навчилися працювати з цільовою функцією, реалізовувати основні етапи GA та оцінювати результати оптимізації.

ПРАКТИЧНА РОБОТА №5
МОДЕЛЮВАННЯ ТА ОПТИМІЗАЦІЯ ПРОЦЕСІВ БІОЕНЕРГЕТИКИ (ОТРИМАННЯ БІОГАЗУ, БІОЕТАНОЛУ, БІОВОДНЮ)

Мета роботи: ознайомитися з основами математичного моделювання біоенергетичних процесів; навчитися визначати оптимальні умови для отримання біогазу, біоетанолу та біоводню; використати методи оптимізації для підвищення ефективності біоенергетичних систем.

Теоретичні відомості

Біоенергетика – це напрям біотехнологій, що передбачає отримання енергії з відновлюваної сировини за допомогою мікроорганізмів або ферментативних систем. До основних біопалив відносяться:

- **Біогаз** (метан та CO_2) – утворюється під час анаеробного розкладу органічних речовин.
- **Біоетанол** – отримується шляхом спиртового бродіння біомаси (зерна, цукровмісних рослин).

• **Біоводень** – продукується мікроорганізмами (ціанобактеріями, фототрофними бактеріями) або шляхом термолізу біомаси.

Оптимізація процесів біоенергетики передбачає підбір найкращих умов для максимального виходу кінцевого продукту (наприклад, метану в біогазі або етанолу при ферментації).

Методи оптимізації включають:

• Математичне моделювання (наприклад, рівняння Моно для росту мікроорганізмів).

• Планування експерименту (методи поверхні відгуку, факторний аналіз).

• Алгоритми оптимізації (градієнтні методи, генетичні алгоритми).

Обладнання та програмне забезпечення

• Комп'ютер із встановленими програмами **Matlab**, **Python (Scipy, Pandas, Numpy)** або **Excel**.

• Дані про процес отримання біогазу/біоетанолу (надані викладачем або отримані з експерименту).

• Калькулятор, графічний редактор для побудови залежностей.

Хід роботи

1. Визначення вихідних параметрів процесу

Отримати та проаналізувати початкові дані:

• Вид біопалива: біогаз, біоетанол, біоводень.

• Склад сировини (відходи сільського господарства, целюлоза, органічні відходи).

• Основні параметри процесу: температура, pH, концентрація субстрату, час ферментації.

2. Побудова математичної моделі

Використати кінетичні рівняння для опису процесу. Наприклад, для анаеробного зброджування (отримання біогазу) можна застосувати рівняння:

$$\mu = \mu_{max} \frac{S}{K_s + S}$$

де:

• μ – питома швидкість росту бактерій,

• μ_{max} – максимальна швидкість росту,

- S – концентрація субстрату,
- Ks – константа насичення.

Для біоетанолу використовується рівняння спиртового бродіння:

$$S \xrightarrow{\text{ферменти}} P + CO_2$$

де S – глюкоза, P – етанол.

3. Оптимізація процесу

Провести оптимізацію одного з параметрів:

- Для біогазу – температура процесу (мезофільний режим 35-40°C або термофільний 50-55°C).
- Для біоетанолу – концентрація глюкози у середовищі.
- Для біоводню – інтенсивність освітлення (якщо використовується фотосинтетичний процес).

Методи оптимізації:

- Використати метод градієнтного спуску для пошуку найкращих параметрів.
- Застосувати метод планування експерименту (2D-графік залежності виходу біопалива від параметрів).

4. Аналіз отриманих результатів

- Побудувати графік залежності виходу біопалива від оптимізованого параметра.
- Порівняти результати із реальними даними (експериментальними або літературними).
- Зробити висновки щодо ефективності процесу та можливості його покращення.

Контрольні запитання

1. Які основні види біопалива використовуються в біоенергетиці?
2. Як впливає температура на вихід біогазу?
3. Які математичні моделі описують процеси біоенергетики?
4. Які методи оптимізації використовуються для покращення виходу біопалива?
5. Як можна автоматизувати процеси отримання біопалива?

ПРАКТИЧНА РОБОТА №6
РОЗРОБКА АЛГОРИТМІВ КЕРУВАННЯ БІОТЕХНОЛОГІЧНИМИ ПРОЦЕСАМИ (МЕТОДИ PID-РЕГУЛЮВАННЯ, НЕЧІТКІ ЛОГІЧНІ СИСТЕМИ)

Мета роботи: ознайомитися з основними методами автоматичного керування біотехнологічними процесами. Освоїти PID-регулювання та нечіткі логічні системи для управління біопроцесами. Навчитися будувати моделі керування параметрами біореакторів (температура, pH, розчинений кисень). Реалізувати простий алгоритм керування у середовищі Matlab/Simulink**,** Python (Control Systems, Sciру) або LabVIEW.

Теоретичні відомості

Методи керування біотехнологічними процесами

Автоматизація у біотехнологіях потрібна для підтримки оптимальних умов культивування мікроорганізмів або клітин. Основні параметри, що підлягають контролю:

- **Температура** (впливає на швидкість ферментативних реакцій).
- **pH середовища** (важливий для життєдіяльності мікроорганізмів).
- **Рівень розчиненого кисню (DO – Dissolved Oxygen)** (важливий у аеробних процесах).
- **Подавання субстрату та видалення продуктів реакції** (забезпечує стабільний ріст клітин).

Для стабільного керування цими параметрами використовуються **PID-регулятори** та **нечіткі логічні системи.**

PID-регулювання

Пропорційно-інтегрально-диференціальне (PID) керування – один із найпоширеніших методів у біотехнологічній автоматизації.

Формула PID-регулятора:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}$$

де:

- u(t) – керуючий сигнал,
- e(t) – відхилення фактичного значення від заданого,
- Kp – коефіцієнт пропорційної складової,

- Ki – коефіцієнт інтегральної складової (усуває статичну похибку),
- Kd – коефіцієнт диференціальної складової (пригнічує швидкі зміни).

Приклад: У біореакторі необхідно підтримувати температуру 37°C. Якщо температура падає, нагрівач активується, а якщо перевищує норму – вимикається або вмикається охолодження.

Нечітке логічне керування (Fuzzy Logic Control, FLC)

Метод нечіткої логіки застосовується для процесів, які складно описати рівняннями.

Ключові етапи:

1. ***Фазифікація*:** перетворення вхідних змінних (наприклад, "низька", "середня", "висока" температура).
2. ***База правил:*** набір логічних правил (IF-THEN).
3. ***Дефазифікація*:** перетворення результату в конкретне значення для виконавчого механізму.

Приклад: Якщо температура низька, збільшити потужність нагрівача; якщо висока – зменшити.

Обладнання та програмне забезпечення

- Matlab/Simulink або Python (Control Systems, Scipy, Fuzzy Logic Toolbox)
- Дані про зміну температури, pH або кисню в біореакторі
- Симуляційне середовище або реальний датчик (наприклад, Arduino з термодатчиком DS18B20)

Хід роботи

1. Побудова математичної моделі біопроцесу

Розглянемо **температурний контроль біореактора**. Для опису процесу можна використати рівняння теплового балансу:

$$C_p m \frac{dT}{dt} = Q_{\text{нагрівача}} - Q_{\text{втрат}}$$

де:

- Ср – теплоємність середовища,
- m – маса середовища,
- T – температура,
- Qнагрівача – потужність нагріву,
- Qвтрат – теплові втрати.

2. Реалізація PID-регулятора

1. Відкрийте **Matlab/Simulink** або Python.
2. Використайте функцію PID-контролю:

У Matlab:

```matlab
s = tf('s');
P = 1/(10*s + 1); % Передаточна функція об'єкта
C = pid(1, 0.1, 0.01); % PID-регулятор (Kp=1, Ki=0.1, Kd=0.01)
T = feedback(C*P, 1);
step(T) % Перевірка роботи регулятора
```

У Python (Scipy):

```python
import control as ctrl
import matplotlib.pyplot as plt

# Передаточна функція об'єкта
P = ctrl.tf([1], [10, 1])

# Налаштування PID-регулятора
Kp, Ki, Kd = 1, 0.1, 0.01
C = ctrl.tf([Kd, Kp, Ki], [1, 0])

# Замкнена система
T = ctrl.feedback(C * P)

# Побудова графіка
t, y = ctrl.step_response(T)
plt.plot(t, y)
plt.xlabel("Час")
plt.ylabel("Температура")
plt.show()
```

3. Реалізація нечіткої логіки (Fuzzy Logic Control)

1. Створіть **Fuzzy Logic Controller** у Matlab або Python.
2. Задайте нечіткі змінні:
   - Вхід: відхилення температури (низьке, нормальне, високе).
   - Вихід: потужність нагрівача.
3. Створіть базу правил:
   - Якщо температура низька, збільшити нагрів.
   - Якщо температура нормальна, не змінювати.
   - Якщо температура висока, зменшити нагрів.

У Python (skfuzzy):

```python
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

temp = ctrl.Antecedent(np.arange(30, 45, 1), 'temp')
power = ctrl.Consequent(np.arange(0, 100, 1), 'power')

temp['low'] = fuzz.trimf(temp.universe, [30, 32, 35])
temp['normal'] = fuzz.trimf(temp.universe, [34, 37, 40])
temp['high'] = fuzz.trimf(temp.universe, [38, 42, 45])

power['low'] = fuzz.trimf(power.universe, [0, 20, 40])
power['medium'] = fuzz.trimf(power.universe, [30, 50, 70])
power['high'] = fuzz.trimf(power.universe, [60, 80, 100])

rule1 = ctrl.Rule(temp['low'], power['high'])
rule2 = ctrl.Rule(temp['normal'], power['medium'])
rule3 = ctrl.Rule(temp['high'], power['low'])

control_system = ctrl.ControlSystem([rule1, rule2, rule3])
controller = ctrl.ControlSystemSimulation(control_system)
controller.input['temp'] = 36
controller.compute()
print(controller.output['power'])
```

Контрольні запитання

1. Що таке PID-регулятор?
2. У яких біотехнологічних процесах застосовується нечітке керування?
3. Як налаштовують коефіцієнти Kp, Ki, Kd?
4. Які переваги нечіткої логіки перед PID-регулюванням?

ПРАКТИЧНА РОБОТА №7
СИСТЕМИ КОМП'ЮТЕРНОГО ЗОРУ В БІОРОБОТОТЕХНІЦІ (АНАЛІЗ ЗОБРАЖЕНЬ ДЛЯ КОНТРОЛЮ БІОПРОЦЕСІВ)

Мета роботи: ознайомитися з основами комп'ютерного зору та його застосуванням у біотехнологіях, освоїти методи аналізу зображень для контролю біопроцесів, використати OpenCV для обробки мікроскопічних зображень (наприклад, підрахунок клітин, аналіз форми колоній мікроорганізмів). Реалізувати

програму для автоматичного розпізнавання та сегментації об'єктів на біологічних зображеннях.

Теоретичні відомості

Комп'ютерний зір *(Computer Vision)* – це технологія аналізу зображень для автоматизації процесів у різних сферах, зокрема у біоробототехніці та біотехнологіях.

Основні застосування:

- Автоматичний підрахунок клітин та колоній мікроорганізмів**.**
- Контроль росту біоплівок у біореакторах.
- Розпізнавання форми та кольору бактерій для діагностики інфекцій.
- Аналіз руху клітин у мікроскопічних зображеннях (наприклад, для дослідження поведінки бактерій).
- Розпізнавання дефектів у біопродуктах за допомогою камер у виробничих лініях.

Основні методи комп'ютерного зору

1. Обробка зображень:
   - Перетворення у відтінки сірого (Grayscale)
   - Фільтрація шумів (Gaussian Blur, Median Blur)
   - Контрастність та порогова обробка (Thresholding)
2. Сегментація об'єктів:
   - Бінаризація зображення (Adaptive Threshold, Otsu Threshold)
   - Виділення контурів (Canny Edge Detection)
   - Сегментація методом Watershed для розділення злиплих колоній
3. Аналіз форми та підрахунок об'єктів:
   - Функції OpenCV для знаходження контурів **(**cv2.findContours**)**
   - Визначення площі, периметра, округлості колоній
   - Автоматичний підрахунок бактерій

Обладнання та програмне забезпечення

- Комп’ютер із Python (бібліотеки OpenCV, NumPy, Matplotlib, Scikit-Image)
- Зображення колоній бактерій або клітин, отримані через мікроскоп або завантажені з відкритих баз
- (За можливості) Цифровий мікроскоп або камера для реального збору зображень

Хід роботи

1. Завантаження та обробка зображення

- Встановлення бібліотек (якщо не встановлено):

```bash
pip install opencv-python numpy matplotlib scikit-image
```

- Завантаження та конвертація у відтінки сірого:

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Завантажуємо зображення
image = cv2.imread('colonies.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Відображення
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1), plt.imshow(image, cmap='gray'), plt.title('Оригінал')
plt.subplot(1, 2, 2), plt.imshow(gray, cmap='gray'), plt.title('Чорно-біле')
plt.show()
```

2. Фільтрація шумів та сегментація колоній

1. Застосування розмиття для згладжування шумів:

```python
blurred = cv2.GaussianBlur(gray, (5,5), 0)
```

2. Порогова обробка для виділення колоній:

```python
_, thresh = cv2.threshold(blurred, 127, 255, cv2.THRESH_BINARY_INV)
plt.imshow(thresh, cmap='gray'), plt.title('Порогова обробка')
plt.show()
```

3. Виділення контурів колоній:

```python
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(image, contours, -1, (0, 255, 0), 2)

plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)), plt.title('Виділені колонії')
plt.show()
```

3. Підрахунок кількості бактерій або колоній

```python
print(f"Знайдено колоній: {len(contours)}")
```

4. Визначення характеристик колоній

```python
for contour in contours:
    area = cv2.contourArea(contour)  # Площа колонії
    perimeter = cv2.arcLength(contour, True)  # Периметр колонії
    print(f"Колонія: площа = {area}, периметр = {perimeter}")
```

5. Сегментація зображення методом Watershed (для злиплих колоній)

```python
# Використання розмиття та знаходження градієнтів
dist_transform = cv2.distanceTransform(thresh, cv2.DIST_L2, 5)
_, markers = cv2.threshold(dist_transform, 0.7 * dist_transform.max(), 255, 0)

# Конвертація у 8-бітний формат
markers = np.uint8(markers)

# Виконання Watershed
cv2.watershed(image, markers)
image[markers == -1] = [255, 0, 0]  # Виділення меж

plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)), plt.title('Сегментовані колонії')
plt.show()
```

Контрольні запитання

1. Які методи комп'ютерного зору використовуються у біотехнологіях?
2. Як працює сегментація Watershed?
3. Які параметри колоній можна аналізувати?
4. Як можна використовувати комп'ютерний зір у виробничих біотехнологічних процесах?

ПРАКТИЧНА РОБОТА №8. ОПТИМІЗАЦІЯ РОБОТИЗОВАНИХ ПЛАТФОРМ ДЛЯ БІОЛОГІЧНОГО СИНТЕЗУ (ПЛАНУВАННЯ РУХІВ МАНІПУЛЯТОРІВ)

Мета роботи: ознайомитися з принципами робототехнічних систем у біотехнології. Освоїти методи кінематики та динаміки маніпуляторів. Використати методи оптимізації для покращення траєкторії руху роботизованих платформ. Реалізувати симуляцію руху маніпулятора у середовищі Python (ROS, Robotics Toolbox, NumPy) або Matlab/Simulink.

Теоретичні відомості

Роботизовані платформи широко використовуються у біотехнологічних процесах для:

- Автоматизованого дозування рідин (піпетувальні роботи).
- Маніпуляцій з біологічними пробами (в лабораторіях молекулярної біології).
- Біопринтингу (друк живих клітин для тканинної інженерії).
- Автоматизованого культивування клітин (керування біореакторами).

Для ефективної роботи маніпуляторів необхідно оптимізувати їх траєкторію, щоб зменшити витрати енергії, час виконання операцій і покращити точність.

Основи кінематики маніпуляторів

Маніпулятор можна розглядати як **ланцюг з'єднаних ланок** із рухомими суглобами.

Важливі поняття:

- *Прямокутна кінематика* – розрахунок положення робочого інструмента за заданими кутами з'єднань.
- *Зворотна кінематика* – визначення кутів з'єднань для досягнення певної точки в просторі.

Формула перетворення координат у дволанковому маніпуляторі:

$$x=L_1cos(\theta_1)+L_2cos(\theta_1+\theta_2)$$
$$y=L_1sin(\theta_1)+L_2sin(\theta_1+\theta_2)$$

де:

- L_1, L_2 – довжини ланок,
- θ_1, θ_2 – кути обертання суглобів.

Оптимізація руху маніпулятора

Оптимізація траєкторії дозволяє зменшити:

- Час виконання операції**.**
- Споживання енергії (мінімізація моментів у суглобах).
- Коливання робота (забезпечення плавного руху).

Методи оптимізації:

1. ***Генетичні алгоритми*** – пошук найкращої траєкторії шляхом природного відбору.

2. ***Метод градієнтного спуску*** – оптимізація траєкторії для мінімізації витрат енергії.

3. ***Методи динамічного програмування*** – розрахунок найкращої послідовності рухів.

Обладнання та програмне забезпечення:

- Python (NumPy, SciPy, Matplotlib, Robotics Toolbox) або Matlab/Simulink
- ROS (Robot Operating System) для реальної взаємодії з роботами
- Симулятор Gazebo або V-REP (за бажанням)

Хід роботи

1. Моделювання маніпулятора

1. Задати параметри маніпулятора:

```python
import numpy as np
import matplotlib.pyplot as plt

# Довжини ланок
L1, L2 = 10, 7

# Кути обертання (в градусах)
theta1 = np.radians(30)
theta2 = np.radians(45)

# Обчислення координат кінцевої точки
x = L1 * np.cos(theta1) + L2 * np.cos(theta1 + theta2)
y = L1 * np.sin(theta1) + L2 * np.sin(theta1 + theta2)

print(f"Кінцева координата: ({x:.2f}, {y:.2f})")
```

2. Візуалізація маніпулятора:

```python
def plot_robot(theta1, theta2):
    x0, y0 = 0, 0  # Початкова точка
    x1, y1 = L1 * np.cos(theta1), L1 * np.sin(theta1)
    x2, y2 = x1 + L2 * np.cos(theta1 + theta2), y1 + L2 * np.sin(theta1 + theta2)

    plt.plot([x0, x1], [y0, y1], 'bo-', linewidth=3)
    plt.plot([x1, x2], [y1, y2], 'ro-', linewidth=3)
    plt.xlim(-20, 20)
    plt.ylim(-20, 20)
    plt.grid(True)
    plt.show()

plot_robot(theta1, theta2)
```

2. Оптимізація руху маніпулятора

1. Використання градієнтного методу для знаходження оптимальних кутів:

```python
from scipy.optimize import minimize

def energy_cost(angles):
    theta1, theta2 = angles
    x = L1 * np.cos(theta1) + L2 * np.cos(theta1 + theta2)
    y = L1 * np.sin(theta1) + L2 * np.sin(theta1 + theta2)
    return np.abs(x - 10) + np.abs(y - 5)  # Мінімізуємо відстань до цілі (10,5)

result = minimize(energy_cost, [np.radians(20), np.radians(30)])
opt_theta1, opt_theta2 = result.x

print(f"Оптимальні кути: {np.degrees(opt_theta1):.2f}, {np.degrees(opt_theta2):.2f}")
plot_robot(opt_theta1, opt_theta2)
```

Контрольні запитання

1. Що таке кінематика маніпулятора?
2. Які методи можна використовувати для оптимізації руху маніпуляторів?
3. Як можна розширити алгоритм для роботи з реальними роботами (ROS, Arduino, UR5, KUKA)?
4. Які критерії важливі для біоробототехнічних платформ?

ПРАКТИЧНА РОБОТА №9. ЗАСТОСУВАННЯ ЦИФРОВИХ ДВІЙНИКІВ У БІОТЕХНОЛОГІЧНИХ ПРОЦЕСАХ (ІМІТАЦІЙНЕ МОДЕЛЮВАННЯ БІОПРОЦЕСІВ)

Мета роботи: ознайомитися з поняттям цифрового двійника та його застосуванням у біотехнологіях. Освоїти методи імітаційного моделювання біопроцесів. Використати математичні моделі для створення цифрового двійника біореактора. Реалізувати цифровий двійник біопроцесу у середовищі Python (SciPy, SimPy, Matplotlib) **або** Matlab/Simulink.

Теоретичні відомості

Цифровий двійник (Digital Twin, DT) – це віртуальна копія фізичного процесу, яка отримує реальні дані у режимі реального часу, аналізує їх і допомагає оптимізувати процес.

Застосування цифрових двійників у біотехнологіях

- *Оптимізація біореакторів* (регулювання подачі кисню, pH, температури).
- *Прогнозування росту клітин* (імітація біологічних реакцій у культуральних середовищах).
- *Контроль ферментації* (оцінка виходу біопродуктів залежно від параметрів).
- *Автоматизація виробництва* (зменшення експериментальних витрат через віртуальне тестування).

Структура цифрового двійника біопроцесу

1. Фізична система (біореактор, ферментатор).
2. Математична модель (диференціальні рівняння для опису росту клітин).
3. Дані реального часу (з датчиків температури, pH, кисню).
4. Алгоритми аналізу та оптимізації (імітація роботи та прогнозування змін).

Обладнання та програмне забезпечення

- Python (SciPy, NumPy, Matplotlib, SimPy) **або** Matlab/Simulink
- База даних експериментальних значень біопроцесів (отримана від викладача або створена самостійно)
- (Опціонально) Датчики для збору реальних даних

Хід роботи

1. Побудова математичної моделі біопроцесу

Розглянемо модель росту мікроорганізмів у біореакторі. Основне рівняння логістичного росту клітин**:**

$$\frac{dX}{dt} = \mu X \left(1 - \frac{X}{X_{max}}\right)$$

де:

- X – концентрація біомаси (г/л),
- μ_{max} – максимальна швидкість росту (год$^{-1}$),
- X_{max} – гранична концентрація біомаси (г/л).

Реалізація в Python:

python Копіювати Редагувати

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

# Параметри моделі
mu_max = 0.4  # Максимальна швидкість росту (год^-1)
X_max = 10  # Гранична концентрація біомаси (г/л)
X0 = 0.1  # Початкова концентрація біомаси (г/л)
t = np.linspace(0, 50, 100)  # Час у годинах

# Диференціальне рівняння
def growth_model(X, t, mu_max, X_max):
    return mu_max * X * (1 - X / X_max)

# Розв'язання рівняння
X_sol = odeint(growth_model, X0, t, args=(mu_max, X_max))

# Відображення результатів
plt.plot(t, X_sol, label="Концентрація біомаси")
plt.xlabel("Час (год)")
plt.ylabel("Концентрація біомаси (г/л)")
plt.legend()
plt.grid()
plt.show()
```

2. Додавання параметра подачі субстрату

У біотехнологіях **подача субстрату** впливає на ріст клітин. Візьмемо рівняння **Монода**:

$$\mu = \mu_{max} \frac{S}{K_s + S}$$

де:

- S – концентрація субстрату (г/л),
- Ks – константа насичення (г/л).

Реалізація у Python:

python Копіювати Редагувати

```python
def monod_model(y, t, mu_max, K_s, S0):
    X, S = y
    mu = mu_max * S / (K_s + S)  # Швидкість росту
    dXdt = mu * X  # Зміна біомаси
    dSdt = -dXdt / 0.5  # Споживання субстрату
    return [dXdt, dSdt]

# Початкові умови
X0, S0 = 0.1, 5  # Початкова біомаса і субстрат
K_s = 1  # Константа насичення

# Часова шкала
t = np.linspace(0, 50, 100)

# Розв'язання рівнянь
sol = odeint(monod_model, [X0, S0], t, args=(mu_max, K_s, S0))

# Відображення результатів
plt.plot(t, sol[:, 0], label="Біомаса (г/л)")
plt.plot(t, sol[:, 1], label="Субстрат (г/л)", linestyle="dashed")
plt.xlabel("Час (год)")
plt.ylabel("Концентрація (г/л)")
plt.legend()
plt.grid()
plt.show()
```

3. Оптимізація цифрового двійника

1. Додавання реальних даних:
   - Завантажити реальні дані про ріст клітин із бази даних.
   - Порівняти модельні результати з реальними значеннями.
2. Калібрування параметрів:
   - Використати методи оптимізації для корекції μmax, Ks.
3. Автоматичне регулювання (PID-контроль подачі субстрату):
   - Використати PID-регулятор для контролю рівня субстрату.

Контрольні запитання

1. Що таке цифровий двійник у біотехнологіях?
2. Як можна покращити точність цифрового двійника біопроцесу?

3. Чому важливо враховувати подачу субстрату при моделюванні росту клітин?
4. Як можна використовувати цифрові двійники для оптимізації біореакторів?

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Біофізична та колоїдна хімія / А. С.Мороз, Л. П.Яворська, Д. Д.Луцевич та ін. Вінниця : НОВА КНИГА, 2007. 600 с.
2. Афанасьєва К. С. Фізичні методи в молекулярній генетиці. Київський університет, 2016. 127 с.
3. Сучасні методи біохімічних досліджень : навчальний посібник / Кучеренко М. Е., Бабенюк Ю. Д. та ін. Київ : Фітосоціоцентр, 2001. 424 с.
4. Мінаєва В. О. Хроматографічний аналіз : підручник для студ. вищ. навч. закл. Черкаси : Вид. від. ЧНУ імені Богдана Хмельницького, 2013. 284 с.
5. Мороз А. С., Луцевич Д. Д., Яворська Л. П. Медична хімія. Вінниця : Нова книга, 2008. 776 с.
6. Колоїдна хімія / Мчедлов-Петросян М. О., Лебідь В. М., Глазкова О. М., Єльцов С. В., Дубина О. М., Панченко В. Г. Харків : Фоліо, 2005. 304 с
7. Біохімія / Остапченко Л. І., Андрійчук Т. Р., Бабенюк Ю. Д. та ін. Київ : ВПЦ «Київ. ун-т», 2012.
8. Важинський С. Е., Щербак Т І. Методика та організація наукових досліджень : навч. посіб. Суми : СумДПУ імені А. С. Макаренка, 2016. 260 с.
9. Сиволоб А. В. Молекулярна біологія. Київ : ВПЦ «Київ. ун-т», 2008.
10. Ушакова Г. О., Тихомиров А. О., Недзвецький В. С. Вивчення методів наукових досліджень у фізіології, біохімії та мікробіології : навч. посіб. Дніпропетровськ : РВВ ДНУ, 2010. 68 с.
11. Федорченко С. В., Курта С. А. Хроматографічні методи аналізу : навч. посіб. Івано-Франківськ : Прикарп. нац. ун-т ім. В. Стефаника, 2012. 146 с.