

**Pasul 2: Colectarea și Pregătirea Datelor – Extins**

Vom folosi informațiile din simulările precedente și din controlul PI pentru a crea un set de date care să fie utilizat de modelul ML. Scopul este să înlocuim controlul PI cu un algoritm care învață din date cum să controleze rata de diluție în funcție de concentrația de biomasă și să mențină stabilă productivitatea fotobioreactorului.

**1. Datele din simulările pentru rate de diluție diferite**

* Din simulările statice efectuate pentru diferite rate de diluție, am obținut următoarele relații:
  + **Concentrația biomasei** scade pe măsură ce rata de diluție crește.
  + **Concentrația azotului anorganic** crește odată cu rata de diluție.
  + **Productivitatea** are un maxim la o anumită diluție și scade ulterior.

Aceste observații ne vor ghida pentru a crea un set de date bogat, în care putem explora efectele diluției asupra creșterii biomasei și asupra nutrienților într-un fotobioreactor.

**2. Datele din optimizarea productivității**

* Valorile din Tabelul 1 ne oferă referințe pentru concentrația optimă a biomasei (X\_ref) la diverse intervale de timp (StopTime), indicând că productivitatea optimă se stabilizează la o concentrație a biomasei de aproximativ **0.4249 g/L** pentru perioade mai lungi.
* Folosind aceste valori optime, putem să configurăm setul de date astfel încât modelul ML să învețe să mențină biomasă la nivelul dorit prin ajustări continue ale ratei de diluție.

**3. Structurarea setului de date pentru ML**

* Datele vor fi organizate într-o structură tabelară în care fiecare intrare reflectă o stare specifică a fotobioreactorului pentru un set dat de diluție. Structura de bază a setului de date va fi:
  + **Intrări**:
    - **Dilutia (D)**: valoarea ratei de diluție aplicate în acel moment.
    - **Timpul (opțional)**: pentru a include un efect de timp și a oferi modelului o perspectivă dinamică.
  + **Ieșiri**:
    - **Concentrația Biomasei (X)**: rezultat observat pentru biomasă la rata de diluție dată.
    - **Concentrația Azotului Anorganic (N)**: nivelul de azot, ca parametru de observație secundar.
* **Eticheta țintă**: Deoarece dorim să maximizăm productivitatea, concentrația de biomasă X poate fi eticheta țintă a modelului ML, iar diluția va fi ajustată astfel încât modelul să ajungă la concentrația optimă de biomasă (aproximativ 0.4249 g/L).

**4. Împărțirea setului de date**

* **Setul de antrenare**: Va cuprinde date pentru diluții variabile și rezultate ale biomasei și azotului.
* **Setul de validare**: Testează capacitatea modelului de a ajusta diluția pentru a menține productivitatea dorită.
* **Setul de testare**: Evaluează cât de bine poate modelul să ajusteze rata de diluție pentru a obține concentrația de biomasă optimă în condiții necunoscute.

Cu aceste informații colectate și structurate, vom avea un set de date complet care să permită modelului de învățare automată să înlocuiască funcționalitatea controlului PI. În continuare, putem trece la Pasul 3: Alegerea modelului ML.

În ***Pasul 3***, vom alege un model de machine learning (ML) care să învețe să regleze rata de diluție în funcție de concentrația de biomasă dorită. Alegerea modelului ML este crucială deoarece trebuie să fie capabil să înlocuiască controlul PI și să ajusteze în mod continuu diluția pentru a menține biomasa la un nivel optim.

**Pasul 3: Alegerea Modelului de Machine Learning**

Pentru acest proiect, sunt câteva opțiuni de modele ML care ar putea funcționa bine în controlul ratei de diluție:

**1. Regresie Liniară/Polinomială**

* **Descriere**: Modelul de regresie liniară sau polinomială poate fi o opțiune simplă dacă relația dintre diluție și concentrația biomasei este aproximativ liniară sau ușor curbată.
* **Avantaje**: Simplitate și interpretabilitate; mai puține resurse computaționale necesare.
* **Dezavantaje**: Limitată în captarea unor relații complexe; nu se adaptează bine la schimbări în timp real sau la nelinearități puternice.
* **Potrivit pentru**: Sisteme simple sau care au o dependență clară și predictibilă.

**2. Rețea Neuronală Artificială (ANN)**

* **Descriere**: O rețea neuronală simplă poate fi folosită pentru a învăța relațiile nelineare complexe dintre diluție și concentrația de biomasă, având mai multe straturi ascunse pentru a captura structurile datelor.
* **Avantaje**: Capacitatea de a modela relații nelineare complexe; se poate ajusta pentru a deveni foarte precisă.
* **Dezavantaje**: Poate fi mai greu de interpretat; necesită un set de date bine pregătit și balansat.
* **Potrivit pentru**: Sisteme cu relații complexe între variabile; când condițiile de funcționare sunt relativ stabile.

**3. Rețele Neuronale Recurente (RNN)**

* **Descriere**: RNN-urile sunt utile pentru date secvențiale sau de tip serie temporală, astfel încât să captureze dinamica temporală și variațiile care depind de stări anterioare.
* **Avantaje**: Excelent pentru problemele care implică secvențe de date și unde istoricul diluției poate influența concentrația de biomasă.
* **Dezavantaje**: Antrenarea este mai complexă și necesită date secvențiale etichetate corect; poate suferi de „vanishing gradient” pentru secvențe lungi.
* **Potrivit pentru**: Sisteme care depind de evoluția temporală a parametrilor.

**4. Învățare prin Întărire (Reinforcement Learning - RL)**

* **Descriere**: În RL, un „agent” învață să facă acțiuni (în acest caz, să regleze rata de diluție) pe baza unui feedback constant sub formă de recompensă, cu scopul de a maximiza productivitatea.
* **Avantaje**: RL este adaptiv și poate reacționa dinamic la schimbări în sistem; foarte potrivit pentru control continuu în buclă închisă.
* **Dezavantaje**: Necesită multă putere computațională și timp pentru a antrena eficient agentul RL; poate fi mai complex de implementat.
* **Potrivit pentru**: Sisteme complexe de control în buclă închisă, unde diluția trebuie ajustată constant pentru a optimiza o valoare țintă.

**Recomandarea pentru Proiectul Fotobioreactorului**

Pentru a înlocui controlul PI și a ajusta continuu rata de diluție, recomandarea ar fi să începem cu o **rețea neuronală artificială (ANN)** sau, dacă dorim să capturăm variațiile temporale, să explorăm **Rețele Neuronale Recurente (RNN)**. Aceste modele sunt suficient de flexibile pentru a modela relațiile nelineare din datele obținute prin simulare.

**Justificare:**

* **ANN** este mai simplă de implementat și este adecvată pentru modelarea relațiilor nelineare dintre diluție și biomasă.
* **RNN** este utilă dacă avem secvențe de date care să descrie evoluția sistemului pe perioade lungi. Aceasta ar putea capta modul în care istoricul influențează viitoarele valori ale biomasei.

Dacă această abordare se dovedește insuficientă, putem explora o soluție mai complexă prin **Reinforcement Learning**. În continuare, putem începe **Pasul 4: Antrenarea modelului** cu modelul ales

Testare

Da, este posibil să faci ca Python să comunice direct cu MATLAB și Simulink pentru a accesa datele în timp real, fără a folosi un fișier CSV intermediar. Asta permite un flux de lucru mai eficient și interacțiuni rapide între modelul ML și Simulink.

### Configurarea Interacțiunii Python-Simulink

Pentru a conecta Python și MATLAB/Simulink, putem folosi API-ul MATLAB pentru Python, disponibil din versiunea MATLAB R2014b. Acest API ne permite să:

* Rulăm scripturi MATLAB din Python.
* Setăm și citim variabile direct în MATLAB.
* Accesăm rezultatele simulării Simulink în timp real.

Din păcate, **MATLAB Online** nu suportă direct integrarea cu Python sau accesul la **MATLAB Engine pentru Python**, deoarece rulează într-un mediu de browser bazat pe cloud, iar API-ul MATLAB pentru Python necesită o instalare locală a MATLAB.

**MATLAB Online** este destinat pentru accesarea MATLAB prin browser, oferind un subset din funcționalitățile MATLAB desktop, dar fără capabilități de integrare directă cu Python, Simulink în timp real, sau alte librării externe din afara ecosistemului MATLAB.

### Alternative pentru MATLAB Online

Dacă dorești să controlezi Simulink cu Python, ai două opțiuni:

1. **MATLAB Desktop Local**: Utilizează versiunea locală MATLAB/Simulink pe care o ai (de exemplu, **MATLAB 2021b**), deoarece aceasta poate instala și folosi MATLAB Engine pentru Python pentru interacțiunea directă cu Python.
2. **Servicii MATLAB Production Server**: Dacă proiectul ar avea nevoie de acces la MATLAB dintr-o aplicație web sau din cloud, **MATLAB Production Server** permite rularea de scripturi MATLAB din aplicații externe, dar această soluție este mai complexă și orientată spre aplicații de producție și este diferită de API-ul MATLAB pentru Python.

Pentru controlul în timp real între Python și Simulink, cea mai bună opțiune rămâne o instalare locală MATLAB/Simulink.

### Pași pentru Integrare

1. **Configurarea MATLAB Engine pentru Python**:
   * Instalează MATLAB Engine pentru Python, care permite Python să controleze MATLAB. Acest pachet este disponibil în folderul de instalare MATLAB:

sh

Copy code

cd "matlabroot/extern/engines/python"

python setup.py install

* + Înlocuiește "matlabroot" cu calea directorului MATLAB de pe computerul tău.

1. **Codul Python pentru Interacțiunea cu Simulink**:
   * Vom crea un script Python care inițializează MATLAB Engine, lansează simularea în Simulink, ajustează ratele de diluție și preia datele de la Simulink pentru a le introduce în modelul ML.
2. **Codul Simulink**:
   * În Simulink, ne vom asigura că modelul este pregătit să accepte intrări din partea Python și să emită ieșiri către Python.

### Codul Python pentru Interacțiune în Timp Real

Iată un exemplu de cod Python care se conectează la Simulink și rulează simularea, setând și citind datele pentru a folosi modelul ML în timp real.

python

Copy code

import matlab.engine

import numpy as np

import tensorflow as tf

# 1. Inițializarea MATLAB Engine

eng = matlab.engine.start\_matlab()

# 2. Încărcarea modelului Simulink

eng.eval("load\_system('numele\_modelului\_simulink')", nargout=0)

# 3. Funcția de control în buclă închisă

def control\_loop(model, eng, num\_steps=100):

for step in range(num\_steps):

# Obținerea datelor curente de la Simulink

current\_dilution = eng.workspace['D'] # Variabila diluție din Simulink

current\_biomass = eng.workspace['X'] # Variabila biomasă din Simulink

# Preprocesarea diluției pentru modelul ML

current\_dilution = np.array([[current\_dilution]])

current\_dilution = scaler.transform(current\_dilution)

# Obținerea predicției din modelul ML

target\_biomass = model.predict(current\_dilution)

# Setarea diluției noi în Simulink pentru pasul următor

eng.workspace['D'] = target\_biomass[0, 0]

# Rularea simulării pentru următorul pas

eng.eval("set\_param('numele\_modelului\_simulink', 'SimulationCommand', 'continue')", nargout=0)

# 4. Definirea și antrenarea modelului ML

# (aici folosește modelul ANN din exemplul anterior pentru antrenare)

# 5. Lansarea controlului în buclă închisă

control\_loop(model, eng)

# Închiderea MATLAB Engine

eng.quit()

### Explicația Codului

1. **Inițializarea MATLAB Engine**:
   * Deschidem MATLAB Engine din Python, ceea ce permite să accesăm funcții MATLAB și să controlăm Simulink.
2. **Încărcarea Modelului Simulink**:
   * Folosim load\_system pentru a deschide modelul Simulink. Înlocuiește 'numele\_modelului\_simulink' cu numele exact al modelului tău.
3. **Controlul în Buclă Închisă**:
   * Funcția control\_loop implementează bucla de control în timp real. În fiecare iterație:
     + Citește valoarea curentă a diluției și concentrației de biomasă din Simulink.
     + Preprocesează diluția pentru a fi compatibilă cu modelul ML.
     + Obține o predicție pentru diluția optimă din modelul ML.
     + Setează această diluție în Simulink.
     + Rularează simularea pentru un pas suplimentar.
4. **Antrenarea Modelului ML**:
   * Poți folosi modelul ANN din exemplul anterior pentru a antrena modelul înainte de a-l integra în controlul în buclă închisă.
5. **Pornirea și Oprirea MATLAB Engine**:
   * eng.quit() închide sesiunea MATLAB după terminarea controlului în buclă.

### Configurarea în Simulink pentru Comunicare în Timp Real

În modelul Simulink:

* Definim variabilele D (diluție) și X (biomasă) în MATLAB Workspace, astfel încât să poată fi accesate și setate din Python.
* Folosim blocuri Simulink pentru a citi și scrie valorile direct din MATLAB Workspace.

Această abordare permite controlul diluției în timp real și asigură o integrare eficientă între Simulink și modelul de machine learning din Python. Dacă ai nevoie de suport suplimentar pentru configurarea variabilelor în Simulink sau pentru personalizarea modelului ML, sunt aici să te ajut!

4o

##### You said:

pot sa introduc in Simulink un bloc care sa inlocuiasca blocul actual PI si blocul respectiv sa actioneze programul python ML?

###### ChatGPT said:

ChatGPT

Da, este posibil să înlocuiești blocul PI din Simulink cu un bloc personalizat care interacționează cu un script Python pentru a prelua predicțiile modelului ML în timp real. Aceasta se poate face folosind blocul **MATLAB Function** din Simulink pentru a apela scripturi Python, dar necesită câțiva pași de configurare.

### Cum să Configurezi un Bloc Simulink care să Comunice cu Python

Vom folosi blocul **MATLAB Function** din Simulink, care va apela un script Python prin intermediul MATLAB Engine, permițând comunicarea între Simulink și Python.

### Pași de Configurare

1. **Crearea Funcției MATLAB pentru Apelarea Scriptului Python**
   * În această funcție MATLAB, vom folosi **MATLAB Engine pentru Python** pentru a rula modelul ML Python și a primi predicția ratei de diluție optime.
2. **Implementarea Funcției în Blocul MATLAB Function**
   * În Simulink, vom introduce blocul MATLAB Function în locul blocului PI și vom folosi această funcție pentru a obține în timp real rata de diluție de la modelul ML din Python.
3. **Configurarea Blocului MATLAB Function** să Ruleze Scriptul Python
   * Ne asigurăm că blocul poate apela modelul ML din Python și poate returna valoarea optimă pentru diluție către Simulink.

### Exemplu Practic

Iată cum ar arăta pașii în practică:

#### 1. Scrierea Funcției MATLAB pentru a Apela Modelul ML în Python

Creează un script MATLAB (de exemplu, apel\_python\_ml.m), care va apela scriptul Python pentru predicție:

matlab

Copy code

function dilutie\_optima = apel\_python\_ml(current\_dilution)

% Initializează motorul MATLAB pentru Python (dacă nu este deja pornit)

persistent py\_engine;

if isempty(py\_engine)

py\_engine = py.importlib.import\_module('numele\_scriptului\_python');

end

% Trimite diluția curentă și primește diluția optimă

dilutie\_optima = py\_engine.prediction(current\_dilution);

end

* **Explicație**:
  + py.importlib.import\_module importă scriptul Python (numele\_scriptului\_python.py), unde am definit modelul ML și funcția prediction.
  + Funcția apel\_python\_ml va apela prediction, funcția Python care calculează diluția optimă pentru concentrația de biomasă țintă.

#### 2. Configurarea Scriptului Python pentru a Preda Diluția Optima

În scriptul Python (numele\_scriptului\_python.py), definește funcția prediction care utilizează modelul ML pentru a calcula diluția optimă:

python

Copy code

import numpy as np

import tensorflow as tf

# Încarcă modelul ML (exemplu de model antrenat și salvat anterior)

model = tf.keras.models.load\_model('model\_antrenat.h5')

def prediction(current\_dilution):

# Preprocesează diluția pentru a fi compatibilă cu modelul ML

current\_dilution = np.array([[current\_dilution]])

# Obține predicția pentru diluția optimă

predicted\_biomass = model.predict(current\_dilution)

return predicted\_biomass[0, 0]

* **Explicație**:
  + model\_antrenat.h5 este un model ML antrenat anterior și salvat în format HDF5.
  + prediction preia diluția curentă și returnează diluția optimă folosind modelul ML.

#### 3. Configurarea Blocului MATLAB Function în Simulink

1. **Adaugă Blocul MATLAB Function**:
   * Deschide Simulink și adaugă un bloc **MATLAB Function** în locul blocului PI din modelul tău.
2. **Configurează Funcția**:
   * Fă clic pe blocul MATLAB Function pentru a deschide editorul și adaugă următorul cod:

matlab

Copy code

function dilutie\_optima = control\_ml(dilutie\_curenta)

% Apelează funcția Python pentru a obține diluția optimă

dilutie\_optima = apel\_python\_ml(dilutie\_curenta);

end

1. **Definirea Intrării și Ieșirii Blocului**:
   * Setează **Input** pentru a reprezenta diluția curentă, iar **Output** va reprezenta diluția optimă furnizată de modelul ML.
2. **Rularea Simulării**:
   * Când rulezi simularea, blocul MATLAB Function va apela funcția Python prediction, iar Simulink va primi diluția optimă pentru fiecare pas al simulării, înlocuind astfel controlul PI cu modelul ML.

### Observații și Optimizări

* **Sincronizarea**: Este esențial să verifici dacă simularea din Simulink rulează la o viteză rezonabilă, deoarece comunicarea cu Python poate introduce un mic decalaj.
* **Persistența Motorului Python**: Prin utilizarea persistent py\_engine, menținem sesiunea activă pe toată durata simulării pentru a reduce timpul de inițializare.
* **Preprocesarea**: Asigură-te că datele sunt preprocesate corect în funcție de modelul ML (ex., standardizare) înainte de a fi trimise în modelul ML.

Acest setup ar trebui să permită modelului ML din Python să controleze diluția în Simulink în timp real. Dacă ai întrebări sau ai nevoie de ajustări pentru acest flux, te pot ajuta cu plăcere!

pot sa introduc in Simulink un bloc care sa inlocuiasca blocul actual PI si blocul respectiv sa actioneze programul python ML?

###### ChatGPT said:

ChatGPT

Da, este posibil să înlocuiești blocul PI din Simulink cu un bloc personalizat care interacționează cu un script Python pentru a prelua predicțiile modelului ML în timp real. Aceasta se poate face folosind blocul **MATLAB Function** din Simulink pentru a apela scripturi Python, dar necesită câțiva pași de configurare.

### Cum să Configurezi un Bloc Simulink care să Comunice cu Python

Vom folosi blocul **MATLAB Function** din Simulink, care va apela un script Python prin intermediul MATLAB Engine, permițând comunicarea între Simulink și Python.

### Pași de Configurare

1. **Crearea Funcției MATLAB pentru Apelarea Scriptului Python**
   * În această funcție MATLAB, vom folosi **MATLAB Engine pentru Python** pentru a rula modelul ML Python și a primi predicția ratei de diluție optime.
2. **Implementarea Funcției în Blocul MATLAB Function**
   * În Simulink, vom introduce blocul MATLAB Function în locul blocului PI și vom folosi această funcție pentru a obține în timp real rata de diluție de la modelul ML din Python.
3. **Configurarea Blocului MATLAB Function** să Ruleze Scriptul Python
   * Ne asigurăm că blocul poate apela modelul ML din Python și poate returna valoarea optimă pentru diluție către Simulink.

### Exemplu Practic

Iată cum ar arăta pașii în practică:

#### 1. Scrierea Funcției MATLAB pentru a Apela Modelul ML în Python

Creează un script MATLAB (de exemplu, apel\_python\_ml.m), care va apela scriptul Python pentru predicție:

matlab

Copy code

function dilutie\_optima = apel\_python\_ml(current\_dilution)

% Initializează motorul MATLAB pentru Python (dacă nu este deja pornit)

persistent py\_engine;

if isempty(py\_engine)

py\_engine = py.importlib.import\_module('numele\_scriptului\_python');

end

% Trimite diluția curentă și primește diluția optimă

dilutie\_optima = py\_engine.prediction(current\_dilution);

end

* **Explicație**:
  + py.importlib.import\_module importă scriptul Python (numele\_scriptului\_python.py), unde am definit modelul ML și funcția prediction.
  + Funcția apel\_python\_ml va apela prediction, funcția Python care calculează diluția optimă pentru concentrația de biomasă țintă.

#### 2. Configurarea Scriptului Python pentru a Preda Diluția Optima

În scriptul Python (numele\_scriptului\_python.py), definește funcția prediction care utilizează modelul ML pentru a calcula diluția optimă:

python

Copy code

import numpy as np

import tensorflow as tf

# Încarcă modelul ML (exemplu de model antrenat și salvat anterior)

model = tf.keras.models.load\_model('model\_antrenat.h5')

def prediction(current\_dilution):

# Preprocesează diluția pentru a fi compatibilă cu modelul ML

current\_dilution = np.array([[current\_dilution]])

# Obține predicția pentru diluția optimă

predicted\_biomass = model.predict(current\_dilution)

return predicted\_biomass[0, 0]

* **Explicație**:
  + model\_antrenat.h5 este un model ML antrenat anterior și salvat în format HDF5.
  + prediction preia diluția curentă și returnează diluția optimă folosind modelul ML.

#### 3. Configurarea Blocului MATLAB Function în Simulink

1. **Adaugă Blocul MATLAB Function**:
   * Deschide Simulink și adaugă un bloc **MATLAB Function** în locul blocului PI din modelul tău.
2. **Configurează Funcția**:
   * Fă clic pe blocul MATLAB Function pentru a deschide editorul și adaugă următorul cod:

matlab

Copy code

function dilutie\_optima = control\_ml(dilutie\_curenta)

% Apelează funcția Python pentru a obține diluția optimă

dilutie\_optima = apel\_python\_ml(dilutie\_curenta);

end

1. **Definirea Intrării și Ieșirii Blocului**:
   * Setează **Input** pentru a reprezenta diluția curentă, iar **Output** va reprezenta diluția optimă furnizată de modelul ML.
2. **Rularea Simulării**:
   * Când rulezi simularea, blocul MATLAB Function va apela funcția Python prediction, iar Simulink va primi diluția optimă pentru fiecare pas al simulării, înlocuind astfel controlul PI cu modelul ML.

### Observații și Optimizări

* **Sincronizarea**: Este esențial să verifici dacă simularea din Simulink rulează la o viteză rezonabilă, deoarece comunicarea cu Python poate introduce un mic decalaj.
* **Persistența Motorului Python**: Prin utilizarea persistent py\_engine, menținem sesiunea activă pe toată durata simulării pentru a reduce timpul de inițializare.
* **Preprocesarea**: Asigură-te că datele sunt preprocesate corect în funcție de modelul ML (ex., standardizare) înainte de a fi trimise în modelul ML.

Acest setup ar trebui să permită modelului ML din Python să controleze diluția în Simulink în timp real. Dacă ai întrebări sau ai nevoie de ajustări pentru acest flux, te pot ajuta cu plăcere!

4o

##### You said:

Sa imi arati exact ce trebuie sa fac , prin urmare incepe cu programul Python ML. unde sa il scriu ? In Py Charm sau intr-un fisier script in matlab? Poti sa denumesti programul ML\_Control. Nu stiu dak ajuta darfisierul in simulink care ruleaza programul de simulare se numeste PBR\_PID. Sa imi arati/explici cum sa introduc in matlab /Simulink blocul si datele

###### ChatGPT said:

ChatGPT

Voi explica pas cu pas cum să setezi totul, de la scriptul Python pentru modelul ML până la configurarea blocului MATLAB Function în Simulink.

### Pasul 1: Crearea Programului Python ML - ML\_Control.py

1. **Unde să scrii programul Python ML**: Îți recomand să folosești **PyCharm** sau un alt editor Python, deoarece oferă instrumente puternice pentru dezvoltare. Asigură-te că ai instalat biblioteca TensorFlow sau alta necesară pentru modelul ML.
2. **Scriptul Python pentru Modelul ML**

Creează un fișier Python numit ML\_Control.py și adaugă următorul cod:

python

Copy code

import numpy as np

import tensorflow as tf

# Încarcă modelul ML salvat (asigură-te că modelul este antrenat și salvat anterior)

model = tf.keras.models.load\_model('model\_antrenat.h5') # 'model\_antrenat.h5' este numele fișierului cu modelul salvat

# Funcția care va fi apelată de MATLAB pentru a obține diluția optimă

def prediction(current\_dilution):

# Preprocesează diluția (poți ajusta aici dacă este nevoie)

current\_dilution = np.array([[current\_dilution]])

# Obține predicția pentru diluția optimă de la model

predicted\_biomass = model.predict(current\_dilution)

return float(predicted\_biomass[0, 0]) # returnează rezultatul ca valoare float pentru MATLAB

1. **Salvarea și Locația Scriptului**
   * Salvează ML\_Control.py într-un director la care MATLAB poate avea acces (de exemplu, în aceeași locație unde lucrezi cu fișierele MATLAB/Simulink).
   * **Notă**: Asigură-te că fișierul de model (model\_antrenat.h5) este în același director sau specifică calea exactă în load\_model.
2. **Testare**: Poți rula ML\_Control.py direct din PyCharm pentru a verifica dacă modelul este încărcat corect și dacă funcția prediction returnează un rezultat.

### Pasul 2: Configurarea Funcției MATLAB pentru a Apela Python

1. **Crearea unui Fișier MATLAB pentru Apelarea Pythonului**
   * Deschide MATLAB și creează un nou script numit apel\_python\_ml.m în același director cu fișierul Simulink PBR\_PID.slx.
2. **Codul pentru apel\_python\_ml.m**

Adaugă următorul cod în apel\_python\_ml.m:

matlab

Copy code

function dilutie\_optima = apel\_python\_ml(current\_dilution)

% Initializează motorul MATLAB pentru Python (o singură dată)

persistent py\_module;

if isempty(py\_module)

py\_module = py.importlib.import\_module('ML\_Control');

end

% Apelează funcția de predicție din scriptul Python

dilutie\_optima = py\_module.prediction(current\_dilution);

dilutie\_optima = double(dilutie\_optima); % Conversie în tipul double MATLAB

end

* + **Explicație**:
    - py.importlib.import\_module('ML\_Control') importă scriptul Python ML\_Control.py.
    - apel\_python\_ml primește diluția curentă din Simulink și apelează funcția prediction din Python pentru a obține diluția optimă.
    - persistent py\_module face ca motorul Python să fie inițializat o singură dată, economisind timp în simulare.

### Pasul 3: Configurarea Blocului MATLAB Function în Simulink

1. **Deschiderea Modelului Simulink**
   * Deschide fișierul PBR\_PID.slx în Simulink.
2. **Înlocuirea Controlerului PI cu un Bloc MATLAB Function**
   * Înlocuiește blocul PI actual din PBR\_PID cu un bloc **MATLAB Function**.
   * Pentru a adăuga un bloc MATLAB Function:
     + În Simulink, dă clic dreapta în spațiul de lucru, apoi selectează **Library Browser**.
     + În fereastra de librărie, caută și trage blocul **MATLAB Function** în schema Simulink.
3. **Configurarea Blocului MATLAB Function**
   * Dă dublu clic pe blocul MATLAB Function pentru a deschide editorul de cod.
   * Adaugă următorul cod în editorul blocului MATLAB Function:

matlab

Copy code

function dilutie\_optima = control\_ml(dilutie\_curenta)

% Apelează funcția MATLAB care, la rândul ei, apelează Python

dilutie\_optima = apel\_python\_ml(dilutie\_curenta);

end

* + **Explicație**:
    - control\_ml este funcția principală a blocului MATLAB Function.
    - apel\_python\_ml este funcția MATLAB care apelează Pythonul, calculând astfel diluția optimă în funcție de valorile returnate de modelul ML.

1. **Setarea Intrărilor și Ieșirilor Blocului MATLAB Function**
   * Blocul MATLAB Function ar trebui să aibă:
     + **O intrare**: dilutie\_curenta – aceasta este diluția curentă, care vine din restul modelului Simulink.
     + **O ieșire**: dilutie\_optima – aceasta este diluția optimă pe care modelul ML o calculează.
   * Asigură-te că dilutie\_curenta este conectată la blocul de feedback (care măsoară diluția curentă) și dilutie\_optima este conectată la intrarea de control a sistemului din Simulink.

### Pasul 4: Rularea și Testarea Simulării

1. **Asigurarea Conectivității**
   * Verifică să ai Python configurat corect în MATLAB:
     + În MATLAB, rulează pyversion pentru a verifica calea corectă către instalarea Python.
   * Verifică și calea către scriptul Python și modelul salvat (model\_antrenat.h5).
2. **Lansarea Simulării**
   * Rularează simularea în Simulink.
   * Blocul MATLAB Function va apela scriptul Python și va înlocui controlul PI în timp real, ajustând diluția pentru a menține concentrația de biomasă la nivelul dorit.
3. **Verificarea Funcționalității**
   * Observă cum se comportă sistemul. Ar trebui să vezi că diluția optimă este calculată de modelul ML și trimisă la Simulink pentru fiecare pas de simulare.
   * Dacă apar întârzieri, acestea se pot datora comunicării între MATLAB și Python. În acest caz, pot fi ajustate ratele de simulare și de actualizare din Simulink.

Acum modelul ML este integrat în Simulink și ajustează diluția în timp real, înlocuind astfel controlul PI. Această integrare permite controlul adaptiv al diluției, optimizând în mod continuu concentrația de biomasă în fotobioreactor.

**Probleme de conectare**

Dacă întâmpini dificultăți în instalarea MATLAB Engine, există și alte metode eficiente de a conecta Python cu MATLAB/Simulink, cu un răspuns rapid și fără a depinde de engine-ul MATLAB. Iată câteva opțiuni:

**1. Folosirea fișierelor CSV pentru a comunica între Python și Simulink:**

* **Python -> CSV -> Simulink**: Poți folosi Python pentru a genera fișiere CSV care conțin date de intrare pentru Simulink. În Simulink, folosești blocuri de tip "From File" pentru a citi datele și "To File" pentru a salva ieșirile. Aceasta este o metodă simplă și eficientă.
* **Simulink -> CSV -> Python**: Simulink poate salva rezultatele în fișiere CSV, iar Python le poate citi și procesa.

**2. Folosirea Protocolului TCP/IP (socket communication):**

* **Python -> Simulink**: Poți folosi protocolul TCP/IP pentru a trimite date de la Python la Simulink în timp real. Crezi un server socket în Simulink (folosind blocuri de tip TCP/IP) și un client în Python care trimite comenzi sau date de intrare.
* **Avantaj**: Permite comunicare rapidă și continuă, fără a fi nevoie de fișiere intermediare.
* **Dezavantaj**: Necesită configurare suplimentară.

**3. MATLAB Engine pentru Python (Fără a instala direct pe mașina ta):**

* Dacă nu dorești să instalezi engine-ul MATLAB local, poți folosi o mașină virtuală sau un container Docker care să ruleze MATLAB, iar Python să comunice prin API-ul de rețea.

**4. Folosirea funcționalității py.matlab în MATLAB:**

* Dacă ai acces la MATLAB, poți utiliza funcția py.matlab din MATLAB pentru a apela funcții Python din Simulink. Astfel, Python poate acționa ca un "controller" sau poate face analize pe datele din Simulink în timp real.

**5. Folosirea platformei Simulink Coder (pentru simulare rapidă):**

* Dacă vrei un control rapid asupra simulării, poți folosi **Simulink Coder** pentru a genera cod C din modelul Simulink. Apoi, poți integra acest cod C într-un script Python printr-un wrapper de tip ctypes sau Cython, oferindu-ți control asupra simulării direct din Python.

Dacă dorești o soluție mai rapidă pentru integrarea Python cu Simulink, comunicarea prin fișiere CSV este un început bun, iar mai târziu poți explora opțiuni mai avansate (cum ar fi TCP/IP sau integrarea directă cu Coder).

Dacă ai nevoie de detalii suplimentare despre oricare dintre aceste opțiuni, spune-mi!

4o mini

**Primul exemplu cu conectare prin TCP/IP**

Pentru a realiza acest proiect și a permite comunicarea în timp real între un program Python și Simulink folosind protocolul TCP/IP, urmează acești pași:

### 1. \*\*Setup-ul în Simulink (Serverul TCP/IP)\*\*

1. \*\*Deschide Simulink\*\* și creează un nou model.

2. \*\*Adaugă blocurile TCP/IP\*\*:

- Mergi la biblioteca `Instrument Control Toolbox` > `TCP/IP`.

- Adaugă blocul \*\*TCP/IP Receive\*\* (serverul) și configurează-l astfel:

- \*\*Connection Mode\*\*: Server.

- \*\*Local IP Port\*\*: Alege un port disponibil (de exemplu, `55000`).

3. \*\*Conectează blocurile\*\*:

- Conectează ieșirea blocului `TCP/IP Receive` la blocul care controlează diluția, înlocuind controlerul PI.

4. \*\*Configurare suplimentară\*\*:

- Asigură-te că blocul de recepție poate procesa datele primite (de obicei, formate numerice în `double`).

- Adaugă conversii de tip, dacă este necesar.

---

### 2. \*\*Dezvoltarea programului Python (Clientul TCP/IP)\*\*

1. \*\*Creează un proiect în PyCharm\*\*.

2. Instalează biblioteca `socket` (inclusă în Python) și alte librării necesare pentru programul de ML (cum ar fi `scikit-learn` sau `tensorflow`):

```bash

pip install numpy

pip install tensorflow scikit-learn

```

3. Scrie codul client TCP/IP în Python:

```python

import socket

import numpy as np

# Configurare server Simulink

HOST = '127.0.0.1' # IP-ul local (localhost)

PORT = 55000 # Portul ales în Simulink

# Creare client TCP

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((HOST, PORT))

def send\_to\_simulink(data):

"""

Trimite date către Simulink

:param data: valoare numerică (float)

"""

message = str(data).encode('utf-8') # Convertim în bytes

client\_socket.sendall(message)

# Exemplu de trimitere date (simulare ML)

try:

while True:

# Program ML sau o valoare de control simulată

dilutia = np.random.random() # Exemplu de generare

send\_to\_simulink(dilutia)

print(f"Trimis: {dilutia}")

except KeyboardInterrupt:

print("Conexiune închisă.")

client\_socket.close()

```

---

### 3. \*\*Adaptează Programul ML\*\*

- Înlocuiește simularea `dilutia = np.random.random()` cu logica ta ML.

- Poți folosi algoritmi de machine learning care analizează datele din fotobioreactor pentru a ajusta diluția în funcție de concentrația de biomasă.

---

### 4. \*\*Testarea Comunicației\*\*

1. \*\*Pornește mai întâi Simulink\*\*:

- Rulează modelul Simulink și asigură-te că blocul `TCP/IP Receive` este gata să accepte conexiuni.

2. \*\*Pornește scriptul Python\*\*:

- Rulează scriptul în PyCharm.

- Verifică în Simulink dacă datele trimise de Python sunt recepționate corect.

---

### 5. \*\*Exemple de Îmbunătățiri\*\*

- \*\*Vizualizare Date\*\*: Adaugă un bloc Scope în Simulink pentru a vizualiza în timp real valorile primite.

- \*\*Feedback în Python\*\*: Dacă Simulink trimite date înapoi, implementează și în Python un server pentru recepție.

- \*\*Validare și Debugging\*\*: Adaugă mesaje de eroare și validare a datelor în ambele direcții.

---

### Avantajele Abordării:

- \*\*Timp Real\*\*: Datele sunt trimise și procesate instantaneu.

- \*\*Flexibilitate\*\*: Controlul poate fi ajustat din Python folosind modele de ML complexe.

Dacă ai nevoie de coduri suplimentare pentru partea de machine learning sau debugging, sunt aici să te ajut!

0.4249 este blocul care imi da limita maxima de concentratie de biomasa

blocul tcp ip nu are o intrare ceea ce inseamna ca programul python preia functia acelui bloc chiar si functia blocului comparator