POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Chimica e dei Processi Sostenibili

Politecnico di Torino

Tesi di Laurea Magistrale

Accoppiamento di Neural ODEs e CFD: modellazione di cinetiche per il processo di Aqueous Phase Reforming

Relatore

Prof. Gianluca BOCCARDO

Correlatori

Prof. Daniele MARCHISIO

Dott.ssa Agnese MARCATO

Dott. Diego FIDA

Candidata

Martina GILARDI

Novembre 2025

Sommario

L'ipotesi di cinetiche analitiche per sistemi reattivi complessi rappresenta una sfida significativa, in particolare quando sono coinvolte numerose reazioni simultanee. Per questo motivo, negli ultimi anni sono stati sviluppati nuovi metodi basati sui dati che permettono di approssimare le cinetiche di reazioni chimiche coinvolte in processi complessi. Tra questi, hanno riscontrato un particolare successo le Neural Ordinary Differential Equations (NODEs), che integrano le reti neurali con i metodi di risoluzione di sistemi di equazioni differenziali ordinarie, rendendoli completamente differenziabili e quindi adatti all'ottimizzazione tramite apprendimento automatico. Le NODEs, infatti, permettono di simulare affidabilmente il comportamento di sistemi dinamici apprendendo le informazioni necessarie da set di dati sperimentali o sintetici, le cui dimensioni sono tipicamente ridotte. Tale limitazione può essere compensata imponendo vincoli di natura cinetica e termodinamica nella struttura delle NODEs stesse.
In questo progetto di tesi questi modelli sono stati implementati per predire la velocità di reazione, quindi la variazione nel tempo delle concentrazioni delle specie chimiche coinvolte nel processo catalitico di Aqueous Phase Reforming (APR) del glicerolo, il principale sottoprodotto derivante dalla sintesi del biodiesel mediante trans-esterificazione di acidi grassi. L'APR del glicerolo è studiato come alternativa sostenibile alle tradizionali tecniche di produzione di idrogeno, che fanno uso di fonti fossili e hanno una richiesta energetica molto elevata. Per questo motivo, tale processo è attualmente al centro di numerose ricerche.
Dopo aver ottenuto un modello in grado di approssimare le cinetiche coinvolte utilizzando la libreria `PyTorch` per effettuare i *training*, lo si è accoppiato a simulazioni CFD finalizzate a riprodurre il reforming del glicerolo in un sistema PFR. A questo scopo, le simulazioni sono state svolte utilizzando il programma `OpenFOAM`. L'obiettivo di questo lavoro è stato quello di verificare l'accuratezza delle NODEs quando utilizzate come modelli cinetici e la fattibilità dell'accoppiamento tra il modello ottenuto con le simulazioni CFD.

Indice

Elenco delle tabelle

Elenco delle figure

Acronimi

APR

Aqueous Phase Reforming

CFD

Computational Fluid Dynamics

CRNN

Chemical Reaction Neural Network

CV

Control Volume

EG

Etilen-glicole

FVM

Finite Volume Method

KCNODE

Kinetics-Constrained Neural Ordinary Differential Equation

KINN

Kinetics-Informed Neural Network

LASSO

Least Absolute Shrinkage and Selection Operator

LHHW

Langmuir-Hinshelwood-Hougen-Watson

MAE

Mean Absolute Error

ML

Machine Learning

MLP

Multi-Layer Perceptron

MSE

Mean Squared Error

NODE

Neural Ordinary Differential Equation

ODE

Ordinary Differential Equation

OpenFOAM

Open-source Field Operation And Manipulation

PG

Propilen-glicole

PINN

Physics-Informed Neural Network

ReLU

Rectified Linear Unit

RMSE

Root Mean Squared Error

SiLU

Sigmoid Linear Unit

Lista dei Simboli

Simbolo	Descrizione	Unità di misura
α	Parametro di regolarizzazione	-
β_i	Coefficienti dei modelli di regressione	-
Γ	Coefficiente di diffusione molecolare generico	m^2/s
ΔH_{sol}	Entalpia di dissoluzione	J/mol
Δp	Perdite di carico	Pa
ϵ	Errore di approssimazione	-
ϵ_0	Porosità del solido catalitico	-
θ	Vettore dei parameri di un MLP	-
μ	Viscosità dinamica	kg/ms
ν	Viscosità cinematica	m^2/s
ρ	Densità	kg/m^3
ϕ	Generica grandezza soggetta ai fenomeni di trasporto nella trattazione FVM	-
A	Fattore pre-esponenziale della legge di Arrhenius	-
C_i	Concentrazione della specie i	mol/m^3
D	Coefficiente di dispersione	m^2/s
D_m	Diffusività molecolare	m^2/s
E_a	Energia di attivazione di una reazione	J/mol
H_s	Costante di Henry	mol/m^3Pa
K	Permeabilità	m^2
L	Lunghezza del reattore	m
$\boldsymbol{M_\nu}$	Matrice dei coefficienti stechiometrici di reazione	-
N	Numero di predizioni	-
Pe	Numero di Péclet	-
R	Costante cinetica dei gas	$J/molK$
S	Superficie generica	m^2
S_i	Termine sorgente nell'equazione di Navier-Stokes	-

S_n	Termine sorgente nell'equazione di advezione-diffusione	-
T	Temperatura assoluta	K
T_m	Temperatura di Arrhenius	mol/J
$\boldsymbol{U}$	Vettore velocità	m/s
U	Velocità nella direzione x	m/s
U_i	Velocità lungo una generica coordinata i	m/s
V	Volume generico	m^3
V	Velocità nella direzione y	m/s
V_L	Volume di liquido nel reattore	m^3
W	Velocità nella direzione z	m/s
a	Termine di *adjoint*	-
b_{ij}	Bias di un MLP	-
c	Numero di Courant	-
d	Numero di diffusione adimensionato	-
$\boldsymbol{d}$	Tensore dei coefficienti di Darcy	$1/m^2$
$\boldsymbol{f}$	Tensore dei coefficienti di Forchheimer	$1/m$
h	Passo di discretizzazione spaziale	m
k	Costante cinetica di reazione generica	-
m_{cat}	Massa di catalizzatore	kg
$\mathbf{n}$	Versore normale ad una superficie	-
p	Pressione	Pa
q_ϕ	Termine sorgente in equazione di conservazione	-
t	Variabile temporale generica	s
w_{ij}	Pesi di un MLP	-
x	Coordinata lungo l'asse delle ascisse	-
$\boldsymbol{x}$	Vettore generico di variabili di input	-
y	Coordinata lungo l'asse delle ordinate	-
$\boldsymbol{y}$	Vettore generico di variabili di output	-
$\hat{\boldsymbol{y}}$	Valore predetto	-
z	Somma pesata in ingresso alle unità di un MLP	-
z	Coordinata lungo l'asse delle altezze	-
$\mathcal{L}$	Funzione di costo o loss function	-

Introduzione

La costruzione di modelli dinamici per la descrizione di sistemi reattivi nel campo dell'ingegneria chimica viene tradizionalmente effettuata a partire da studi della micro-cinetica, dove si richiede un'approfondita conoscenza dei meccanismi di reazione alla base del processo analizzato. Tuttavia, è molto complicato acquisire tali informazioni nel caso di sistemi complessi e descritti da numerose equazioni differenziali tra loro accoppiate: alcuni processi, quali la combustione di idrocarburi [1], la pirolisi di biomassa [2] o l'idrogenazione della CO_2 per produrre metano [3], coinvolgono un elevato numero di reazioni chimiche che, a loro volta, producono e consumano numerose specie presenti all'interno del sistema. Di conseguenza, la costruzione di un modello cinetico basato su equazioni analitiche ricavate, ad esempio, da espressioni di Langmuir-Hinshelwood-Hougen-Watson (LHHW) richiederebbe un'eccessiva semplificazione del processo, con conseguente perdita di informazioni utili per un'opportuna modellazione.
Per questo motivo, negli ultimi anni si è rapidamente diffusa l'implementazione di modelli *data-driven* [4], al fine di eseguire studi cinetici di processi chimici complessi. Essi hanno il principale vantaggio di poter descrivere sistemi dinamici apprendendo solamente dai dati, generati sinteticamente o sperimentalmente, che vengono loro forniti come *input*, senza il bisogno che si definiscano o conoscano i principali meccanismi di reazione coinvolti.

In questo campo di applicazione hanno riscontrato un particolare successo i modelli proposti da Chen et al. [5], conosciuti con il nome di *Neural Ordinary Differential Equations* (NODEs). Essi sono efficaci approssimatori di sistemi dinamici, poiché finalizzati ad approssimare delle derivate facendo uso di reti neurali:

$$\frac{dx}{dt} = MLP(x, t, \theta)$$

dove θ è un generico set dei parametri del modello. Dall'equazione sopra riportata si può evincere come tali modelli ben si prestino ad approssimare i *reaction rates* coinvolti in un processo descritto da una o più reazioni chimiche.

Uno dei maggiori limiti legati all'implementazione delle Neural ODEs risiede nella necessità di avere a disposizione un'elevata quantità di dati per eseguire il *training*

dei modelli ed ottenere buoni risultati. Quando si hanno a disposizione set di dati raccolti sperimentalmente, come per il caso di modellazione di cinetiche chimiche, molto spesso le dimensioni di questi *datasets* sono limitate.
Per cercare di compensare tale limitazione, Fedorov et al. [3] hanno dimostrato la possibilità di allenare modelli basati su dati sperimentali applicando dei vincoli derivanti da conoscenze cinetiche e termodinamiche del processo, quali la dipendenza della cinetica dalla temperatura secondo la legge di Arrhenius o la stechiometria delle reazioni coinvolte: tali modelli prendono il nome di *Kinetics-Constrained Neural Ordinary Differential Equations* (KCNODEs).
Le KCNODEs non sono gli unici modelli di *deep learning* ad imporre degli *hard contraints* nell'architettura del modello: esistono infatti diverse alternative che funzionano in maniera analoga. Ad esempio, Ji e Deng nel 2021 hanno proposto le *Chemical Reaction Neural Networks* (CRNN) [6], le quali impongono, durante la fase di *training*, che vengano rispettate le leggi fisiche principali coinvolte in una reazione chimica, quali la legge di azione di massa, e la legge di Arrhenius. All'interno del loro lavoro Ji e Deng hanno dimostrato l'applicabilità delle CRNN per predire le cinetiche di reazione presentando una serie di esempi numerici. Inoltre, Ji et al. [2] hanno applicato tali modelli per predire i meccanismi di reazione coinvolti in un processo di pirolisi della biomassa a partire da dati sperimentali raccolti mediante analisi termo-gravimetrica. I risultati ottenuti sono buoni, tuttavia il modello sviluppato risulta essere difficile da interpretare: una delle possibili cause potrebbe essere la presenza di specie chimiche allo stato solido, le quali difficilmente riescono ad essere modellate mediante l'utilizzo di una legge di potenza.
Un'altra tipologia di modelli proposta da Raissi et al. nel 2019 sono le *Physics-Informed Neural Networks* (PINNs), le quali risolvono problemi descritti da equazioni differenziali parziali non lineari che devono rispettare dei vincoli imposti da leggi della fisica [7]. Nel 2022 Gusmão et al. [8] hanno generalizzato tali modelli per molteplici tipi di cinetiche di reazione, proponendo le *Kinetics-Informed Neural Networks* (KINNs): tuttavia, il loro corretto funzionamento richiede che si forniscano informazioni legate al meccanismo di reazione e, in alcuni casi, le espressioni delle equazioni cinetiche. Le KCNODEs, invece, mirano ad evitare la ricerca di equazioni che forniscano un *fitting* dei dati accettabile, sostituendo queste ultime con i modelli di *deep learning* allenati a partire da dati sperimentali [3].

Il processo considerato per questo lavoro di modellazione di cinetiche chimiche implementando le Neural ODEs è la produzione di idrogeno tramite Aqueous Phase Reforming del glicerolo: è un caso studio interessante perché le reazioni coinvolte sono molte, quindi è difficile ipotizzare un modello analitico ed ottimizzarne i molti parametri cinetici contenuti nelle equazioni.

L'APR è attività di ricerca nel dipartimento perché l'utilizzo dell'idrogeno come fonte di energia sostenibile è stato fortemente incentivato negli ultimi anni: esso,

infatti, se utilizzato come combustibile all'interno delle *fuel cells* per produrre energia, comporterebbe delle emissioni di inquinanti molto basse, se non addirittura nulle [9]. Di conseguenza, potrebbe costituire una delle possibili alternative all'utilizzo di combustibili fossili come fonti energetiche. Tuttavia, i principali processi di produzione dell'idrogeno, come lo steam reforming, l'ossidazione parziale di frazioni pesanti del petrolio o la gassificazione del carbone, fanno uso di fonti fossili come materia prima e richiedono un consumo molto elevato di energia [10], con conseguente emissione di elevate quantità di CO_2 equivalente. È necessario, quindi, ricercare dei modi per produrre idrogeno utilizzando dei percorsi più sostenibili e renderlo a tutti gli effetti un possibile sostituto *green* dei combustibili tradizionali. Per questo motivo, il processo di APR del glicerolo è attualmente oggetto di grande interesse nel campo della ricerca scientifica, in quanto risulta essere una valida alternativa alle classiche tecniche per la produzione di idrogeno sopra citate.

Uno dei principali vantaggi dell'APR rispetto allo steam reforming è che non si ha necessità di portare i reagenti allo stato gassoso [10]: di conseguenza, le temperature richieste hanno un valore contenuto, che tendenzialmente non supera i 250°C [11], a fronte dei valori di temperatura da mantenere nello steam reforming del metano, che di norma si aggirano attorno agli 800-1000°C [12]. Il risparmio energetico in termini di calore necessario a mantenere in temperatura il sistema è dunque significativo.

Un secondo aspetto interessante legato al reforming di glicerolo è proprio la scelta del *feedstock* utilizzato: negli ultimi anni, infatti, la disponibilità di tale sostanza è cresciuta molto rapidamente, in quanto il glicerolo è il principale sottoprodotto in termini quantitativi del processo di produzione del biodiesel, il quale viene ottenuto mediante reazione di trans-esterificazione di acidi grassi. Si è stimato che il rapporto volumetrico tra biodiesel e glicerolo ottenuto sia di 10:1 [13]: trovare quindi delle possibili soluzioni che permettano di valorizzare uno scarto risulta essere ulteriormente vantaggioso, in quanto si sta utilizzando un sottoprodotto per ottenere una sostanza altamente richiesta come l'idrogeno. In Figura 1 è riportato un *flow diagram* che schematizza il processo, a partire dal glicerolo prodotto dalla sintesi del biodiesel fino ad arrivare ai prodotti desiderati.

Per le ragioni sopra citate, l'APR del glicerolo è attualmente al centro di diverse ricerche, sia in campo sperimentale sia nel campo della modellazione reattoristica: l'implementazione delle NODEs per riprodurre le reazioni coinvolte in questo processo è, infatti, finalizzata ad ottenere un modello di cinetica *data-driven*, che fornisca approssimazioni soddisfacenti da poter accoppiare con simulazioni di *Computational Fluid Dynamics* (CFD).

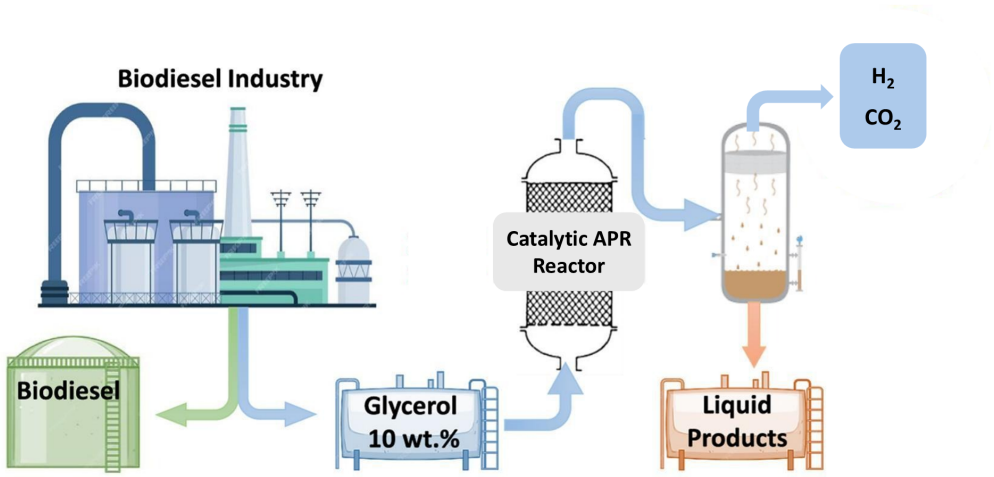


Figura 1: *Flow diagram* del processo di APR del glicerolo prodotto dalla sintesi di biodiesel (immagine adattata da [14]).

La modellazione di sistemi reattivi deve tenere conto delle numerose interazioni chimico-fisiche che si sviluppano all'interno del sistema [4], comprese le reazioni chimiche che possono svilupparsi. Il successo ottenuto nell'accoppiamento di una simulazione CFD con un modello di cinetica *data-driven* ha permesso di simulare il processo di APR del glicerolo tenendo conto di più reazioni parallele, difficilmente descrivibili realisticamente mediante l'utilizzo di cinetiche analitiche.

Di seguito si riporta una panoramica del contenuto dei capitoli presenti nella tesi:

- Nel primo capitolo si presentano gli aspetti teorici inerenti al *Machine Learning*, con particolare riferimento ai tipi di reti neurali maggiormente diffusi e alle NODEs implementate nello studio;

- Nel capitolo 2 vengono riportati i risultati ottenuti nella prima parte del lavoro svolto, finalizzata all'ottenimento di modelli ottimizzati in grado di predire l'andamento delle concentrazioni nel tempo delle specie chimiche coinvolte nel processo considerato. In particolare, si confrontano i risultati acquisiti considerando un caso di APR semplificato, dove sono state considerate due possibili reazioni, ed un secondo scenario più articolato e realistico, in cui i meccanismi di reazione possibili sono quattro in totale;

- Il terzo capitolo è dedicato alla presentazione della teoria implementata all'interno delle simulazioni CFD effettuate, con descrizione delle equazioni risolte all'interno dei *solver* utilizzati;

- Nel quarto capitolo vengono commentati i risultati dell'accoppiamento effettuato tra CFD e modelli di *deep learning*;
- Nel quinto ed ultimo capitolo si riportano le conclusioni del progetto sviluppato, con eventuali prospettive di miglioramento per lavori futuri.

Il lavoro svolto in questo progetto di tesi può essere rappresentato attraverso il *workflow* riportato nella figura seguente:

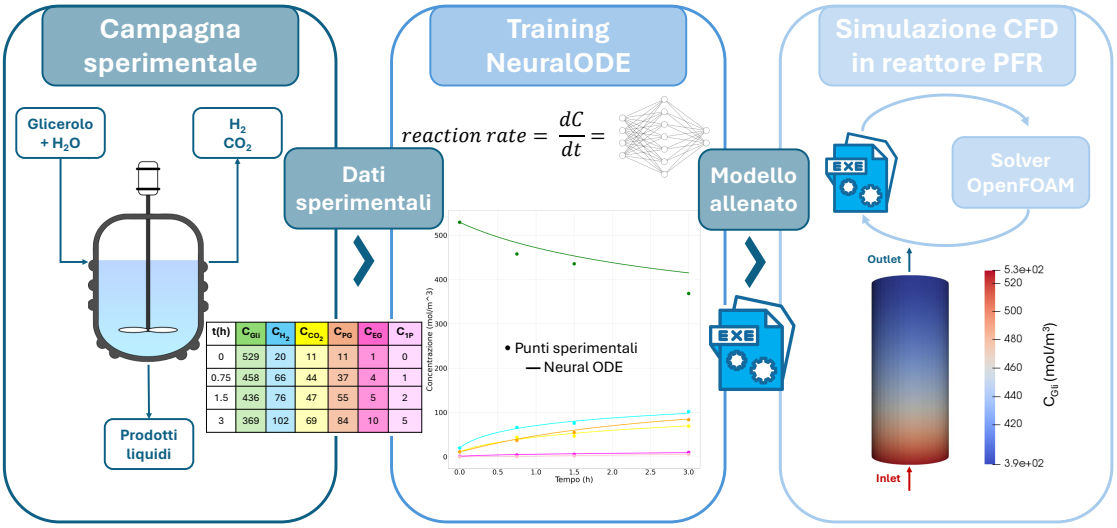


Figura 2: *Workflow* del progetto svolto.

1 Aspetti teorici: NODEs

In questo capitolo vengono descritte le teorie alla base della prima parte del lavoro svolto, legato all'utilizzo delle KCNODEs per la modellazione delle cinetiche di reazione coinvolte nel processo di APR. In particolare, si riporta una panoramica relativa al campo del *Machine Learning* (ML) e dei modelli di regressione (sezioni 1.1 e 1.2) e, successivamente, vengono descritti i principali modelli di reti neurali (sezioni 1.3, 1.4, 1.5), concentrandosi principalmente su quelli utilizzati per la generazione dei modelli predittivi implementati nello studio.

1.1 Machine Learning

Con il termine *Machine Learning* si fa riferimento oggi a numerose tecnologie utilizzate al fine di sviluppare algoritmi in grado di apprendere in autonomia a partire da dataset disponibili. A differenza delle tradizionali tecniche di programmazione, infatti, tali algoritmi hanno la capacità di imparare a svolgere i propri compiti a partire dai dati che vengono loro forniti come *input*, senza la necessità che si forniscano istruzioni e comandi su come risolvere tali problemi.
Il termine Machine Learning venne coniato da Arthur Samuel nel 1959, il quale affermò che: *"Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed"* [15].
I campi di applicazione di tali tecnologie sono numerosi e molto diversi tra loro: esse sono alla base dei sistemi di filtraggio delle e-mail ricevute, del funzionamento della tecnologia di riconoscimento facciale, dei suggerimenti di amicizia che appaiono sui *social network*. Negli ultimi anni, inoltre, si è avuta una rapida crescita nell'implementazione di tecniche di ML in campo medico diagnostico [16].

Il prerequisito fondamentale per poter implementare tecniche di ML è la disponibilità di un set di dati dal quale i modelli sviluppati possono estrapolare le informazioni utili per poter effettuare predizioni affidabili e compiere azioni articolate, senza bisogno di intervento umano [17]. Spesso tali dataset contengono quantità di dati enormi e organizzati in maniera poco funzionale; di conseguenza, essi necessitano di essere sottoposti ad una fase preliminare di *pre-processing*. Le

operazioni che possono essere eseguite sono molteplici: è molto importante, ad esempio, identificare la presenza di possibili *outliers* che possono compromettere la bontà dell'addestramento del modello, così come risulta fondamentale andare ad eseguire delle normalizzazioni o standardizzazioni per riscalare i valori dei dati in *input* in intervalli che conferiscono maggiore stabilità alla creazione dell'algoritmo. Una volta ottenuto un dataset opportuno, esso viene tipicamente suddiviso in due parti principali: la prima sarà destinata al *training*, costituita da una fase di apprendimento in cui il modello cerca di individuare possibili *patterns* utili ad eseguire le funzioni richieste in modo affidabile ed efficace. La seconda parte del dataset, invece, servirà successivamente al *training* per testare se effettivamente il modello sia in grado di effettuare predizioni valide, ricevendo in *input* dati che non appartenevano al dataset dedicato alla fase di apprendimento. Nel caso in cui il modello addestrato fornisca risultati soddisfacenti, esso potrà essere implementato in sistemi diversi, con lo scopo di effettuare predizioni partendo da dati non contenuti nel dataset utilizzato per l'apprendimento.

Le tecniche di ML che vengono oggi implementate sono numerose e si differenziano principalmente per l'approccio che viene utilizzato durante il *training*. Di seguito si riportano le classi principali:

- *Supervised learning*: durante la fase di *training* il modello riceve un set di dati etichettati, in cui per ogni input è noto il corrispondente output [17]. L'obiettivo di tale apprendimento è stimare la funzione che lega gli input agli output, per poter effettuare predizioni corrette su nuovi dati non ancora visti. Quando gli output forniti sono costituiti da valori discreti si parla di classificazione, mentre se hanno un andamento continuo ci si trova davanti ad un caso di regressione [18]. Un esempio tipico di questa tipologia di apprendimento è il problema della *spam detection* nelle e-mail, che può essere formulato come una classificazione binaria (spam/non spam);

- *Unsupervised learning*: in questo secondo caso il dataset fornito al modello per l'apprendimento contiene dati non etichettati: non esiste, di conseguenza, una relazione esplicita tra input ed output. Sarà il modello stesso a dover individuare possibili *patterns* o relazioni latenti, senza la presenza di una supervisione esterna [19]. Una delle tecniche di *Unsupervised learning* più utilizzate è il *clustering*, implementato quando si ha la necessità di raggruppare gli elementi di un dataset -ad esempio i clienti di una catena di supermercati- in base alle loro affinità;

- *Semi-supervised learning*: tale paradigma di apprendimento mira all'utilizzo di entrambe le tecniche appena presentate. Esso utilizza sia dati etichettati sia non etichettati. Tipicamente il numero di elementi con output noto è inferiore rispetto a quello dove le correlazioni non sono esplicitate. L'idea

alla base di questo tipo di apprendimento è quella di sfruttare le informazioni implicitamente contenute nei dati non etichettati per rinforzare i *patterns* già esistenti nel set di dati etichettati [20];

- *Reinforcement learning*: è una tecnica di apprendimento nella quale il modello ottimizza le proprie azioni e decisioni al fine di massimizzare le ricompense ottenute e minimizzare le penalità. A differenza delle altre tecniche viste, nel *Reinforcement learning* il *training* ed il *test* del modello avvengono simultaneamente [18].

1.2 Modelli di regressione

In *machine learning* con il termine regressione si fa riferimento a tecniche di *Supervised learning* che mirano a predire output con andamento continuo (y) a partire da una o più variabili indipendenti (x_i). L'obiettivo di tali metodi è quello di individuare una funzione $f(x_i)$ che meglio approssimi la dipendenza dell'output dalle variabili fornite in input [21]:

$$y = f(x_1, x_2, ..., x_i) + \epsilon \tag{1.1}$$

dove:

- y è la variabile dipendente o *target* che si vuole predire;
- x_i sono le variabili indipendenti o *features* che influenzano l'andamento della predizione;
- ϵ è l'errore rispetto al valore reale y.

In base alla relazione esistente tra queste variabili si possono suddividere i modelli di regressione in due categorie principali:

- regressione lineare: la dipendenza tra la variabile target e le *features* è di tipo lineare
- regressione non lineare: la relazione tra y e x_i non ha carattere lineare, ma può essere descritta da funzioni polinomiali o di altro carattere.

Per valutare la bontà dell'addestramento durante il *training* di un modello di regressione, tipicamente si osserva l'andamento della *loss function*, la cui forma e ordine di grandezza variano in base al tipo di funzione utilizzata. Di seguito si riportano le formule delle *loss* maggiormente utilizzate.

Mean Squared Error (MSE): esso è costituito dalla media eseguita sui quadrati delle differenze tra valore predetto e quello reale.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{1.2}$$

dove:

MSE: *Mean Squared Error*;

N: numero di predizioni;

y_i: valori reali;

$\hat{y}_i$: valori predetti.

Root Mean Squared Error (RMSE): il RMSE non è altro che la radice quadrata della media dei quadrati degli errori. In altre parole, esso costituisce la deviazione standard degli errori.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{1.3}$$

dove gli elementi contenuti nell'equazione sono gli stessi visti per il MSE.

Mean Absolute Error (MAE): esso fornisce il valore mediato della differenza in valore assoluto tra i valori predetti e i valori reali. La formula per il calcolo del MAE è la seguente:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{1.4}$$

dove gli elementi contenuti nell'equazione sono gli stessi visti per il MSE ed il RMSE. Nel progetto di tesi svolto, la tipologia di *loss function* implementata per valutare la bontà di addestramento delle reti è stata il *Mean Squared Error*.

L'obiettivo, durante il *training* di un modello, è quello di minimizzare il valore della *loss*, in quanto essa indica la distanza tra la curva che meglio approssima i dati e i valori reali: più è prossima allo 0, migliore sarà il *fitting* dei dati [22].
Tuttavia, quando il modello si adatta troppo ai dati utilizzati per l'addestramento, memorizzando anche un possibile rumore dovuto alla raccolta di dati sperimentali, si presenta il fenomeno dell'*overfitting*: la *loss* assumerà valori prossimi allo 0, ma il modello non avrà alcuna capacità di generalizzare ciò che ha appreso; di conseguenza, se si applicasse tale modello per effettuare predizioni su dati nuovi -non visti durante il *training*- i risultati potrebbero essere non accettabili.
In Figura 1.1 viene riportato un esempio tipico di *overfitting* ottenuto durante l'addestramento di un modello di regressione lineare. Come si può osservare, la

curva auspicabile è costituita dalla retta tratteggiata, che ben approssima i dati senza essere influenzata dal *noise* da cui sono affetti. La curva continua, invece, ha memorizzato in maniera eccessiva l'andamento dei punti, senza acquisire capacità di generalizzazione.

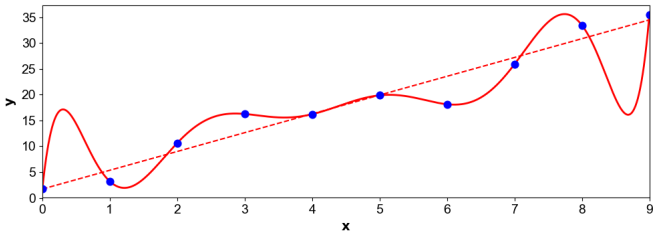


Figura 1.1: Esempio grafico del problema di *overfitting* dei dati.

Nei prossimi paragrafi si riporta una presentazione sintetica di alcuni dei principali metodi di regressione implementati in *Machine Learning*.

1.2.1 Regressione lineare

Nella regressione lineare, l'addestramento è finalizzato alla modellazione di una dipendenza lineare tra la variabile dipendente (y) e una o più variabili indipendenti (x_i). Quando si ha la dipendenza di y da una sola x si parla di regressione lineare semplice; se, invece, la variabile dipendente presenta all'interno del modello dipendenze da più x_i si parla di regressione multi-lineare. La forma generale della funzione che si ottiene da tale metodo viene riportata di seguito:

$$\hat{y} = \beta_0 + \sum_{i=1}^{n} \beta_i x_i \tag{1.5}$$

dove β_0 rappresenta il valore dell'intercetta con l'asse y del piano cartesiano, x_i sono le variabili indipendenti da cui y dipende linearmente e β_i sono i coefficienti di regressione, i quali rappresentano la variazione di y al variare di x_i [22].

1.2.2 Regressione lineare Ridge

La regressione *Ridge* è una variante del metodo lineare, mirata a limitare il problema dell'*overfitting* visto precedentemente. Lo scopo di tale metodo è quello di minimizzare la funzione *Costo* (equazione 1.6), che sfavorisce i coefficienti β_i con valori assoluti maggiori. La formula è la seguente:

$$Costo = ||y - \hat{y}||_2^2 + \alpha||\beta||_2^2 \tag{1.6}$$

dove α rappresenta il parametro di regolarizzazione, che tipicamente assume valore maggiore di zero: più esso è elevato, maggiore sarà la penalizzazione imposta, mentre diminuirà la sensibilità verso i dati di *training*.

1.2.3 Regressione lineare LASSO

Analogamente al metodo *Ridge*, la regressione LASSO (*Least Absolute Shrinkage and Selection Operator*) è finalizzata ad evitare eventuali problemi di *overfitting*. In questo caso, però, la funzione costo ha la forma sotto riportata:

$$Costo = \frac{1}{2N}||y - \hat{y}||_2^2 + \alpha||\beta||_1 \tag{1.7}$$

La carattersitica interessante di questo metodo, se confrontato al *Ridge*, risiede nella tendenza a spingere alcuni dei coefficienti a zero, diminuendo il numero di *features* che costituiscono il modello e rendendolo più semplice da interpretare.

1.2.4 Regressione polinomiale

L'ultimo metodo di regressione presentato, prima di passare alle reti neurali (sezione 1.3), è quello polinomiale. Esso può essere considerato come un caso particolare di regressione multi-lineare, dove le variabili indipendenti non sono altro che un'unica x elevata ad esponenti anche maggiori di uno. La funzione che esprime opportunamente il modello implementato è la seguente:

$$y = \beta_0 + \sum_{k=1}^{n} \beta_k x^k \tag{1.8}$$

dove n rappresenta il grado del polinomio. Tale modello di regressione viene utilizzato quando la relazione tra la variabile dipendente e quella indipendente non è lineare [21].

1.3 Multi-Layer Perceptrons (MLP)

I *Multi-Layer Perceptrons (MLPs)*, anche noti con il nome di *feedforward neural networks*, sono dei modelli di *deep learning* molto efficienti ed oggi ampiamente utilizzati nel campo del *machine learning*. Il loro scopo principale è quello di andare ad approssimare una funzione f che lega il vettore di variabili di input $\boldsymbol{x}$ con quello delle variabili di output $\boldsymbol{y}$, apprendendo durante la fase di *training* i valori dei parametri $\boldsymbol{\theta}$.

$$\boldsymbol{y} = MLP(\boldsymbol{x}, \boldsymbol{\theta}) \tag{1.9}$$

L'equazione 1.9 descrive in maniera semplice il concetto appena presentato: dando in input alla rete un dataset contenente i dati di *training* ($\boldsymbol{x}$), essa calibra i propri parametri ($\boldsymbol{\theta}$) affinché l'output fornito ($\boldsymbol{y}$) sia il più vicino possibile a quello reale. Il termine *feedforward* mette in evidenza come il passaggio di informazioni all'interno del modello abbia un'unica direzione: partendo dall'ingresso dei dati, esse vengono elaborate attraverso una serie di computazioni intermedie, fino a generare l'output della rete [23].
Il termine *neural*, invece, indica come tali modelli abbiano un principio di funzionamento analogo a quello delle reti di neuroni biologici di un cervello umano. Questi ultimi comunicano tra loro attraverso una serie di impulsi elettrici, trasmessi grazie alla presenza di connessioni che prendono il nome di sinapsi. Ogni neurone riceve un elevato numero di stimoli provenienti da altri neuroni circostanti, il cui effetto totale, se supera un determinato *threshold*, scatenerà una conseguente risposta che verrà a sua volta trasmessa all'esterno [24]. Analogamente, i MLPs sono costituiti da un numero più o meno elevato di nodi o unità, organizzati in *layers*, il cui numero può variare e stabilisce la profondità del modello (da qui il termine *deep learning*) [23]. Tali *layers* possono essere suddivisi in tre categorie principali: l'*input layer*, il quale riceve i dati in ingresso e li trasmette al modello senza processarli, l'*output layer*, che fornisce i risultati predetti, e gli *hidden layers*, che costituiscono la parte "nascosta" della rete, in quanto all'interno dell'equazione 1.9 non si hanno elementi che descrivano esplicitamente il comportamento che tali unità devono assumere.

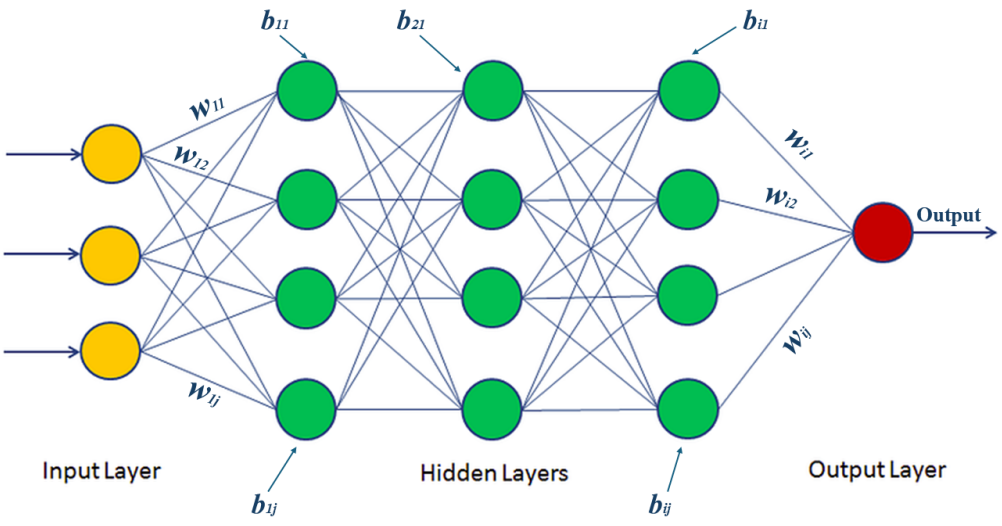


Figura 1.2: Struttura di un *Multi-Layer Perceptron* (MLP).

In Figura 1.2 viene riportato uno schema della struttura di un MLP: si possono

notare le diverse tipologie di *layers* sopra descritte e, inoltre, si evidenziano due termini non ancora introdotti:

- $\boldsymbol{w_{ij}}$ sono i pesi (*weights*) della rete, ognuno dei quali moltiplica l'output x_{ij} ottenuto dal j-esimo nodo dell'i-esimo *layer* prima che questo venga dato in input ai nodi del *layer* successivo;
- $\boldsymbol{b_{ij}}$ sono i bias della rete, ovvero parametri che permettono di avere un *threshold* di attivazione del nodo diverso da zero; in altre parole, i bias consentono ai nodi di attivarsi anche quando tutti i pesi in input a tale unità sono pari a zero.

I valori uscenti dai nodi di un generico *layer*, moltiplicati per i rispettivi pesi, verranno successivamente sommati, insieme al bias, per generare quello che viene chiamato input netto ad un nodo del *layer* successivo [25]:

$$z = \sum_{j=1}^{n} w_{ij} x_j + b \tag{1.10}$$

dove n è il numero di nodi contenuti all'interno dell'i-esimo *layer*. L'input netto z è il valore che stabilisce se il nodo ricevente possa o meno attivarsi; tale attivazione, inoltre, dipenderà dal tipo di *activation function* utilizzata.

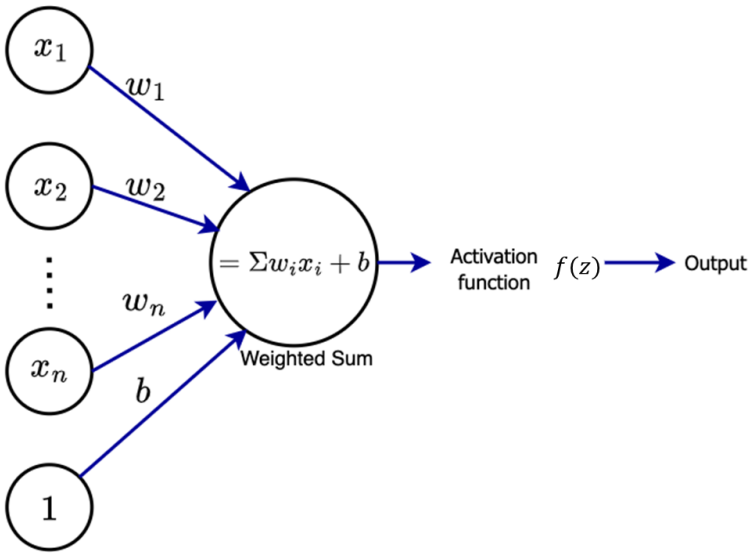


Figura 1.3: Schema di funzionamento alla base di un'unità.

Nella Figura 1.3 si riporta una schematizzazione grafica del flusso computazionale alla base del funzionamento di un nodo: i valori in output dalle unità precedenti

x_j, insieme ai pesi w_{ij} e al bias riferito al nodo di interesse, vengono inseriti all'interno dell'equazione 1.10. L'input netto z verrà successivamente utilizzato come argomento della funzione di attivazione $f(z)$, il cui valore risultante costituirà l'output del nodo.

Nei paragrafi successivi si riporta una breve descrizione delle principali tipologie di funzioni di attivazione, con particolare attenzione a quelle utilizzate per la costruzione dei modelli implementati nel lavoro svolto.

1.3.1 Binary step function

La *binary step function* è la più semplice delle funzioni di attivazione. Tipicamente essa viene implementata quando si fa uso di MLPs per effettuare una classificazione binaria, mentre difficilmente viene utilizzata per le classificazioni multi-classe. A livello matematico essa può essere espressa come:

$$\begin{cases} f(z) = 1 & \text{se } z \geq 0 \\ f(z) = 0 & \text{se } z < 0 \end{cases} \tag{1.11}$$

Nonostante la sua semplicità, l'equazione 1.11 rappresenta pienamente il principio di funzionamento di un nodo, introdotto da Rosenblatt nel 1958 [26], in quanto si ha una variazione netta, assimilabile ad un'attivazione del *perceptron*, dell'output della funzione una volta raggiunto un valore limite (*threshold*) di z. L'andamento di $f(z)$ viene riportato nella Figura 1.4.

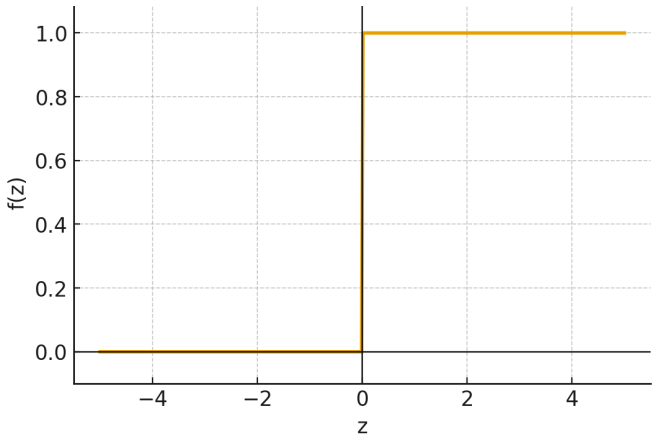


Figura 1.4: *Binary step function*: rappresentazione grafica dell'andamento.

1.3.2 Logistic e tanh functions

Le *logistic e tanh functions* sono due funzioni differenziabili ampiamente utilizzate: esse assumono una tipica forma ad "S" in prossimità dell'origine. Si fa riferimento,

infatti, a tali funzioni di attivazione con il termine *sigmoid functions* [27]. Uno dei motivi principali del loro diffuso utilizzo è legato al fatto che esse approssimano il comportamento della *binary step function* con il vantaggio di essere differenziabili: si vedrà in seguito come tale caratteristica sia molto importante per la fase di apprendimento e ottimizzazione dei parametri della rete. Per quanto riguarda la *logistic function*, l'equazione è la seguente:

$$f(z) = \frac{1}{1 + e^{-z}} \tag{1.12}$$

mentre per la *tanh function* si ha:

$$f(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{1.13}$$

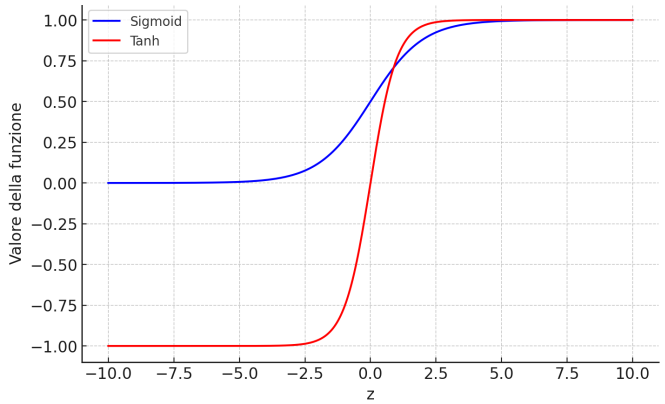


Figura 1.5: Confronto degli andamenti di *logistic function* e *tanh function*.

Dalla Figura 1.5 si evince come la principale differenza consiste nel range di output che si possono ottenere: se per la *logistic function* l'intervallo coperto è [0,1] (come quello visto nella *binary step function*), per la *tanh function* $f(z)$ può assumere valori compresi tra [-1,1].

1.3.3 ReLU e Softplus

La funzione di attivazione ReLU (*Rectified Linear Unit*) [27] ha come principale vantaggio l'economicità in termini computazionali, in quanto costituita da una semplice funzione che, per $z < 0$, assume un comportamento costante e pari a zero, mentre per z positive ha un andamento lineare con coefficiente angolare pari a 1. In termini matematici può essere scritta come:

$$f(z) = max(0, z) \tag{1.14}$$

Tuttavia, la funzione ReLU presenta un punto di non derivabilità in corrispondenza dell'origine; una possibile alternativa a tale *activation function* differenziabile su tutto il dominio è la funzione *softplus*, la cui equazione è la seguente:

$$f(z) = log(1 + e^z) \tag{1.15}$$

In Figura 1.6 vengono messi a confronto gli andamenti delle funzioni sopra descritte, ponendo particolare attenzione al comportamento assunto nell'intorno di $z = 0$: si può notare, infatti, come la *softplus* non sia altro che una versione più *"smooth"* della ReLU.

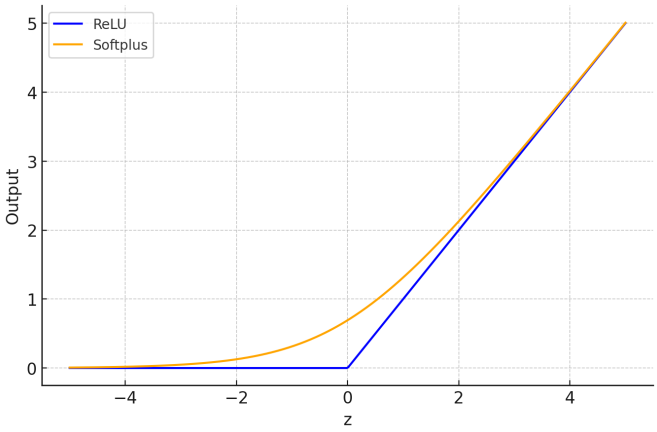


Figura 1.6: Confronto degli andamenti di *ReLU* e *Softplus*.

1.3.4 SiLU

La SiLU (*Sigmoid Linear Unit*), anche nota come *swish function*, è stata proposta come funzione di attivazione nel campo del *reinforcement learning* da Elfwing et al. nel 2017 [28]. Essa si ottiene andando a moltiplicare la funzione in input z, ottenuta con l'equazione 1.10, per la funzione logistica vista nella sezione 1.3.2:

$$f(z) = \frac{z}{1 + e^{-z}} \tag{1.16}$$

Per valori di z molto elevati, l'esponenziale al denominatore tende a 0; di conseguenza, l'equazione 1.16 assumerà valori prossimi a z, comportandosi in maniera analoga alla ReLU. Per valori della variabile in input negativi e molto grandi in valore assoluto, invece, la funzione tende nel complesso a zero: anche in questo caso, il comportamento assunto è paragonabile alla funzione ReLU. Tuttavia, la principale differenza tra le due *activation functions* risiede nel fatto che la SiLU non ha un comportamento monotono crescente [28]. Infatti, dalla Figura 1.7 si può osservare come essa presenti un minimo locale per $z \approx -1.28$ pari a circa -0.28.

Un'ulteriore caratteristica importante della SiLU consiste nella proprietà di autostabilizzazione: la presenza del minimo locale con derivata nulla, infatti, funge da inibitore nei confronti dell'apprendimento di valori dei pesi troppo grandi, che causerebbero instabilità numerica durante la fase di *training* dei parametri.

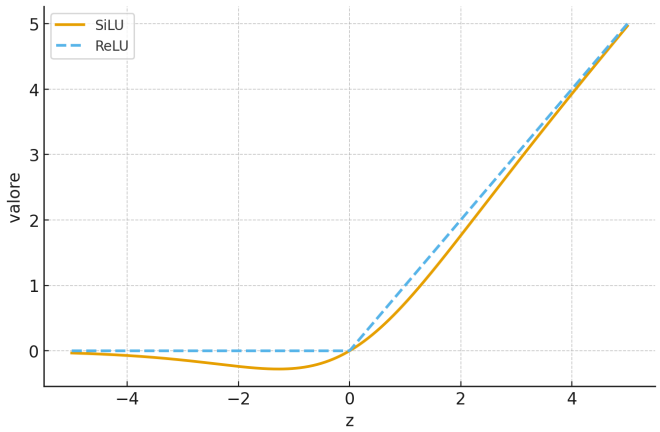


Figura 1.7: Confronto degli andamenti di *ReLU* e *SiLU*.

Grazie alle caratteristiche evidenziate in questo paragrafo, si è scelto di implementare la funzione SiLU come funzione di attivazione di alcuni dei *layers* costituenti il modello generato per l'approssimazione delle cinetiche di reazione coinvolte nel processo di APR.

1.3.5 Funzione esponenziale

L'ultima funzione presentata è quella esponenziale. Essa viene raramente utilizzata come *activation function*, in quanto presenta una velocità di crescita per $z > 0$ molto elevata, con conseguente aumento del valore del gradiente, il quale potrebbe far apprendere pesi dai valori molto elevati e instabili numericamente.
Tuttavia, per la modellazione di cinetiche di reazione eseguita nello studio, è stato necessario che la struttura della rete fosse in grado di predire delle dipendenze di tipo esponenziale dalla temperatura, in particolare quelle imposte dalla legge di Arrhenius (equazione 1.17) e dalla costante di Henry (equazione 1.18). Per questo motivo, i *layers* di nodi implementati per approssimare tali dipendenze sono caratterizzati da una funzione di attivazione di tipo esponenziale.
Di seguito si riportano le equazioni relative alla legge di Arrhenius ed alla costante di Henry come funzioni della temperatura.

Legge di Arrhenius:

$$k = Ae^{\frac{-Ea}{RT}} \tag{1.17}$$

dove:

k è la costante cinetica di reazione, con unità di misura che variano al variare dell'ordine della cinetica;

A è il fattore pre-esponenziale, con unità di misura uguali a quelle di k;

Ea è l'energia di attivazione della reazione, espressa in $\frac{J}{mol}$;

R è la costante universale dei gas, pari a 8.314 $\frac{J}{molK}$;

T è la temperatura in K.

Costante di Henry (in forma empirica) [29]:

$$ln(H_s) = \frac{-\Delta H_{sol}}{R}\frac{1}{T} + C \tag{1.18}$$

dove:

H_s è la costante di Henry in $\frac{mol}{m^3 Pa}$;

ΔH_{sol} è l'entalpia di dissoluzione in $\frac{J}{mol}$;

R è la costante universale dei gas, pari a 8.314 $\frac{J}{molK}$;

T è la temperatura in K.

1.4 Back-propagation e gradient descent

Per meglio poter comprendere il funzionamento dei MLPs è importante conoscere quali siano tutti i meccanismi e gli algoritmi che, messi insieme, costituiscono tali modelli. Dopo aver affrontato l'analisi del flusso computazionale alla base della singola unità della rete, si introduce ora l'algoritmo che permette il passaggio di informazioni a partire dal calcolo della funzione di costo, che nelle sezioni precedenti è stata definita come *loss function*, fino ad arrivare al calcolo del gradiente: esso è l'algoritmo di *back-propagation*.
Il nome fa riferimento all'inversione del flusso di informazioni rispetto a quello seguito durante i *forward steps*: se in questi si aveva un verso computazionale diretto dai dati di input all'output della rete, con l'algoritmo di *back-propagation* si ripercorre il modello al contrario, calcolando la funzione di costo a partire dalla $\hat{\boldsymbol{y}}$ ottenuta con l'equazione 1.9 fino ad arrivare al *layer* di input.
Cercando di semplificare, il processo di apprendimento di un MLP può essere suddiviso in quattro fasi principali:

- *Forward pass*: si forniscono i dati in input e, attraverso l'attivazione dei neuroni mediante funzione di attivazione, viene generato un output $\hat{\boldsymbol{y}}$;

- Calcolo della *loss function* per confrontare la predizione con i valori veri $\boldsymbol{y}$;

- *Backward pass*: si valuta come la funzione di costo varia in funzione dei parametri del modello (pesi e bias), attraverso l'implementazione della *chain rule*. L'errore, in questo modo, "risale" dal *layer* di output con conseguente passaggio di informazioni all'interno del modello;

- Ricalibrazione dei parametri: si modificano i valori di pesi e bias nella direzione in cui si ha la massima velocità di diminuzione della *loss*, implementando algoritmi quali la discesa del gradiente.

L'algoritmo di *back-propagation*, il cui principio di funzionamento si basa sulla tecnica di *chain rule* [23], viene implementato nel terzo step.
Data una variabile reale x e le funzioni f e g, entrambe con immagine compresa nel campo dei reali, si consideri $y = g(x)$ e $z = f(g(x)) = f(y)$. L'algoritmo della catena del calcolo differenziale stabilisce che:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{1.19}$$

L'equazione 1.19 è stata scritta per funzioni di tipo scalare, ma la *chain rule* è una tecnica che può essere implementata anche per funzioni che hanno come variabili dei vettori o tensori. In notazione vettoriale l'equazione sopra riportata diventa [23]:

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\mathsf{T}} \nabla_{\boldsymbol{y}} z \tag{1.20}$$

dove $\left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\mathsf{T}}$ è lo Jacobiano della funzione $\boldsymbol{y}$ rispetto alla variabile $\boldsymbol{x}$. Da entrambe le equazioni, scritte in forma scalare e vettoriale, si evince l'importanza di utilizzare delle funzioni di attivazione dei nodi che siano differenziabili nel loro dominio, proprio come evidenziato nella sezione dedicata alle *sigmoid functions* (sezione 1.3.2).
L'applicazione ricorsiva della *chain rule*, in cui si differenzia la funzione di costo rispetto a tutti i parametri del MLP, genera l'algoritmo di *back-propagation*.

L'ultimo step presentato implica la modifica dei parametri del modello; tale calibrazione di pesi e bias viene effettuata utilizzando algoritmi di ottimizzazione che, nel campo del *deep learning*, si basano tipicamente sulla discesa del gradiente (*gradient descent*). L'alterazione di tali parametri, infatti, viene effettuata andando a massimizzare la velocità di diminuzione della funzione di costo. In altre parole,

verranno modificati maggiormente i pesi e i bias che tendono a minimizzare il valore del gradiente.

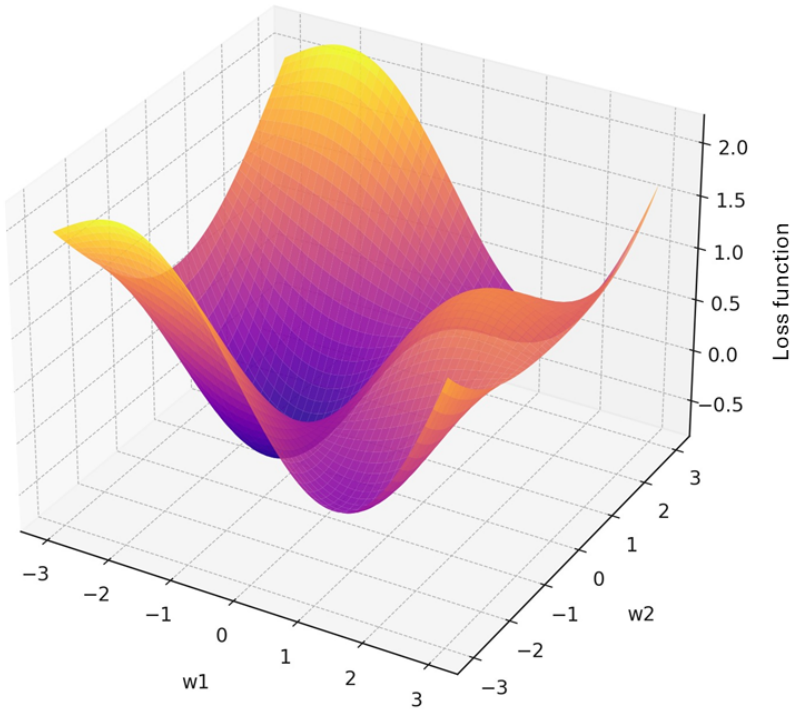


Figura 1.8: Rappresentazione grafica dell'andamento della *loss* in funzione dei parametri w_1 e w_2.

In Figura 1.8 viene riportato l'andamento di una *loss function* al variare dei parametri w_1 e w_2: l'algoritmo della discesa del gradiente cercherà di raggiungere il valore di minimo globale della funzione di costo eseguendo una serie di step orientati nella direzione con valore di gradiente più negativo.
L'ampiezza di tali step computazionali è uno dei numerosi iper-parametri del modello, il cui valore ha da essere ottimizzato al fine di massimizzare la stabilità numerica e la velocità di raggiungimento del minimo [30]. Le dipendenze di tali comportamenti dall'ampiezza dello step -anche chiamata *learning rate*- risultano essere in contrasto tra loro: per elevati valori dell'iper-parametro, si avrà elevata velocità di discesa ma una maggiore instabilità numerica, mentre diminuendo il *learning rate* si renderà l'algoritmo più stabile ma anche più lento.

Nel paragrafo a seguire è riportato il principio di funzionamento delle NODEs, le quali costituiscono le basi modellistiche su cui si sono costruiti i modelli per

l'approssimazione delle cinetiche di reazione di interesse.

1.5 NODEs

Le *Neural Ordinary Differential Equations* (NODEs) sono state introdotte da Chen et al. nel 2018 [5] e rappresentano un'alternativa efficiente ai modelli di rete che approssimano trasformazioni complesse attraverso operazioni discrete, basando il loro funzionamento sugli algoritmi presentati fino ad ora in questo capitolo. Esse fondono il mondo delle reti neurali a quello delle equazioni differenziali ordinarie (ODEs).
Alcuni modelli, come le *recurrent neural networks* o le *residual networks*, effettuano numerose trasformazioni discretizzando il passaggio di informazioni tra un *layer* e l'altro. A livello matematico, uno step da un *hidden layer* generico t a quello successivo può essere scritto come:

$$\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + f(\boldsymbol{h}_t, \theta_t) \tag{1.21}$$

Tale algoritmo può essere trasposto da un campo discreto ad uno continuo andando a ridurre la grandezza dello step, trasformando quindi l'equazione 1.21 nella seguente:

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta), \qquad \boldsymbol{h}(0) = \boldsymbol{h}_0 \tag{1.22}$$

A partire dall'*input layer* $\boldsymbol{h}(0)$, si può ottenere l'output del modello $\boldsymbol{h}(T)$ utilizzando un risolutore di ODEs per risolvere l'equazione 1.22 (ad esempio: Runge-Kutta, metodo di Eulero adattivo, Dopri5) [5].

La criticità principale delle NODEs risiede nel metodo implementato per calcolare il gradiente della funzione di costo rispetto ai parametri θ del modello. L'algoritmo di *back-propagation* analizzato nella sezione 1.4, infatti, richiederebbe che per ogni *time-step* intermedio si memorizzi la funzione $\boldsymbol{h}(t)$ per poter successivamente implementare la *chain rule*; tuttavia, utilizzando un risolutore di equazioni differenziali i passi di ottimizzazione possono essere numerosi e memorizzare i parametri di tutti i *layers* richiederebbe molta memoria. Per questa ragione, tali modelli implementano l'*adjoint sensitivity method*, che calcola il gradiente risolvendo una seconda ODE a ritroso rispetto alla variabile t [5].
In primo luogo, per poter effettuare l'ottimizzazione della *loss* è necessario valutarne la dipendenza in ogni istante da tutti gli *hidden layers*. Tale derivata prende il nome di *adjoint* [5] e ha la seguente forma:

$$\boldsymbol{a}(t) = \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}(t)} \tag{1.23}$$

Andando ad analizzare la variazione rispetto alla variabile t di tale parametro, si ottiene una nuova equazione differenziale, che verrà risolta in verso opposto rispetto all'equazione 1.22:

$$\frac{d\boldsymbol{a}(t)}{dt} = -\boldsymbol{a}(t)^{\intercal}\frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \boldsymbol{h}} \tag{1.24}$$

dove il segno negativo a destra dell'equazione evidenzia l'inversione dell'integrazione. Per poter ottimizzare i parametri, però, risulta ancora necessario calcolare il gradiente della funzione di costo $\mathcal{L}$ rispetto ai parametri stessi, con una terza equazione differenziale:

$$\frac{d\mathcal{L}}{d\theta} = -\int_{t_1}^{t_0} \boldsymbol{a}(t)^{\intercal}\frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \theta} dt \tag{1.25}$$

Computazionalmente, il problema si riassume nella risoluzione di tre ODEs rispetto alla variabile t, le quali possono essere risolte richiamando un solo risolutore per ottenere le tre soluzioni [5]. Il sistema di equazioni è riportato di seguito:

$$\frac{d}{dt}\begin{bmatrix} h(t) \\ a(t) \\ g_\theta(t) \end{bmatrix} = \begin{bmatrix} f\Big(h(t), t, \theta\Big) \\ -\boldsymbol{a}(t)^{\top}\,\partial_h f\Big(h(t), t, \theta\Big) \\ -\boldsymbol{a}(t)^{\top}\,\partial_\theta f\Big(h(t), t, \theta\Big) \end{bmatrix} \tag{1.26}$$

dove la funzione $g_\theta(t)$ racchiude il membro sinistro dell'equazione 1.25.

I principali vantaggi legati all'uso delle *Neural* ODEs sono molteplici:

- Efficienza nell'occupazione di memoria: implementando l'*adjoint sensitivity method* per il calcolo del gradiente della funzione di costo, non è necessario memorizzare le funzioni descriventi le trasformazioni implementate da ogni nodo, riducendo molto l'occupazione di memoria durante la costruzione del modello;
- Adattività della computazione: grazie allo sviluppo di risolutori moderni di ODEs, che sono in grado di adattare la dimensione degli step di integrazione in base alla complessità della dinamica, si ha un aumento dell'efficienza computazionale, che permette un risparmio delle risorse laddove il problema non risulti complesso da risolvere;
- Modellazione naturale di fenomeni continui nel tempo: le NODEs hanno un principio di funzionamento tale da renderle adatte all'approssimazione di fenomeni con comportamento continuo come processi biologici, dinamiche fisiche o sistemi di controllo.

Nel progetto di tesi svolto tali modelli di *deep learning* sono risultati infatti adeguati, in quanto lo scopo dell'implementazione delle NODEs è stato proprio quello di

utilizzarle come una *"black box"* che approssimasse la variazione nel tempo delle concentrazioni di specie chimiche coinvolte nel processo di *Aqueous Phase Reforming*.

2 Risultati: NODEs

Si prosegue ora nel presentare il lavoro svolto per adattare i modelli di *deep learning* descritti nelle sezioni precedenti al caso di interesse.
Per la costruzione e il *training* dei modelli è stata utilizzata la libreria `PyTorch` di Python accoppiata alla libreria `torchDyn`, la quale fornisce un facile accesso ai *continuous-depth models* -ovvero approssimatori neurali di equazioni differenziali- e che risulta essere compatibile con `PyTorch` [31].

2.1 Reazioni chimiche

Durante il processo di APR del glicerolo i possibili cammini di reazione coinvolti sono numerosi. Per questo motivo è necessario l'utilizzo di catalizzatori, al fine di aumentare la selettività del processo verso i meccanismi di reazione che producono idrogeno. Questi ultimi dipenderanno dalla formulazione del catalizzatore che si utilizza nel processo. I più comunemente utilizzati attualmente sono i catalizzatori a base di Pt su supporto di Al_2O_3 [32]. Tuttavia, è inevitabile che si sviluppino reazioni parallele, quali la deidrogenazione e la disidratazione delle specie chimiche presenti nel sistema [32], che entrano in competizione con quella principale di seguito riportata [10]:

$$C_3H_8O_3(l) + 3H_2O(l) \longrightarrow 7H_2(g) + 3CO_2(g) \tag{2.1}$$

Per la costruzione dei modelli si sono tenuti in considerazione due scenari differenti: nel primo si è ipotizzata una sola reazione parallela, con formazione di propilen-glicole come prodotto secondario, generato dalla reazione del glicerolo presente nel sistema con l'idrogeno formatosi dalla reazione 2.1 [32]:

$$C_3H_8O_3(l) + H_2(aq) \longrightarrow C_3H_8O_2(l) + H_2O(l) \tag{2.2}$$

In questo primo caso, dunque, le specie chimiche coinvolte nel processo di APR sono cinque in totale: glicerolo, acqua, CO_2, idrogeno e propilen-glicole. Tuttavia, il processo di reforming del glicerolo avviene in condizioni di elevata diluizione: la

concentrazione percentuale in peso di tale reagente, infatti, difficilmente raggiunge valori superiori al 10 wt% [11]. Si è ipotizzato quindi che la variazione nel tempo della concentrazione di acqua rispetto al quantitativo totale presente nel sistema fosse trascurabile. Di conseguenza, i modelli sono stati utilizzati per stimare l'andamento temporale delle altre quattro concentrazioni.
Nel secondo scenario si sono considerate due reazioni aggiuntive rispetto al caso appena presentato: la prima è data dalla produzione di etilen-glicole a partire dal reforming del glicerolo:

$$C_3H_8O_3(l) + H_2O(l) \longrightarrow C_2H_6O_2(l) + 2H_2(g) + CO_2(g) \tag{2.3}$$

mentre la seconda prevede la formazione di 1-propanolo per idrodeossigenazione del propilen-glicole [32]:

$$C_3H_8O_2(l) + H_2(aq) \longrightarrow C_3H_8O(l) + H_2O(l) \tag{2.4}$$

Le specie chimiche di cui si è modellata la concentrazione nel tempo in questo secondo caso sono quindi le quattro già considerate nel caso semplificato con l'aggiunta di etilen-glicole e 1-propanolo come sotto-prodotti.
La scelta di considerare uno scenario più articolato e che portasse alla formazione di altre specie è stata fatta poiché, come si vedrà in seguito, l'eccessiva semplificazione del sistema impedisce alle NODEs implementate di predire in maniera opportuna i comportamenti di alcune specie, in particolare il propilen-glicole e il glicerolo, a causa della mancanza di informazioni legate allo sviluppo di altre reazioni secondarie che portano al consumo di tali sostanze.

2.2 Architettura del modello

Nella descrizione del funzionamento delle KCNODEs si è sottolineata l'importanza di imporre dei vincoli cinetici e termodinamici all'interno della definizione dell'architettura del modello, affinché questo possa predire in maniera opportuna l'andamento delle concentrazioni nel tempo. Sono stati individuati quattro possibili *contraints* da utilizzare nel caso preso in esame:

- Dipendenza della cinetica dalla temperatura: come riportato da Fedorov et al. nel loro studio [3] la velocità di reazione dipende dalla temperatura assoluta del sistema seguendo la legge di Arrhenius, descritta dall'equazione 1.17. La temperatura viene fornita come input al modello dopo essere stata trasformata nel seguente modo:

  $$T_m = \frac{1}{R}\left(\frac{1}{T} - \frac{1}{T_{ref}}\right) \tag{2.5}$$

  dove T_m è la temperatura di Arrhenius [3] e T_{ref} è stata considerata pari a 573.15 K;

- Stechiometria delle reazioni: al fine di ottenere un consumo dei reagenti ed una produzione dei prodotti corretta si deve tenere conto dei coefficienti stechiometrici presenti nelle equazioni riportate (sezione 2.1). Per questo motivo, all'interno del `forward` del modello vengono moltiplicate le *reaction rates* in output dalla rete per la matrice dei coefficienti:

$$\boldsymbol{r}_c = \boldsymbol{M}_\nu \boldsymbol{r}_R \tag{2.6}$$

  $\boldsymbol{r}_c$ rappresenta il vettore contenente le velocità di variazione delle concentrazioni delle specie chimiche e $\boldsymbol{r}_R$ è il vettore delle velocità di reazione.
  La matrice dei coefficienti stechiometrici $\boldsymbol{M}_\nu$ per il caso con due sole reazioni è la seguente:

$$\boldsymbol{M}_\nu = \begin{pmatrix} -1 & -1 \\ 7 & -1 \\ 3 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.7}$$

  Ogni riga corrisponde ai coefficienti stechiometrici riferiti ad una specie chimica (nell'ordine: glicerolo, idrogeno, CO_2 e propilen-glicole), mentre le colonne rappresentano i coefficienti di una reazione. Per il caso a sei specie, invece, si avrà:

$$\boldsymbol{M}_\nu = \begin{pmatrix} -1 & -1 & -1 & 0 \\ 7 & -1 & 2 & -1 \\ 3 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.8}$$

  dove le due righe aggiunte fanno riferimento ai coefficienti di etilen-glicole ed 1-propanolo;

- *Reaction rates* pari a zero in caso di assenza dei reagenti: per poter imporre tale vincolo il vettore contenente le velocità di reazione predette dalla rete viene semplicemente moltiplicato per le concentrazioni delle specie reagenti. Ad esempio, nel caso della reazione principale (reazione 2.1)si avrà:

$$r_1 = C_{glicerolo}\hat{r}_1 \tag{2.9}$$

  Ricordando l'ipotesi di presenza di acqua in quantità molto elevata e sovra-stechiometrica rispetto al glicerolo, la velocità di reazione del reforming con produzione di idrogeno sarà nulla nel caso in cui venga esaurito il reagente limitante ($C_{glicerolo} = 0$);

- Dipendenza della costante di Henry dalla temperatura: analogamente al vincolo imposto con la legge di Arrhenius, la costante di Henry dipende

esponenzialmente dalla temperatura del sistema. Tale costante è stata considerata per l'idrogeno, la cui concentrazione in acqua è proporzionale alla sua concentrazione in fase gassosa secondo la legge di Henry:

$$C_{H_2,liq} = H'_s C_{H_2,gas} \tag{2.10}$$

dove $H'_s = RT\,H_s$. La dipendenza della costante H_s dalla temperatura è descritta dall'equazione 1.18 che può essere riformulata come [29]:

$$H_s(T) = H_s(T_{ref})\,\exp\left[-\frac{\Delta H_{\text{sol}}}{R}\left(\frac{1}{T}-\frac{1}{T_{ref}}\right)\right] \tag{2.11}$$

Anche in questo caso la temperatura riportata in forma di Arrhenius (equazione 2.5) viene fornita in input alla rete che modella tale dipendenza.

Per comprendere meglio l'architettura del modello sviluppato, viene riportata in Figura 2.1 una rappresentazione grafica della struttura. Essa fa riferimento allo scenario semplificato, dove si considerano solo quattro specie; tuttavia, l'architettura implementata per il caso a sei specie è analoga, con le sole differenze di avere sei concentrazioni in input alla rete invece di quattro e che gli *hidden layers* sono costituiti da 64 unità.

La prima rete raffigurata è quella implementata per la modellazione della dipendenza della costante di Henry dalla temperatura: quest'ultima viene trasformata in forma di temperatura di Arrhenius (equazione 2.5) e fornita all'*input layer*, che passa tale informazione all'unico *hidden layer* di questa rete, con funzione di attivazione SiLU (sezione 1.3.4) e numero di nodi pari a 32. Il *layer* di output riceve successivamente i risultati delle computazioni precedenti e, con una *activation function* di tipo esponenziale, simula il comportamento descritto dall'equazione 2.11. Avendo a che fare con delle concentrazioni in fase gassosa e non delle pressioni parziali, è necessario moltiplicare l'output del modello per il termine RT. Per garantire la stabilità numerica, tale temperatura è stata normalizzata effettuando una divisione per 500. Infine, per poter modellare la legge di Henry (equazione 2.10), si è ancora moltiplicato per la concentrazione normalizzata di idrogeno in fase gassosa.

Passando alla seconda rete implementata, essa riceve in input le concentrazioni delle specie coinvolte nel processo normalizzate dividendo per 500. La struttura interna del modello è anch'essa composta da un *hidden layer* di 32 unità con funzione di attivazione SiLU, i cui risultati vengono trasmessi all'*output layer*, con tanti nodi quante sono le reazioni coinvolte. Le approssimazioni ottenute dalla seconda rete vengono moltiplicate con quelle uscenti dall'ultimo modello riportato in figura, che approssima la dipendenza delle cinetiche di reazione dalla temperatura di Arrhenius. Tale prodotto fornisce come risultato le *reactione rates*, le quali, stando alle equazioni 2.9 e 2.6, hanno da essere moltiplicate ancora per

le concentrazioni dei reagenti coinvolti (al fine di imporre il vincolo di $rate = 0$ in assenza specie che reagiscono) e per la matrice dei coefficienti stechiometrici, al fine di ottenere il consumo o la produzione nel tempo di ogni specie considerata. Il termine aggiuntivo posto uguale a zero corrisponde alla derivata della temperatura, che è stata ipotizzata costante nel tempo.

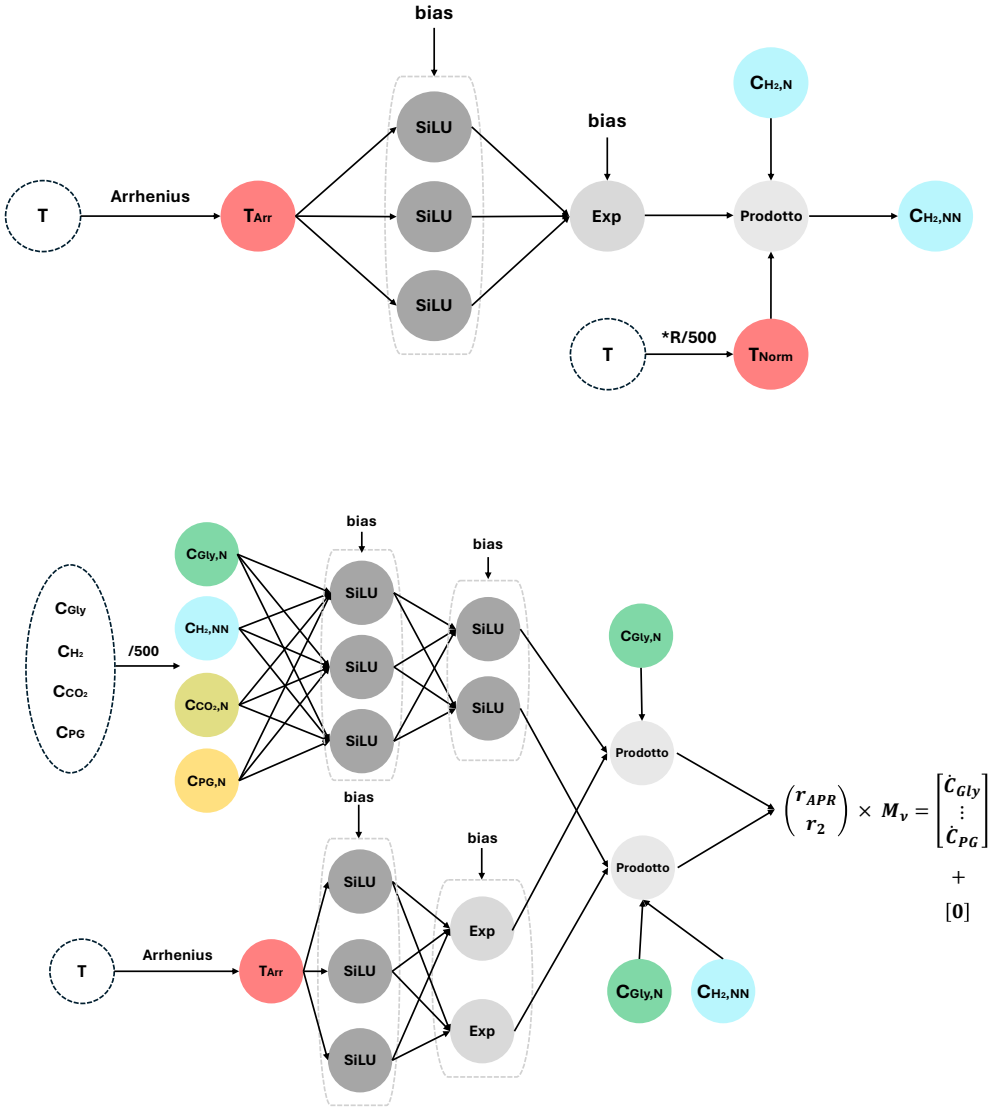


Figura 2.1: Rappresentazione grafica dell'architettura del modello implementato.

Gli script utilizzati per la costruzione dei modelli con quattro e sei specie sono consultabili nell'Appendice.

2.3 Dati sperimentali

Per il *training* dei modelli sono stati utilizzati dati raccolti sperimentalmente, considerando sei condizioni operative differenti dove si è variata, per i primi tre esperimenti, la percentuale in peso iniziale del glicerolo, per poi successivamente variare la temperatura del sistema. Per ogni prova svolta sono state registrate le concentrazioni a quattro tempi differenti, per un totale di tre ore per ciascuna prova eseguita. Nella tabella sotto riportata si presentano le condizioni operative considerate nei diversi casi:

Tabella 2.1: Condizioni operative considerate per le diverse prove sperimentali.

	Prova 1	Prova 2	Prova 3	Prova 4	Prova 5	Prova 6
Conc. $C_3H_8O_3$ (wt%)	5	10	15	5	5	5
Temperatura (K)	513	513	513	523	533	543

Di seguito viene riportata, invece, la tabella contenente le concentrazioni delle specie ottenute con le suddette prove, tenendo conto che per il primo scenario a quattro specie non sono stati considerati i dati raccolti per l'etilen-glicole e il propanolo:

Tabella 2.2: Dati raccolti nelle sei prove sperimentali a $t = 0, t = 0.75\text{h}, t = 1.5\text{h}, t = 3\text{h}$.

	$C_G(mol/m^3)$	$C_{H_2}(mol/m^3)$	$C_{CO_2}(mol/m^3)$	$C_{PG}(mol/m^3)$	$C_{EG}(mol/m^3)$	$C_{1P}(mol/m^3)$
P1: $t = 0$	529.22	19.88	11.08	11.21	1.04	0.00
P1: $t = 0.75$h	457.70	66.24	43.82	36.90	3.88	0.54
P1: $t = 1.5$h	435.63	75.81	46.85	54.85	5.47	1.82
P1: $t = 3$h	368.50	102.12	69.21	83.86	9.81	5.45
P2: $t = 0$	1064.67	23.06	12.45	16.76	1.13	0
P2: $t = 0.75$h	989.79	64.25	36.86	51.51	3.75	1.01
P2: $t = 1.5$h	942.81	85.62	55.45	76.88	6.34	2.78
P2: $t = 3$h	871.15	101.31	82.07	110.88	9.93	8.65
P3: $t = 0$	1601.26	15.87	9.63	15.52	0.00	0.00
P3: $t = 0.75$h	1548.18	65.97	45.67	65.38	3.92	0.00
P3: $t = 1.5$h	1480.35	76.60	50.94	91.57	6.72	3.91
P3: $t = 3$h	1329.99	94.09	88.95	125.75	10.31	11.88
P4: $t = 0$	521.58	24.11	13.04	13.11	0.00	0.00
P4: $t = 0.75$h	427.82	97.64	58.62	48.61	5.20	1.06
P4: $t = 1.5$h	380.84	116.88	68.99	67.68	8.27	4.22
P4: $t = 3$h	314.44	154.42	135.33	119.86	13.00	13.62
P5: $t = 0$	481	52.38	31.89	27.73	2.68	0.00
P5: $t = 0.75$h	381.38	118.49	66.57	65.20	7.04	3.33
P5: $t = 1.5$h	292.28	152.27	133.89	96.51	12.05	13.82
P5: $t = 3$h	224.76	169.14	168.89	93.98	18.56	34.2
P6: $t = 0$	427.25	104.61	54.45	45.44	5.21	0.75
P6: $t = 0.75$h	310.18	175.61	98.68	79.82	11.30	9.97
P6: $t = 1.5$h	233.87	199.89	125.20	89.82	19.08	27.20
P6: $t = 3$h	153.82	207.09	182.01	64.71	20.76	59.84

La Tabella 2.2 costituisce il dataset utilizzato per il *training* delle reti costruite. Come già evidenziato in precedenza, quando si fa uso di set di dati raccolti mediante prove sperimentali, questi hanno dimensioni tipicamente ridotte. Tale aspetto non è vantaggioso quando lo scopo è quello di allenare un modello di *deep learning*: da qui l'esigenza di vincolare cineticamente e termodinamicamente l'apprendimento delle reti al fine di ottimizzare al meglio i parametri del modello [3].

2.4 Setting del training

Prima di passare al commento dei risultati ottenuti, in questo paragrafo viene riportata una sintetica descrizione relativa all'impostazione dei *training* eseguiti, partendo dal funzionamento del *solver* scelto per la risoluzione delle ODEs all'interno dei modelli implementati: il metodo di Runge-Kutta di quarto ordine. Per problemi descritti da equazioni differenziali con condizione di Cauchy -come il caso di interesse- viene spesso utilizzato a causa dell'elevata accuratezza ed efficienza computazionale [33]. L'algoritmo implementato all'interno di tale *solver* è il seguente:

$$\begin{cases} y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ k_1 = f(t_n, y_n) \\ k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right) \\ k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right) \\ k_4 = f\left(t_n + h, y_n + hk_3\right) \end{cases} \tag{2.12}$$

dove il parametro h è dato da:

$$h = t_{n+1} - t_n \tag{2.13}$$

Tale schema si può ottenere integrando l'espansione di Taylor arrestata al quarto ordine della soluzione reale; si tratta, inoltre, di un metodo numerico esplicito, non contenente quindi termini dipendenti da y_{n+1} nella parte destra della prima equazione del sistema 2.12. Tale caratteristica rende i metodi espliciti computazionalmente economici, tuttavia essi possono essere implementati per la risoluzione di problemi di tipo non-stiff [33].

Per quanto riguarda il *setting* degli iper-parametri, i *training* dei modelli sono stati eseguiti impostando un valore di *learning rate* pari a $1 \cdot 10^{-4}$ e un numero di *epochs* pari a 90000. Per il calcolo della *loss function*, infine, è stato implementato il MSE (equazione 1.2). Si procede ora a presentare e commentare i risultati ottenuti considerando i due scenari a quattro e sei specie.

2.5 Cinetiche analitiche

I risultati del *fitting* dei dati sperimentali (Tabella 2.2) mediante l'utilizzo dei modelli allenati sono stati comparati, sia per il caso a quattro specie sia per il caso a sei specie, ai risultati che invece possono essere ottenuti andando ad utilizzare delle cinetiche analitiche, costruite tradizionalmente a partire da un modello di cinetica di Langmuir-Hinshelwood-Hougen-Watson (LHHW). Tali equazioni sono state ricavate per le prime due reazioni chimiche presentate nella sezione 2.1, ovvero il reforming di glicerolo con formazione di idrogeno e la reazione di quest'ultimo con il glicerolo per la produzione di propilen-glicole. Le cinetiche implementate sono riportate di seguito:

$$\begin{aligned} R_1 &= kg_{\text{cat}} \frac{k_1 C_{\text{GLY}}}{1 + K_{\text{GLY}} C_{\text{GLY}} + K_{H_2} C_{H_2} H} \\ R_2 &= kg_{\text{cat}} \frac{k_2 C_{\text{GLY}} C_{H_2} H}{(1 + K_{\text{GLY}} C_{\text{GLY}} + K_{H_2} C_{H_2} H)^2} \end{aligned} \tag{2.14}$$

dove la massa di catalizzatore utilizzato (kg_{cat}) è pari a $3 \cdot 10^{-4}$ kg. Le costanti cinetiche $k1$ e $k2$ e quella di Henry H sono state modellate nel seguente modo:

$$\begin{aligned} k_1 &= A_1 \cdot e^{-\frac{E_{a1}}{RT}} \\ k_2 &= A_2 \cdot e^{-\frac{E_{a2}}{RT}} \\ H &= 7.8 \cdot 10^{-6} \cdot RT \cdot e^{530\left(\frac{1}{T} - \frac{1}{298}\right)} \end{aligned} \tag{2.15}$$

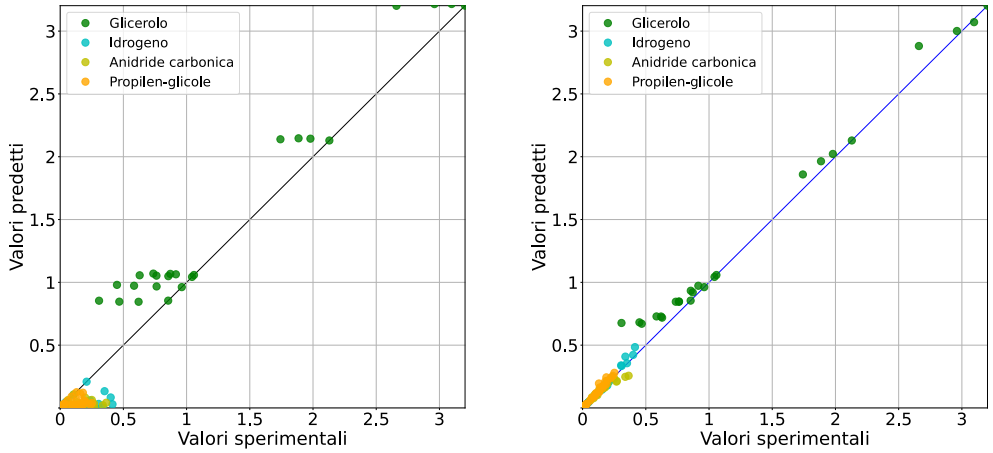
Di seguito si riportano in una tabella i valori dei parametri cinetici ottenuti a partire dall'ottimizzazione del *fitting* dei dati sperimentali, con le corrispondenti unità di misura:

Tabella 2.3: Valori dei parametri presenti nelle cinetiche analitiche.

Parametro	U. misura	Valore
A_1	$\frac{1}{\text{s kg}_{\text{cat}}}$	$1.11 \cdot 10^8$
A_2	$\frac{1}{\text{s kg}_{\text{cat}}}$	$5.32 \cdot 10^{16}$
Ea_1	$\frac{\text{J}}{\text{mol}}$	$9.98 \cdot 10^3$
Ea_2	$\frac{\text{J}}{\text{mol}}$	$3.51 \cdot 10^3$
K_{GLY}	$\frac{\text{m}^3}{\text{mol}}$	$4.4 \cdot 10^5$
K_{H_2}	$\frac{\text{m}^3}{\text{mol}}$	$3.63 \cdot 10^8$

2.6 Risultati: quattro specie

Per il caso a quattro specie, il processo di *training* del modello ha portato all'ottenimento di un valore minimo di *loss function* pari a $6.694 \cdot 10^{-3}$. La bontà dell'apprendimento da parte della rete è stata inoltre valutata attraverso l'utilizzo dei diagrammi di parità, dove sull'asse delle ascisse vengono riportati i valori normalizzati delle concentrazioni sperimentali e sull'asse delle ordinate quelli predetti dal modello. La diagonale del grafico rappresenta la funzione identità: più i punti si avvicinano ad essa, più il valore predetto sarà simile a quello reale. I grafici ottenuti sono riportati di seguito:



(a) Diagramma di parità prima del *training* **(b)** Diagramma di parità dopo il *training*

Figura 2.2: Diagrammi di parità del modello a quattro specie prima e dopo il *training*.

Come si può osservare dalla Figura 2.2b alcune predizioni del modello non approssimano in maniera efficiente i dati sperimentali, in particolare per quanto riguarda il glicerolo, che tende ad essere sovrastimato rispetto al valore reale. Per le altre specie, invece, le concentrazioni predette si avvicinano maggiormente a quelle sperimentali.

Al fine di valutare la stabilità numerica del processo di apprendimento e la bontà dell'ottimizzazione dei parametri del modello, è utile analizzare l'andamento della *loss function* in funzione dell'avanzamento delle *epochs* di *training*. Il grafico ottenuto per questo scenario è riportato nella Figura 2.3: si può osservare come, nella prima fase del *training*, il valore della funzione di costo diminuisca rapidamente all'avanzare del numero di *epochs* fino a assumere un comportamento pressoché costante a circa 25000 iterazioni. Tale andamento è indice di una buona stabilità

dell'algoritmo, in quanto non sono presenti picchi improvvisi della *loss* e le sue oscillazioni hanno ampiezze contenute. Inoltre, il raggiungimento di un valore minimo costante indica che l'ottimizzazione dei parametri è giunta a convergenza. Da un punto di vista numerico, dunque, il *training* eseguito considerando la presenza delle quattro specie principali coinvolte nel processo è stabile.

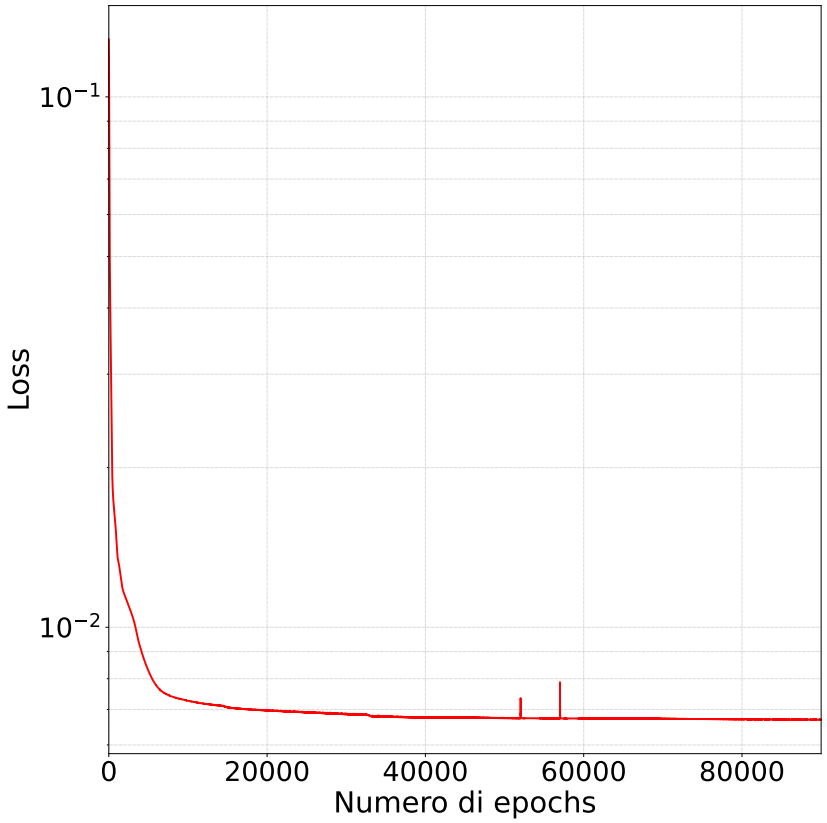


Figura 2.3: Andamento della *loss function* durante il training del modello a quattro specie.

2.6.1 Confronto con dati sperimentali

Per meglio poter valutare la bontà dell'approssimazione del modello ottimizzato sono stati osservati gli andamenti delle concentrazioni al variare del tempo predetti dalla rete, mettendoli a confronto con i punti sperimentali e con le cinetiche analitiche (sezione 2.5), ottenendo i seguenti risultati:

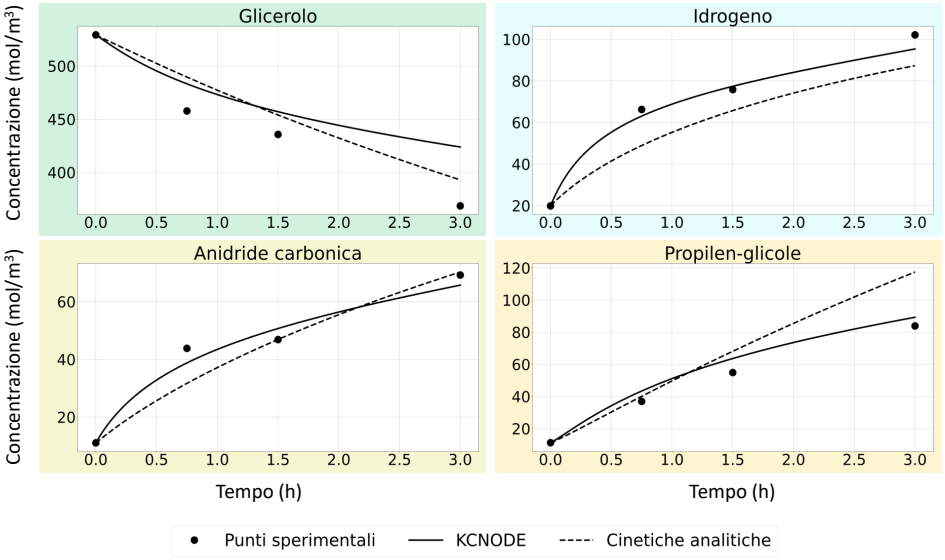


Figura 2.4: Concentrazioni VS tempo per la prova 1: glicerolo al 5 wt% e T = 513K.

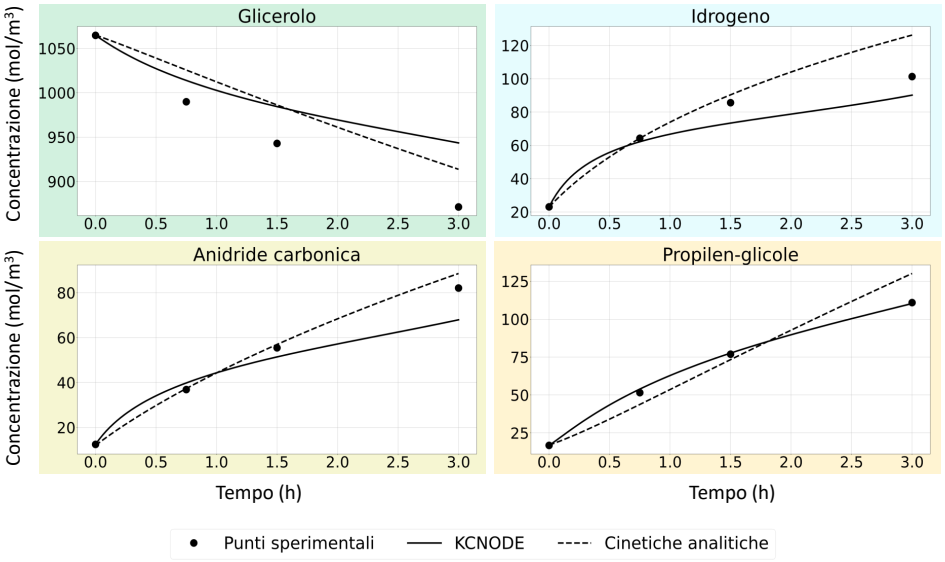


Figura 2.5: Concentrazioni VS tempo per la prova 2: glicerolo al 10 wt% e T = 513K.

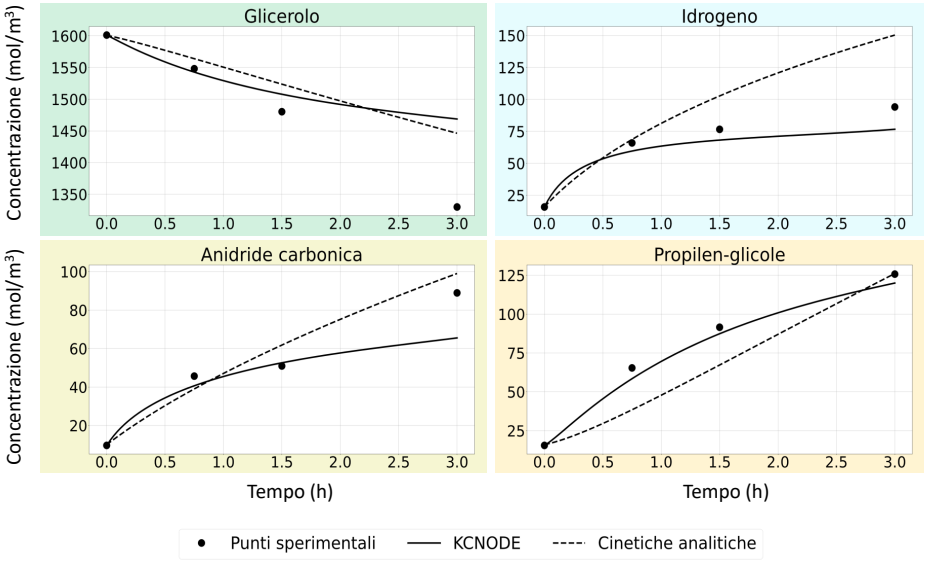


Figura 2.6: Concentrazioni VS tempo per la prova 3: glicerolo al 15 wt% e T = 513K.

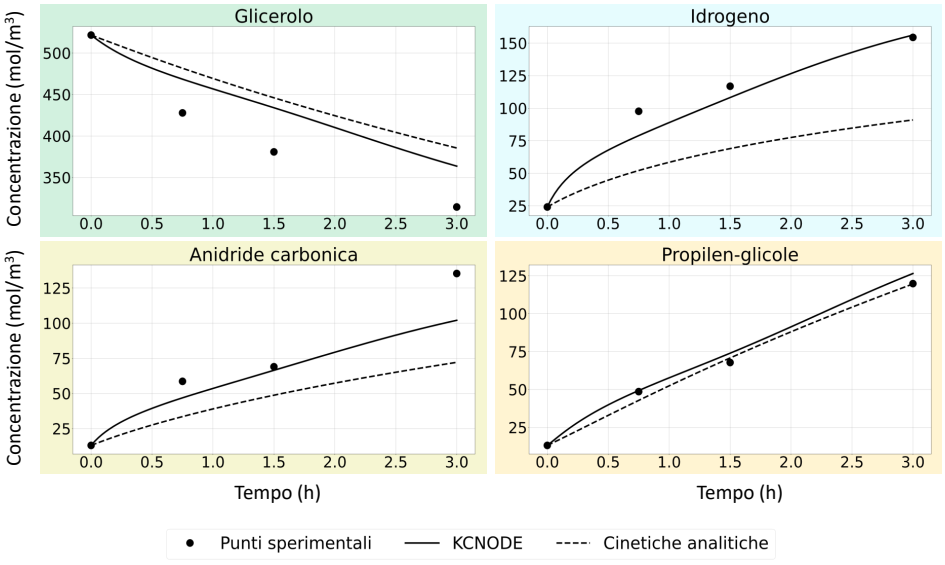


Figura 2.7: Concentrazioni VS tempo per la prova 4: glicerolo al 5 wt% e T = 523K.

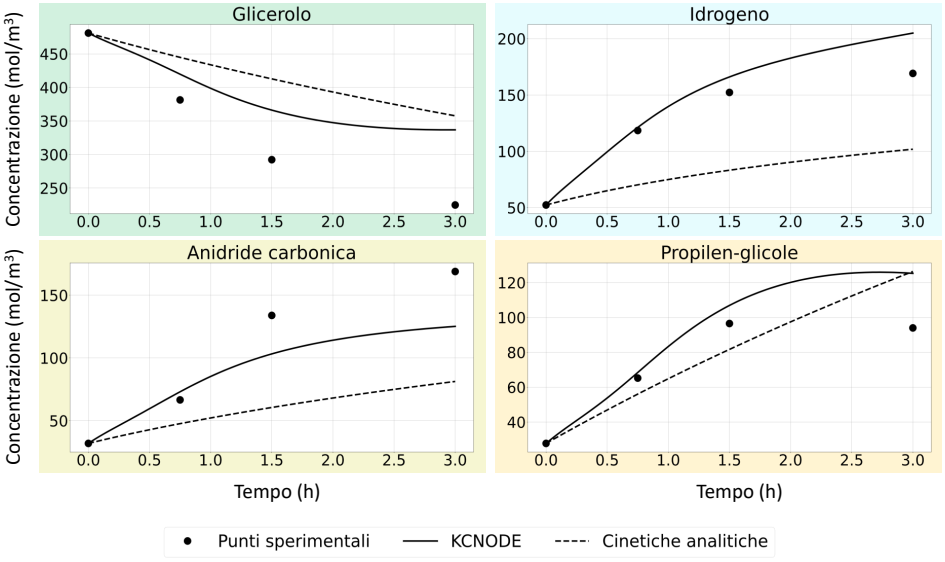


Figura 2.8: Concentrazioni VS tempo per la prova 5: glicerolo al 5 wt% e T = 533K.

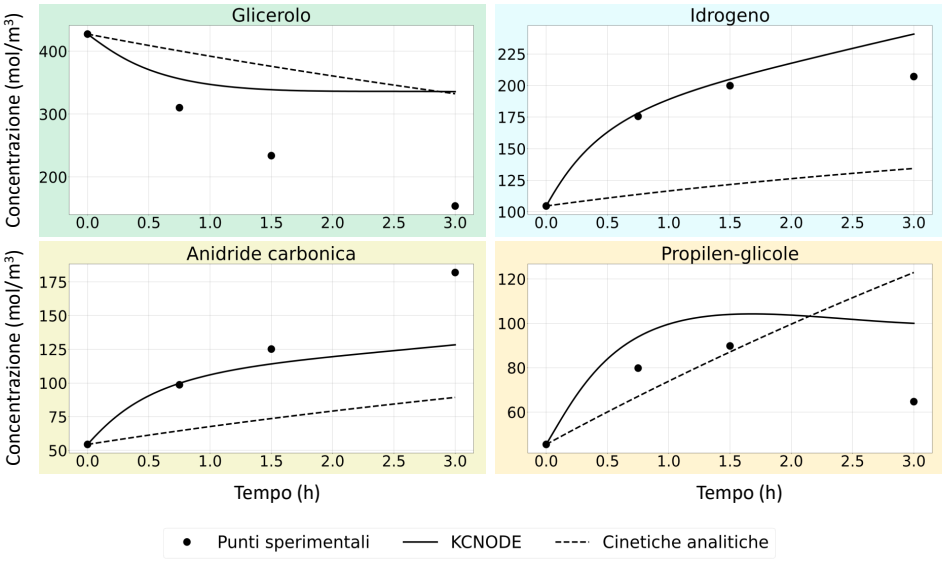


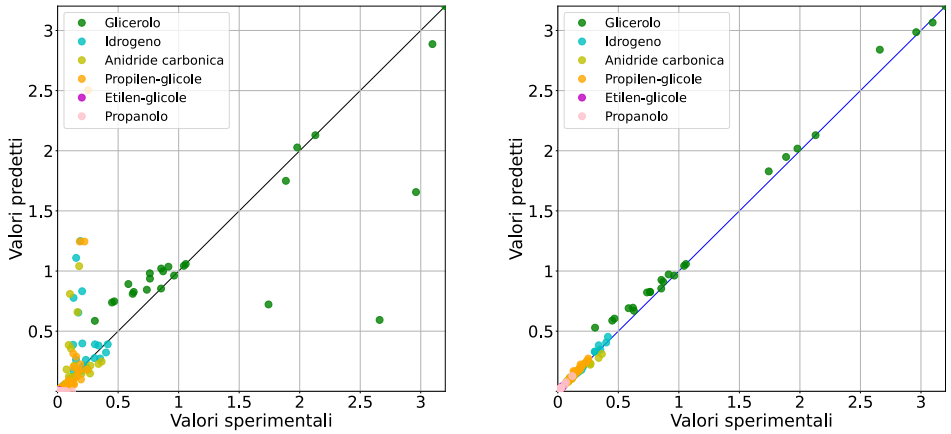
Figura 2.9: Concentrazioni VS tempo per la prova 6: glicerolo al 5 wt% e T = 543K.

I grafici ottenuti evidenziano l'incapacità da parte del modello di predire in maniera affidabile l'andamento della concentrazione di glicerolo. Tale limite, inoltre, viene accentuato dall'aumento della temperatura del sistema. In Figura 2.9, infatti, il valore della concentrazione di glicerolo stimata dalla rete dopo tre ore di processo è di circa $340\,\mathrm{mol/m^3}$ a fronte di una concentrazione sperimentale di circa $150\,\mathrm{mol/m^3}$. Inoltre, a valori di temperatura maggiori il modello non è in grado di predire opportunamente anche la concentrazione di propilen-glicole, la quale presenta un comportamento non monotono. Tuttavia, il consumo di tale specie chimica non è previsto da nessuna delle due reazioni considerate all'interno di questo primo caso: di conseguenza, il modello non sarà in grado di associare tale comportamento ai vincoli imposti nella definizione dell'architettura.
Una possibile causa legata ai limiti predittivi del modello potrebbe essere lo sviluppo di meccanismi di reazione paralleli a quelli considerati, che comportano un maggiore consumo di glicerolo ed un consumo di propilen-glicole quando la temperatura del sistema aumenta. L'intervallo ottimale di tale parametro, al fine di massimizzare la selettività verso l'idrogeno, è infatti di $470-520\,\mathrm{K}$ [11]. Per le prove riportate nelle Figure 2.8 e 2.9 la temperatura è maggiore del range sopra definito: si potrebbero quindi favorire altri cammini di reazione, sfavorendo il reforming del glicerolo [32].

Un aspetto positivo già evidente in questo scenario semplificato si riscontra confrontando le predizioni effettuate dal modello (curve continue nelle figure) con l'andamento delle concentrazioni stimato mediante cinetiche analitiche tradizionali (curve tratteggiate): salvo le suddette criticità, la rete effettua delle approssimazioni generalmente migliori rispetto al *fitting* ottenuto analiticamente. Tale risultato risulta essere promettente: l'utilizzo delle KCNODEs permette di evitare la ricerca di un modello analitico che approssimi opportunamente l'andamento sperimentale delle concentrazioni e, inoltre, fornisce risultati migliori.

2.7 Risultati: sei specie

Per cercare di potenziare le prestazioni del modello precedente sono state introdotte due reazioni chimiche aggiuntive, la prima delle quali comporta un ulteriore consumo di glicerolo con produzione di idrogeno e di etilen-glicole, specie chimica non considerata nel primo scenario. La seconda reazione, invece, introduce un termine di consumo per il propilen-glicole, che, reagendo con l'idrogeno, genera propanolo. In questo secondo caso, in sintesi, si sono considerate sei specie complessivamente coinvolte in quattro reazioni chimiche. Le modalità di *training* impostate sono le stesse utilizzate per il modello precedente (sezione 2.4) ed i diagrammi di parità ottenuti sono riportati di seguito:



(a) Diagramma di parità prima del *training* **(b)** Diagramma di parità dopo il *training*

Figura 2.10: Diagrammi di parità del modello a sei specie prima e dopo il *training*.

Confrontando le Figure 2.2b e 2.10b si può facilmente osservare il miglioramento nella capacità di predizione della rete per la concentrazione di glicerolo: è ancora affetta da una leggera sovrastima dei valori, che è tuttavia più contenuta rispetto al modello precedente. Nel complesso, dal diagramma di parità le prestazioni appaiono migliori.
Per l'analisi di stabilità numerica e bontà di ottimizzazione dei parametri è stato analizzato, come per la sezione 2.6, il comportamento della *loss function* all'avanzare del numero di *epochs*:

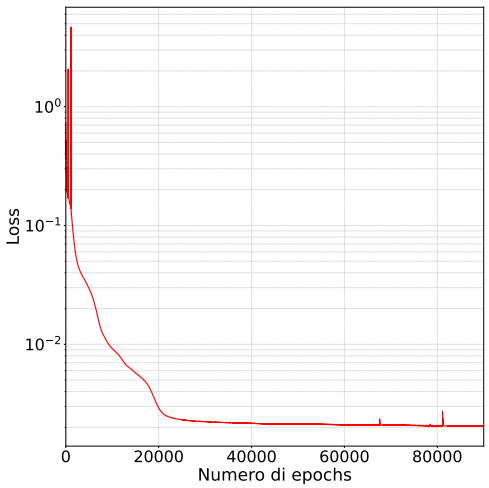


Figura 2.11: Andamento della *loss function* durante il training del modello a sei specie.

L'andamento riportato in Figura 2.11 è del tutto analogo a quello osservato per la funzione di costo del primo scenario, con l'unica differenza che per le prime *epochs* si ha un comportamento fortemente oscillante, che tuttavia tende a stabilizzarsi e a decrescere in maniera stabile. Per lo scenario a sei specie, inoltre, la *best loss* ottenuta è pari a $2.09 \cdot 10^{-3}$, tre volte più piccola rispetto al valore ottenuto con quattro specie.

2.7.1 Confronto con dati sperimentali

Come per il modello a quattro specie, si presentano i risultati acquisiti dal confronto delle predizioni eseguite dalla rete con i dati sperimentali e le equazioni analitiche, le quali sono state ricavate per le sole due equazioni già considerate nel primo caso: per l'etilen-glicole e il propanolo, di conseguenza, non c'è un termine di paragone analitico, in quanto coinvolti solamente nelle due nuove reazioni.

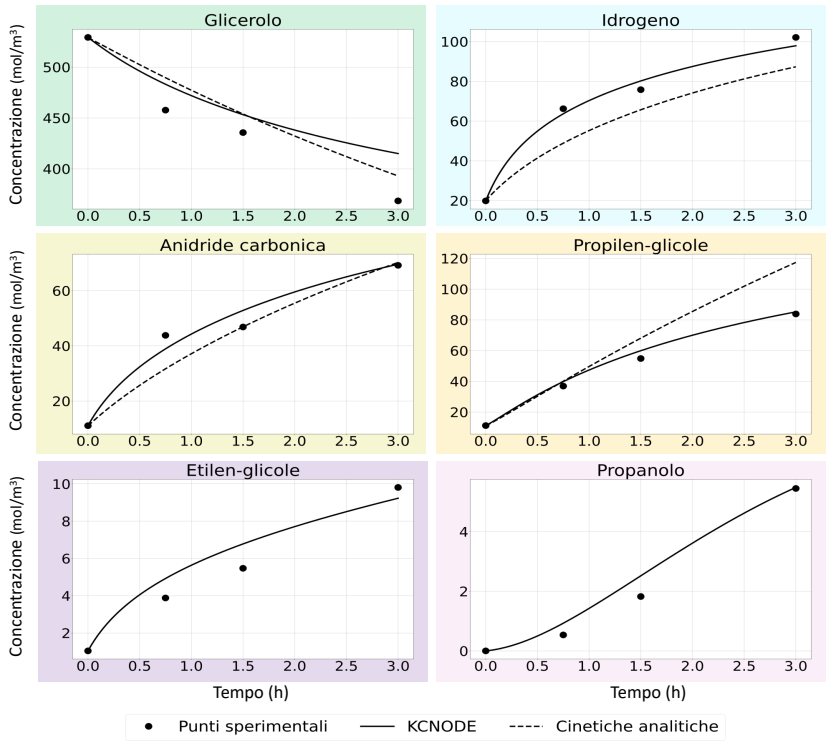


Figura 2.12: Concentrazioni VS tempo per la prova 1: glicerolo al 5 wt% e T = 513K.

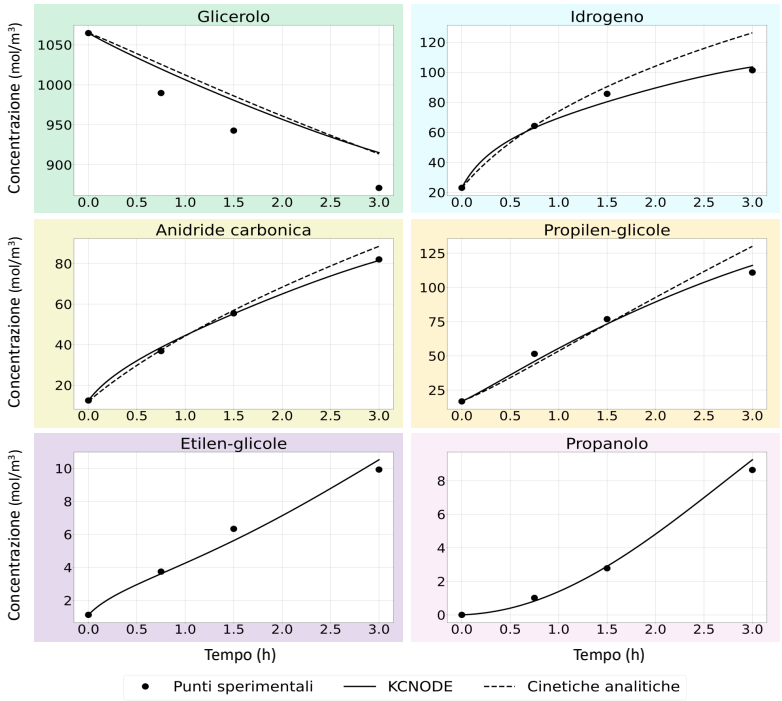


Figura 2.13: Concentrazioni VS tempo per la prova 2: glicerolo al 10 wt% e T = 513K.

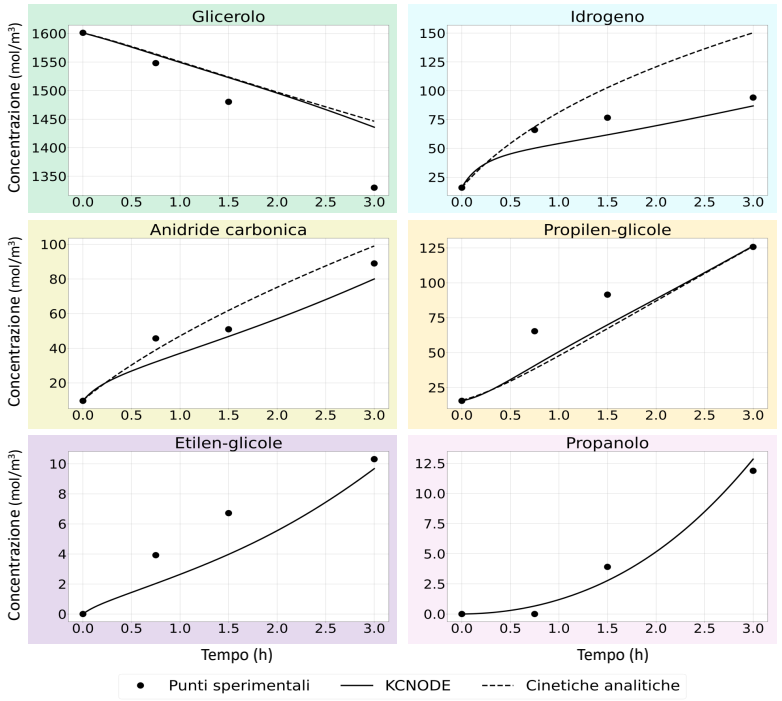


Figura 2.14: Concentrazioni VS tempo per la prova 3: glicerolo al 15 wt% e T = 513K.

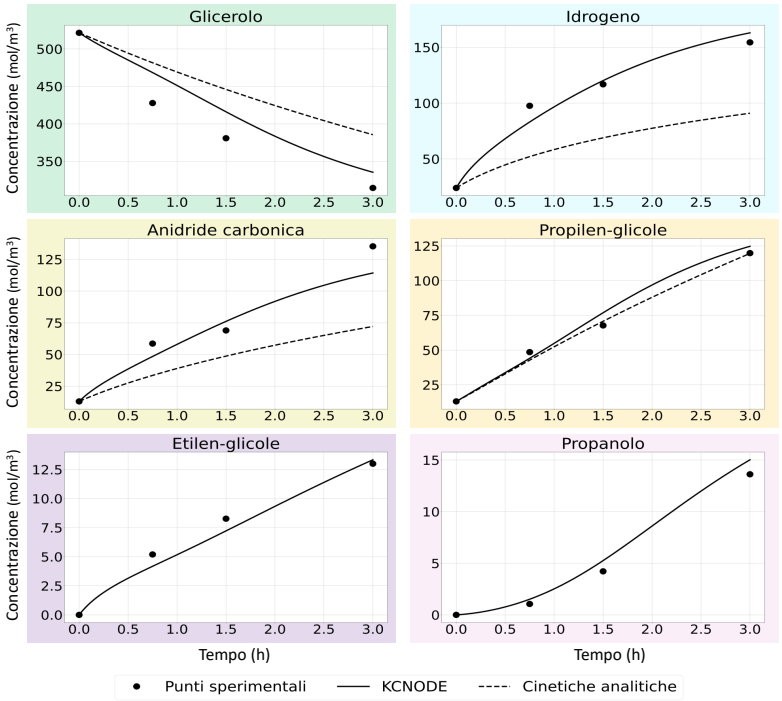


Figura 2.15: Concentrazioni VS tempo per la prova 4: glicerolo al 5 wt% e T = 523K.

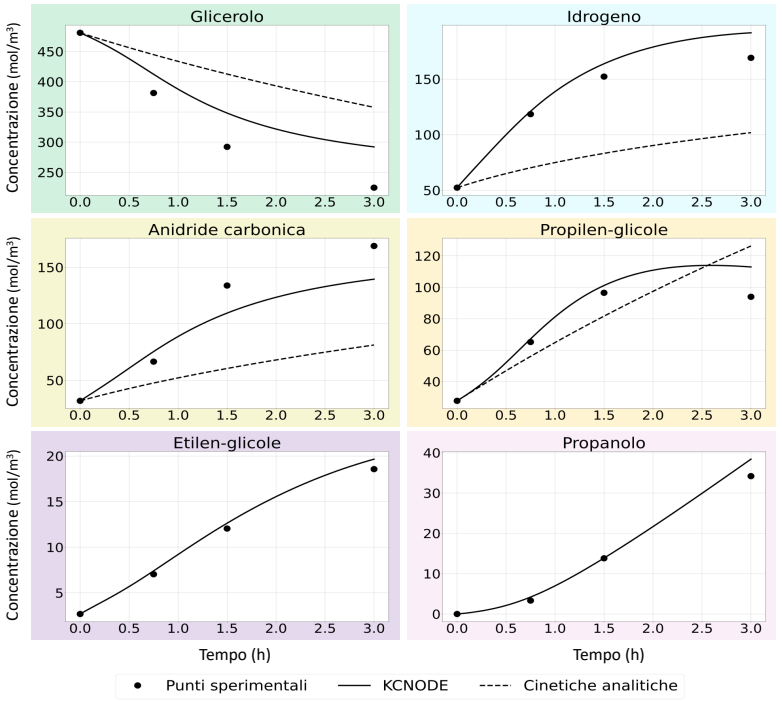


Figura 2.16: Concentrazioni VS tempo per la prova 5: glicerolo al 5 wt% e T = 533K.

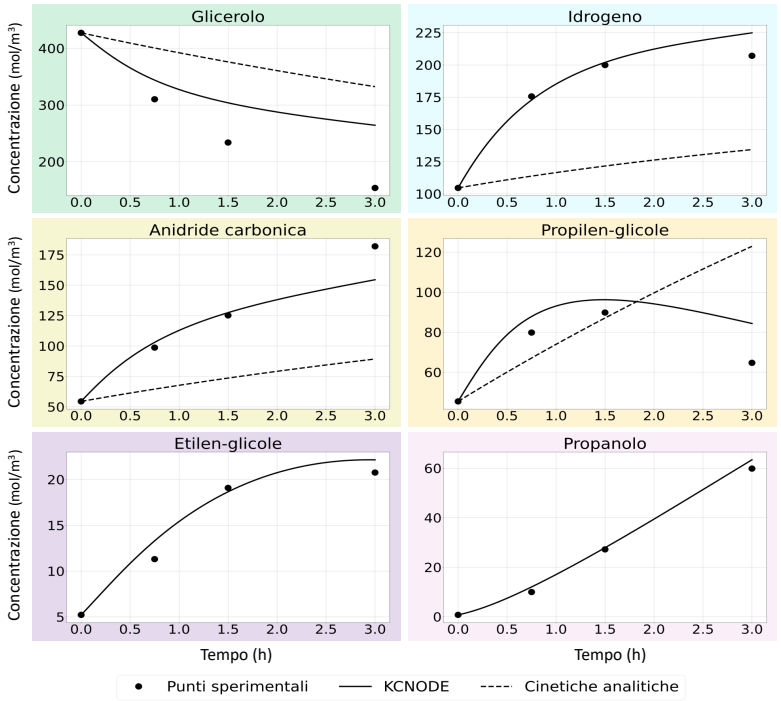


Figura 2.17: Concentrazioni VS tempo per la prova 6: glicerolo al 5 wt% e T = 543K.

Come già anticipato nel confronto tra i diagrammi di parità acquisiti al termine dei *training*, i risultati ottenuti andando a considerare un maggior numero di specie chimiche e di reazioni sono più promettenti: dalle figure sopra riportate, infatti, si evince come l'andamento nel tempo della concentrazione di propilen-glicole predetto dalla rete sia più coerente con i dati sperimentali rispetto alle predizioni presentate nelle Figure 2.4-2.9. Anche per quanto riguarda il glicerolo è stato osservato un miglioramento dell'approssimazione effettuata dal modello, seppur esso sia più contenuto: per alcune condizioni operative, infatti (ad esempio la prova prova 3 e la prova 6) i valori di concentrazione predetti sono ancora significativamente maggiori rispetto a quelli sperimentali. Le concentrazioni delle altre specie chimiche vengono opportunamente approssimate, generando delle curve in grado di interpolare in maniera efficace le concentrazioni reali registrate durante le prove sperimentali.

In linea generale, il modello ottenuto considerando lo sviluppo di diverse reazioni parallele al reforming di glicerolo ha permesso di ottenere delle predizioni migliori al caso precedente. Inoltre, andando a confrontare le curve ottenute con l'utilizzo delle KCNODEs in questo scenario con l'andamento descritto dalle equazioni analitiche, si può osservare come la bontà di approssimazione delle reti implementate sia decisamente maggiore.

2.8 Confronto KCNODEs e cinetiche analitiche

Nelle sezioni precedenti sono stati illustrati i risultati ottenuti implementando le KCNODEs per la modellazione delle cinetiche di reazione coinvolte nel processo di APR del glicerolo. Siamo partiti da un caso semplificato, dove si sono considerate quattro specie chimiche coinvolte in due meccanismi di reazione differenti, per passare successivamente ad uno scenario più complesso, dove sono state aggiunte due reazioni rispetto a quelle già tenute presenti, le quali comportano la produzione di due specie chimiche aggiuntive.
Come già evidenziato in precedenza, le predizioni ottenute, specialmente con il secondo modello implementato, sono soddisfacenti; inoltre, è importante ricordare come l'utilizzo di tali modelli di *deep learning* permetta di evitare il complicato processo di ricerca di equazioni analitiche che siano in grado di approssimare l'andamento delle concentrazioni nel tempo all'interno di un processo chimico.

Al fine di effettuare un confronto numerico tra predizioni effettuate dalle reti e funzioni analitiche, sono stati calcolati gli errori, espressi come MSE e MAE, associati alle singole prove sperimentali considerate per entrambi i modelli sviluppati e per le cinetiche proposte nella sezione 2.5, ottenendo i valori riportati nella seguente tabella:

Tabella 2.4: MSE e MAE ottenuti con i modelli a quattro e sei specie e con le equazioni analitiche.

	MSE 4 specie	MSE 6 specie	MSE analitiche	MAE 4 specie	MAE 6 specie	MAE analitiche
Prova 1	0.0011	0.0005	0.0014	0.0175	0.0102	0.0226
Prova 2	0.0014	0.0006	0.0019	0.0197	0.0101	0.0239
Prova 3	0.0034	0.0015	0.0056	0.0233	0.0126	0.0427
Prova 4	0.0017	0.0007	0.0071	0.0262	0.0150	0.0581
Prova 5	0.0064	0.0019	0.0163	0.0507	0.0240	0.0922
Prova 6	0.0132	0.0033	0.0235	0.0634	0.0263	0.1095
TOTALE	**0.0045**	**0.0014**	**0.0093**	**0.0335**	**0.0164**	**0.0582**

I valori ottenuti sono stati calcolati utilizzando i valori di concentrazione normalizzati dividendo per 500, al fine di ottenere dati in input alla rete nell'ordine dell'unità e conferire una maggiore stabilità numerica al processo di *training*.
Osservando la tabella, gli errori valutati per le singole prove sperimentali confermano come l'utilizzo delle KCNODEs in entrambi gli scenari considerati fornisca delle prestazioni migliori nella stima dell'andamento delle concentrazioni nel tempo rispetto alle approssimazioni ottenibili utilizzando le cinetiche analitiche. Tale miglioramento della bontà di predizione da parte dei modelli implementati diventa più evidente all'aumentare della temperatura del sistema: per la sesta prova, infatti, il MAE ottenuto con le equazioni analitiche ha un valore circa quattro volte superiore rispetto a quello calcolato per il modello a sei specie.

Confrontando invece i due casi di implementazione delle KCNODEs, si può concludere come l'utilizzo di uno scenario più complesso e dettagliato permetta di migliorare la capacità predittiva del modello: in tutte le prove sperimentali considerate, infatti, i valori di MSE e MAE ottenuti per il caso a sei specie sono minori degli stessi errori calcolati per il caso a quattro specie. In conclusione alla presentazione di questa prima parte di risultati ottenuti nel progetto svolto, è possibile affermare che:

- l'implementazione delle KCNODEs per la stima dell'andamento delle concentrazioni nel tempo durante un processo chimico quale l'APR ha permesso di ottenere risultati soddisfacenti;
- tali modelli presentano capacità predittive significativamente superiori a quelle delle equazioni analitiche;
- al fine di incrementare la qualità delle predizioni, è utile fornire più informazioni possibili al modello relativamente alle reazioni chimiche che possono svilupparsi durante l'esecuzione del processo. Tali vincoli, infatti, permettono alla rete di effettuare un'ottimizzazione dei parametri migliore e, di conseguenza, di stimare con maggiore precisione le variazioni temporali delle concentrazioni.

3 Aspetti teorici: CFD

In questo capitolo vengono riportate le equazioni di trasporto che descrivono la fisica del sistema analizzato nel lavoro di tesi (3.1), per passare successivamente all'analisi degli aspetti numerici legati alle simulazioni di fluidodinamica computazionale (3.2).
Per l'esecuzione delle simulazioni CFD è stata utilizzata la versione 9 del programma *open-source* `OpenFOAM` (*Open-source Field Operation And Manipulation*), il quale utilizza come linguaggio di programmazione `C++` che, in questo caso di studio, è stato accoppiato all'utilizzo della libreria `libtorch`, necessaria all'implementazione dei modelli cinetici *data-driven*.

3.1 Equazioni di governo

Prima di presentare le equazioni implementate per descrivere il trasporto delle specie chimiche all'interno del letto impaccato, modellato come un mezzo poroso, si riportano le ipotesi formulate sul sistema:

- Il sistema è stato considerato altamente diluito; di conseguenza, le proprietà del fluido, quali la densità e la viscosità cinematica, sono state considerate pari a quelle dell'acqua;

- I valori di diffusione molecolare considerati per il calcolo della dispersione nel mezzo poroso delle specie chimiche coinvolte tengono conto della sola interazione che tali specie hanno con l'acqua, trascurando possibili influenze date dalla presenza di altre sostanze;

- Si è ipotizzato che il fluido modellato, essendo allo stato liquido, abbia densità costante e quindi sia incomprimibile;

- Il fluido presenta una viscosità costante al variare dello *strain rate* e viene considerato come newtoniano;

- Il solido è considerato isotropo; di conseguenza, i parametri che dipendono dal mezzo poroso, come ad esempio la permeabilità K, sono indipendenti dalla direzione considerata.

Le equazioni risolte all'interno delle simulazioni per descrivere il flusso nel sistema vengono di seguito presentate in forma indiciale:

- L'equazione di continuità, che nel caso di un fluido incomprimibile, ha la seguente forma:

$$\frac{\partial U_i}{\partial x_i} = 0 \tag{3.1}$$

  dove U_i rappresenta l'i-esima componente del vettore velocità, espressa in m/s;

- L'equazione di Navier-Stokes nel caso di fluido incomprimibile e newtoniano, con termine sorgente S_i costituito dal contributo dato dalla presenza di un mezzo poroso:

$$\frac{\partial U_i}{\partial t} + U_j \frac{\partial U_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 U_i}{\partial x_j^2} + S_i \tag{3.2}$$

  Nell'equazione sopra riportata ρ rappresenta la densità del fluido (kg/m^3), mentre ν la sua viscosità cinematica (m^2/s). Per entrambi i parametri si sono considerati quelli dell'acqua, pari rispettivamente a 10^3 kg/m^3 e 10^{-6} m^2/s;

- Il termine sorgente S_i riportato nell'equazione 3.2, come già anticipato, è dato dalla presenza del catalizzatore all'interno del sistema, il quale è stato modellato come un mezzo poroso. Tale contributo viene calcolato mediante l'utilizzo dell'equazione di Darcy, la quale descrive la dipendenza lineare tra la velocità di un fluido che attraversa un solido poroso e la caduta di pressione [34]. L'equazione è la seguente:

$$U_i = -\frac{K}{\mu}\frac{\partial p}{\partial x_i} \tag{3.3}$$

  dove K rappresenta la permeabilità del solido, espressa in m^2, μ è la viscosità dinamica del fluido in $Pa \cdot s$, mentre U_i e $\frac{\partial p}{\partial x_i}$ sono, rispettivamente, la velocità ed il gradiente di pressione lungo la coordinata i. Nello studio svolto, abbiamo ipotizzato di avere una perdita di carico lungo la sola coordinata assiale del reattore modellato. Di conseguenza, l'equazione 3.3 può essere riformulata come segue:

$$U = -\frac{K}{\mu}\frac{\Delta p}{L} \tag{3.4}$$

  dove L rappresenta la lunghezza del reattore espressa in metri. Il segno negativo indica una diminuzione di pressione nella direzione di L quando il flusso

ha un valore positivo [35].
L'equazione di Navier-Stokes accoppiata con quella di Darcy è stata implementata per ricavare il campo di moto all'interno del reattore ed il profilo di pressione;

- L'ultima equazione implementata, necessaria a modellare la fisica alla base del trasporto delle specie chimiche presenti nel sistema, è l'equazione di advezione e diffusione, modificata aggiungendo il termine sorgente costituito dalle cinetiche *data-driven* ottenute implementando le NODEs:

$$\frac{\partial C_n}{\partial t} + \frac{\partial (U_i C_n)}{\partial x_i} = D_n \frac{\partial^2 C_n}{\partial x_i^2} + S_n \tag{3.5}$$

  Il primo termine rappresenta il transitorio, il secondo descrive il trasporto dato da fenomeni convettivi, mentre i termini a secondo membro costituiscono i contributi diffusivo e reattivo dell'equazione. Il pedice n fa riferimento alle specie chimiche presenti nel sistema, ad ognuna delle quali è associato un coefficiente di dispersione D_n espresso in m^2/s.

Generalmente le equazioni appena presentate non sono risolvibili analiticamente, in quanto descritte da equazioni differenziali non lineari che contengono derivate rispetto alla variabile temporale e a quelle spaziali. Di conseguenza è necessario utilizzare dei metodi di integrazione numerica che permettano di valutare delle soluzioni approssimate delle equazioni. Il programma `OpenFOAM` fa uso del metodo dei volumi finiti (FVM), il cui funzionamento viene presentato nella sezione a seguire.

3.2 Metodo dei volumi finiti

Il metodo dei volumi finiti richiede che il dominio computazionale venga suddiviso in sotto-volumi definiti, che prendono il nome di celle; all'interno di tali elementi, si considera il valore della generica variabile ϕ costante su tutto il volume. In fluidodinamica computazionale questo metodo riveste un ruolo fondamentale, poiché in grado di trasformare un sistema di equazioni differenziali parziali in un sistema di equazioni lineari algebriche. Il processo di discretizzazione può essere suddiviso in due step principali [36]:

- Le equazioni differenziali vengono integrate e trasformate in bilanci applicati alle singole celle del dominio computazionale;

- Nel secondo step si applicano degli algoritmi di approssimazione che permettono di associare il valore superficiale della variabile ϕ a quello noto all'interno del volume del singolo elemento.

Nella Figura 3.1 si riporta, a scopo di esempio, una rappresentazione di cella bidimensionale (sopra) e tridimensionale (sotto) con nomenclatura corrispondente:

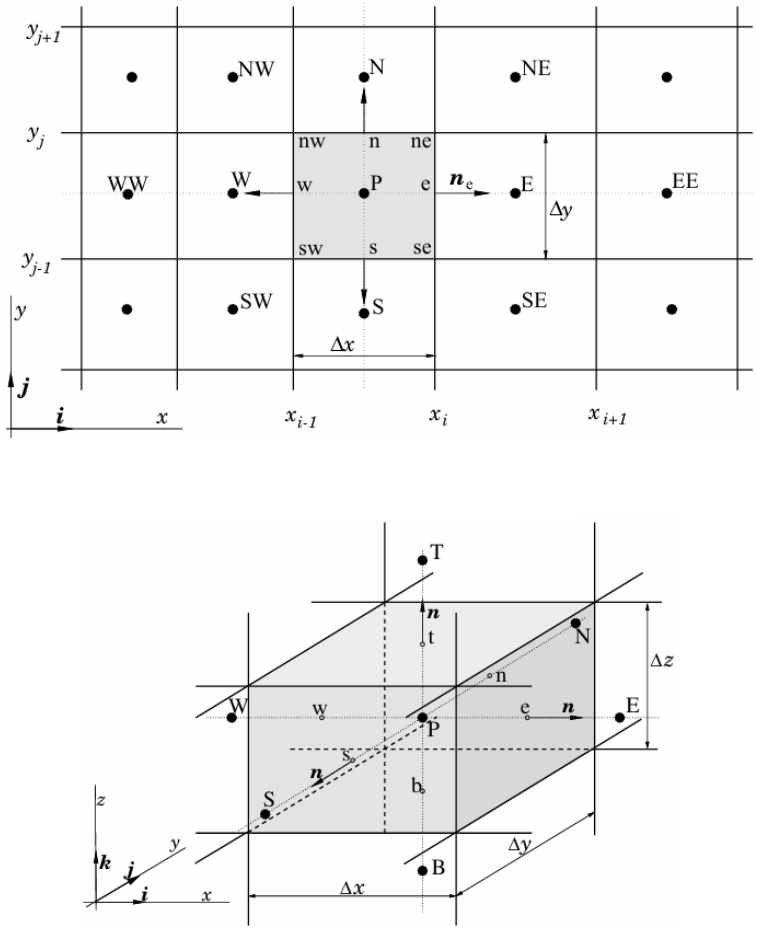


Figura 3.1: Rappresentazione grafica di una cella nei casi di dominio computazionale 2D e 3D [37].

Il primo step quindi prevede l'integrazione dell'equazione di conservazione di una generica variabile ϕ applicata al volume di controllo (CV) della singola cella:

$$\int_V \rho \frac{\partial \phi}{\partial t}\, dV + \int_V \rho \nabla \cdot (\boldsymbol{U}\phi)\, dV = \int_V \nabla(\Gamma\, \nabla\phi)\, dV \; + \int_V q_\phi\, dV \tag{3.6}$$

Successivamente si applica il teorema della divergenza ai termini convettivo e diffusivo, riformulando l'equazione come segue:

$$\int_V \rho \frac{\partial \phi}{\partial t}\, dV + \int_S \rho\phi\, \boldsymbol{U} \cdot \mathbf{n}\, dS = \int_S \Gamma\, \nabla\phi \cdot \mathbf{n}\, dS \; + \int_V q_\phi\, dV \tag{3.7}$$

dove S è la superficie che racchiude il volume di controllo V ed $\mathbf{n}$ il versore normale ad essa. Il flusso netto di ϕ attraverso la superficie S è dato dalla somma degli integrali eseguiti sulle singole facce della cella [37]:

$$\int_S f \, dS = \sum_k \int_{S_k} f \, dS \tag{3.8}$$

dove il parametro k si riferisce al numero di facce del CV ed f alla componente convettiva ($\rho\phi \boldsymbol{U} \cdot \mathbf{n}$) o diffusiva ($\Gamma \nabla\phi \cdot \mathbf{n}$) del flusso normale alla superficie.

Andando ora a considerare il flusso convettivo, l'integrale può essere trasformato in un'equazione algebrica nel seguente modo:

$$\begin{aligned}\int_S \rho\phi \boldsymbol{U} \cdot \mathbf{n} \, dS = -\rho\Big[&(US\phi)_\mathrm{w} - (US\phi)_\mathrm{e} + (VS\phi)_\mathrm{s} - \\ &- (VS\phi)_\mathrm{n} + (WS\phi)_\mathrm{b} - (WS\phi)_\mathrm{t}\Big]\end{aligned} \tag{3.9}$$

All'interno dell'espressione ottenuta per l'integrale convettivo le lettere w, e, s, n, b e t fanno riferimento rispettivamente alle facce *west*, *east*, *south*, *north*, *bottom* e *top* della cella 3D (figura 3.1), mentre U, V e W sono le componenti del vettore velocità lungo le direzioni x, y e z.
Il termine diffusivo può essere approssimato in maniera analoga:

$$\begin{aligned}\int_S \Gamma \nabla\phi \cdot \mathbf{n} \, dS = -\Bigg[&\left(\Gamma S \frac{\partial\phi}{\partial x}\right)_\mathrm{w} - \left(\Gamma S \frac{\partial\phi}{\partial x}\right)_\mathrm{e} + \left(\Gamma S \frac{\partial\phi}{\partial y}\right)_\mathrm{s} - \\ &- \left(\Gamma S \frac{\partial\phi}{\partial y}\right)_\mathrm{n} + \left(\Gamma S \frac{\partial\phi}{\partial z}\right)_\mathrm{b} - \left(\Gamma S \frac{\partial\phi}{\partial z}\right)_\mathrm{t}\Bigg]\end{aligned} \tag{3.10}$$

Infine, il termine di generazione viene approssimato calcolando un valore medio del termine sorgente all'interno del volume della cella:

$$\int_V q_\phi \, dV \approx \bar{q}_\phi V \tag{3.11}$$

Partendo dall'equazione di conservazione in forma integrale (equazione 3.7) si sono ottenute delle espressioni algebriche per i singoli termini (equazioni 3.9, 3.10 e 3.11).
Tuttavia, è necessario interpolare i valori di ϕ e del suo gradiente in corrispondenza delle facce delle celle partendo dai valori della variabile al centro delle celle stesse, in quanto l'unico valore noto e calcolato nel FVM corrisponde a quello nel centro di ogni CV considerato.

3.3 Metodi di discretizzazione spaziale

Per la risoluzione del campo di moto e del trasporto delle concentrazioni all'interno del sistema modellato in `OpenFOAM` si è implementato uno schema di discretizzazione spaziale di tipo *linearUpwind* di secondo ordine. A scopo di esempio, si riporta un'applicazione del metodo a un caso semplificato mono-dimensionale. Si ipotizzi di voler determinare il valore di ϕ in corrispondenza della faccia est riportata nello schema in Figura 3.2 nel caso in cui il flusso sia diretto verso destra:

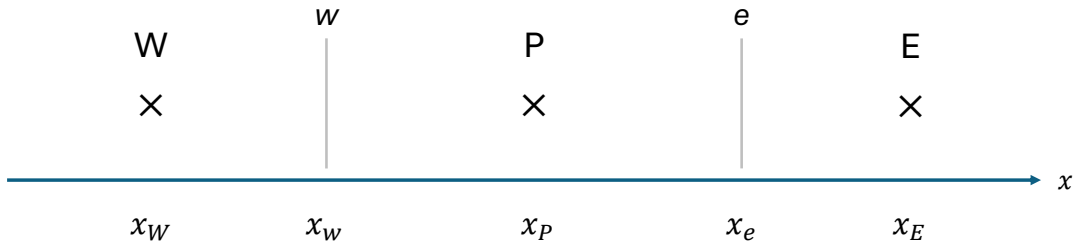


Figura 3.2: Rappresentazione di un caso mono-dimensionale per applicazione di schemi *Upwind*.

Per gli schemi di tipo *Upwind*, il valore di ϕ_i viene approssimato con quello immediatamente "a monte" rispetto al verso del flusso: in questo esempio, il valore di ϕ_w, corrispondente alla faccia sinistra della cella P, verrà posto pari a ϕ_W e così via. Nello schema implementato, tuttavia, tale assegnazione viene modificata andando a considerare anche il gradiente di ϕ_P, valutato utilizzando i valori al centro delle celle W ed E. Il valore di ϕ_e è quindi calcolato come segue:

$$\phi_e = \phi_P + \nabla\phi_P(x_e - x_P) \tag{3.12}$$

con:

$$\nabla\phi_P = \frac{\phi_E - \phi_W}{x_E - x_W} \tag{3.13}$$

Si può dimostrare che l'errore associato allo schema di discretizzazione presentato scala proporzionalmente con il quadrato della dimensione delle celle. Per semplicità, si ipotizza che la griglia computazionale abbia un passo uniforme pari ad h:

$$h = x_W - x_P = x_P - x_E \tag{3.14}$$

L'equazione 3.12 quindi diventa:

$$\phi_e = \phi_P + \frac{\phi_E - \phi_W}{2h}\frac{h}{2} = \phi_P + \frac{\phi_E - \phi_W}{4} \tag{3.15}$$

Al fine di valutare l'ordine dell'errore rispetto ad h è necessario calcolare il valore esatto di $\phi(x_e)$ mediante gli sviluppi di Taylor nell'intorno di x_P, dove H rappresenta i termini di ordine superiore. Per ϕ_E si avrà:

$$\begin{aligned} x_E &= x_P + h \\ \phi_E &= \phi_P + h\phi'_P + \frac{h^2}{2}\phi''_P + \frac{h^3}{6}\phi'''_P + H \end{aligned} \tag{3.16}$$

mentre per ϕ_W:

$$\begin{aligned} x_W &= x_P - h \\ \phi_W &= \phi_P - h\phi'_P + \frac{h^2}{2}\phi''_P - \frac{h^3}{6}\phi'''_P + H \end{aligned} \tag{3.17}$$

Facendo la differenza membro a membro si ottiene:

$$\phi_E - \phi_W = 2h\phi'_P + \frac{h^3}{3}\phi'''_P + H \tag{3.18}$$

da cui:

$$\frac{\phi_E - \phi_W}{4} = \frac{h}{2}\phi'_P + \frac{h^3}{12}\phi'''_P + H \tag{3.19}$$

Sostituendo all'interno dell'equazione 3.15 si può calcolare il valore approssimato ϕ_e come segue:

$$\phi_e = \phi_P + \frac{h}{2}\phi'_P + \frac{h^3}{12}\phi'''_P + H \tag{3.20}$$

Si procede ora sviluppando il polinomio di Taylor del valore esatto di ϕ valutato in corrispondenza di $x_e = x_P + \frac{h}{2}$:

$$\phi(x_e) = \phi_P + \tfrac{h}{2}\,\phi'_P + \tfrac{1}{2}\left(\tfrac{h}{2}\right)^2\phi''_P + \tfrac{1}{6}\left(\tfrac{h}{2}\right)^3\phi'''_P + H \tag{3.21}$$

Sottraendo le equazioni 3.20 e 3.21 si ottiene l'errore di approssimazione associato all'implementazione dello schema *linearUpwind*:

$$\phi_e - \phi(x_e) = -\tfrac{h^2}{8}\phi''_P + \tfrac{h^3}{16}\phi'''_P + H \tag{3.22}$$

Dall'equazione 3.22 si evince come il termine dominante dell'errore sia $-\frac{h^2}{8}\phi''_P$, il quale conferma che lo schema di discretizzazione spaziale *linearUpwind* è del secondo ordine.

3.4 Metodi di discretizzazione temporale

Le simulazioni effettuate con `OpenFOAM` in questo progetto di tesi sono in regime transitorio: il primo termine delle equazioni 3.2 e 3.5 deve quindi essere risolto

numericamente implementando un metodo di discretizzazione temporale. In particolare, si è utilizzato il metodo di Eulero implicito, uno schema del primo ordine ed incondizionatamente stabile, il cui algoritmo di funzionamento è riportato di seguito:

$$\phi_i^{n+1} = \phi_i^n + \Delta t \left(-U \frac{\phi_{i+1}^{n+1} - \phi_{i-1}^{n+1}}{2\Delta x} + \frac{\Gamma(\phi_{i+1}^{n+1} - 2\phi_i^{n+1} + \phi_{i-1}^{n+1})}{\Delta x^2} \right) \tag{3.23}$$

L'indice i fa riferimento alla discretizzazione spaziale presentata nella sezione precedente, mentre l'indice n riguarda la discretizzazione temporale. Si possono ora definire due parametri utili a riformulare l'equazione 3.23 in maniera più compatta; essi sono il numero di Courant (c) e il numero di diffusione adimensionato (d):

$$c = \frac{U\Delta t}{\Delta x}, \quad d = \frac{\Gamma \Delta t}{\rho \Delta x^2} \tag{3.24}$$

Sostituendo nello schema di calcolo del metodo di Eulero Implicito si ottiene:

$$\phi_i^{n+1} = \phi_i^n - \frac{c}{2}\left(\phi_{i+1}^{n+1} - \phi_{i-1}^{n+1}\right) + d\left(\phi_{i+1}^{n+1} - 2\phi_i^{n+1} + \phi_{i-1}^{n+1}\right) \tag{3.25}$$

da cui:

$$(1+2d)\phi_i^{n+1} + \left(\frac{c}{2} - d\right)\phi_{i+1}^{n+1} + \left(-\frac{c}{2} - d\right)\phi_{i-1}^{n+1} = \phi_i^n \tag{3.26}$$

Facendo riferimento alla nomenclatura utilizzata per la discretizzazione spaziale nella sezione 3.3 e riconducendosi alla forma canonica $a_P\phi_P = a_W\phi_W + a_E\phi_E + b$ (con $P = i$, $W = i-1$ e $E = i+1$) si ottiene:

$$\begin{aligned} a_P &= 1 + 2d \\ a_W &= d + \frac{c}{2} \\ a_E &= d - \frac{c}{2} \\ b &= \phi_i^n \end{aligned} \tag{3.27}$$

Di seguito si riporta lo schema di discretizzazione in forma matriciale:

$$\begin{bmatrix} a_P & a_E & 0 & 0 & \cdots & 0 \\ a_W & a_P & a_E & 0 & \cdots & 0 \\ 0 & a_W & a_P & a_E & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a_W & a_P & a_E \\ 0 & \cdots & 0 & 0 & a_W & a_P \end{bmatrix} \begin{bmatrix} \phi_1^{n+1} \\ \phi_2^{n+1} \\ \phi_3^{n+1} \\ \vdots \\ \phi_{N-1}^{n+1} \\ \phi_N^{n+1} \end{bmatrix} = \begin{bmatrix} \phi_1^n \\ \phi_2^n \\ \phi_3^n \\ \vdots \\ \phi_{N-1}^n \\ \phi_N^n \end{bmatrix} \tag{3.28}$$

dove N rappresenta il numero di celle in cui è stato suddiviso l'intero dominio computazionale.
Tale sistema può essere risolto mediante l'applicazione di metodi iterativi, come ad esempio il metodo di Gauss-Seidel, al fine di valutare il valore di ϕ nel tempo all'interno delle celle ottenute dalla discretizzazione spaziale.

3.5 Risoluzione del campo di moto

Prima di effettuare le simulazioni CFD finalizzate a modellare il processo di APR, è necessario risolvere il campo di moto all'interno del dominio computazionale. Le equazioni risolte sono presentate nelle sezioni precedenti (3.1), così come gli algoritmi che permettono di trasformarle da equazioni differenziali ad algebriche (3.2).
Si procede ora a descrivere il funzionamento di `porousSimpleFoam`, il *solver* utilizzato per risolvere il campo di moto e di pressione in presenza di un mezzo poroso. Esso implementa l'algoritmo SIMPLE (*Semi-Implicit Method for Pressure Linked Equations*), il quale richiede un'inizializzazione del campo di pressione per successivamente eseguire i seguenti step:

1. a partire dal campo di pressione, si risolve l'equazione di Navier-Stokes (3.2) per ottenere il campo di moto;

2. il campo di moto calcolato viene utilizzato per risolvere l'equazione di Poisson, ottenuta applicando la divergenza all'equazione di Navier-Stokes e sostituendo l'equazione di continuità (3.1):

$$\frac{\partial}{\partial x_i}\left(\frac{\partial p}{\partial x_i}\right) = -\frac{\partial}{\partial x_i}\left(\frac{\partial(\rho U_i U_j)}{\partial x_j}\right) \tag{3.29}$$

   In questo modo è possibile ottenere un campo di pressione aggiornato;

3. si effettua un confronto tra il profilo di pressione utilizzato nello step 1 con quello ottenuto nello step 2: se non si è raggiunta la convergenza, si esegue una nuova iterazione utilizzando il campo di pressione appena calcolato.

All'interno del *solver* implementato si tiene conto del mezzo poroso modellando il termine sorgente dell'equazione 3.2 mediante l'equazione di Darcy-Forchheimer:

$$\boldsymbol{S} = -\left(\mu \boldsymbol{d} + \frac{\rho|\boldsymbol{U}|}{2}\boldsymbol{f}\right)\boldsymbol{U} \tag{3.30}$$

dove $\boldsymbol{d}$ e $\boldsymbol{f}$ sono due tensori che rappresentano rispettivamente il coefficiente di Darcy $(1/m^2)$ e il coefficiente di Forchheimer $(1/m)$. Nello studio svolto si è ipotizzato

un regime di moto laminare, quindi una dipendenza lineare tra il gradiente di pressione e la velocità: il contributo dato dal termine di Forchheimer è stato quindi trascurato, riducendo l'equazione 3.30 alla forma espressa nell'equazione 3.3.

3.6 Solver implementati

Eseguita la simulazione con `porousSimpleFoam` per ottenere il campo di moto, quest'ultimo è stato utilizzato come *boundary condition* per risolvere delle simulazioni di advezione-dispersione-reazione. A tal fine sono stati implementati dei *solver* creati modificando `scalarTransportFoam`, un solver presente nel pacchetto base di `OpenFOAM` che permette di risolvere l'equazione di advezione e dispersione per uno scalare. Per quanto riguarda il termine di dispersione, `scalarTransportFoam` richiede, come parametro di input, il valore del coefficiente D dello scalare di cui si vuole studiare il trasporto. Nel caso in esame, il coefficiente di dispersione è richiesto per le specie chimiche coinvolte nel processo: il parametro diffusivo corrisponde quindi al valore di dispersione, espresso in m^2/s, che ognuna delle specie ha in un sistema diluito in acqua.
Tuttavia, `scalarTransportFoam` non tiene conto del termine sorgente dovuto allo sviluppo di reazioni chimiche all'interno del sistema: per questo motivo esso è stato modificato affinché l'equazione risolta durante le simulazioni fosse uguale alla 3.5.
I *solver* ottenuti applicando tali modifiche vengono di seguito presentati:

- il primo risolutore, da noi denominato `myScalarReactionFoamAPR`, in cui i termini sorgente di reazione delle equazioni risolte per ogni specie sono funzioni analitiche. Questo permette di simulare il processo di APR solo per lo scenario semplificato, in cui si considera lo sviluppo di due reazioni che coinvolgono complessivamente quattro specie chimiche.
  Il fine principale delle simulazioni eseguite utilizzando questo *solver* è quello di valutare il tempo di esecuzione richiesto, per poterlo confrontare successivamente con quello ottenuto dalle simulazioni di accoppiamento con i modelli *Neural* ODE;

- il secondo risolutore implementato, i cui codici sono consultabili nell'Appendice, da noi denominato `mytorchODEFoam`. Tale risolutore consente di modellare il termine sorgente S_n dell'equazione di advezione, dispersione e reazione (3.5) utilizzando i modelli di cinetiche *data-driven* ottenuti con l'implementazione delle NODEs (capitolo 2).
  Per rendere possibile l'accoppiamento tra simulazioni in `OpenFOAM`, il cui linguaggio di programmazione principale è `C++`, e i modelli neurali, allenati e costruiti utilizzando `PyTorch`, è stata utilizzata la libreria `libtorch`, che costituisce il *backend* nativo in `C++` su cui `PyTorch` si appoggia.

Di seguito si riporta la forma dell'equazione implementata all'interno del *solver*:

$$\frac{\partial C_n}{\partial t} + \frac{\partial (U_i C_n)}{\partial x_i} = D_n \frac{\partial^2 C_n}{\partial x_i^2} + NODE(\boldsymbol{C}, T_m) \tag{3.31}$$

dove $\boldsymbol{C}$ è il vettore contenente le concentrazioni di tutte le specie chimiche coinvolte nel processo e T_m è la temperatura di Arrhenius. Tutte le variabili in input al modello vengono normalizzate all'interno del *solver* analogamente a come è stato fatto per il *training* dei modelli.
Il modello può essere considerato come una *"black-box"* che permette di approssimare le cinetiche delle reazioni coinvolte senza l'utilizzo di funzioni analitiche.

Nel capitolo a seguire si riportano i passaggi eseguiti per impostare le simulazioni CFD in accoppiamento con i modelli *data-driven* e i risultati ottenuti.

4 Risultati: CFD

In questo capitolo viene descritta l'impostazione delle simulazioni eseguite su `OpenFOAM` (sezione 4.1), presentando in maniera dettagliata la costruzione della geometria del reattore modellato, le condizioni al contorno necessarie a simulare il processo e i parametri richiesti dal *solver* per risolvere le equazioni di governo. Nella sezione 4.2 vengono riportati i risultati ottenuti dalle simulazioni svolte.

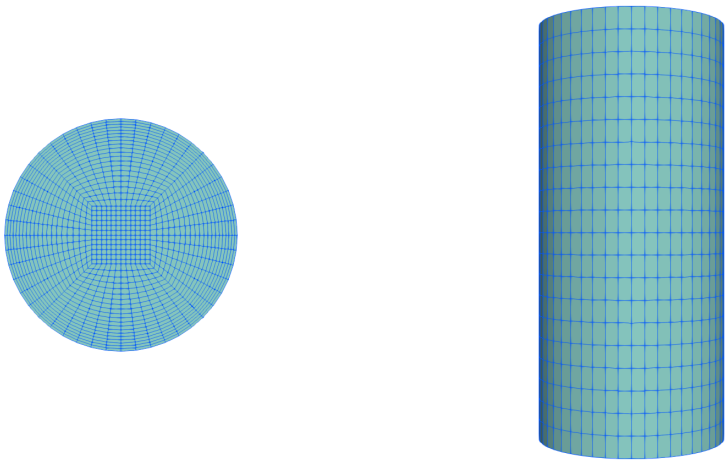
4.1 Impostazione delle simulazioni

Lo scopo principale delle simulazioni CFD svolte in questo progetto di tesi è di verificare se sia possibile implementare dei modelli cinetici ottenuti a partire dal *training* di *Neural* ODEs utilizzando dati sperimentali. In particolare, il sistema simulato è costituito da un *Plug-Flow Reactor* (PFR) in scala di laboratorio all'interno del quale avviene il processo catalitico di APR. Le dimensioni del reattore sono riportate nella seguente tabella:

Tabella 4.1: Dimensioni caratteristiche del PFR modellato.

Dimensione	Unità di misura	Valore
Diametro	mm	9.00
Lunghezza	mm	20.00

Utilizzando il comando `blockMesh` abbiamo ottenuto una geometria di dimensioni pari a quelle appena riportate; la *mesh* è costituita da 22080 celle, ottenendo una griglia computazionale piuttosto fitta, che garantisce la *grid independence* del campo di moto. In Figura 4.1 si può osservare la geometria ottenuta da due prospettive differenti. La corrente in ingresso al reattore, facendo riferimento alla Figura 4.1b, viene alimentata dal basso e si muove lungo la direzione assiale del PFR, uscendo dall'estremità superiore.



(a) Vista della *mesh* lungo l'asse z **(b)** Vista laterale della *mesh*

Figura 4.1: Rappresentazione grafica della geometria ottenuta da comando `blockMesh`.

Il catalizzatore solido, a base di Pt su supporto di carboni attivi, è stato modellato utilizzando il comando `topoSet`, il quale richiede come parametri di *input* i tensori dei coefficienti di Darcy e di Forchheimer (equazione 3.30). Il primo tensore è costituito da tre valori, calcolati come il reciproco della permeabilità K del mezzo poroso. Avendo ipotizzato che il solido sia isotropo, tali valori sono uguali lungo le tre direzioni. Il secondo tensore, invece, è posto pari a zero per l'ipotesi di regime laminare e dipendenza lineare tra le perdite di carico e la velocità del fluido. Nella tabella sottostante si riportano i valori dei parametri inseriti:

Tabella 4.2: Valori dei coefficienti di Darcy e Forchheimer utilizzati per la modellazione del solido poroso.

Tensore	Unità di misura	Direzione x	Direzione y	Direzione z
d	$1/m^2$	10^9	10^9	10^9
f	$1/m$	0	0	0

Una volta generata la *mesh* ed aver inserito il catalizzatore nel reattore, è stata effettuata una prima simulazione CFD utilizzando il *solver* `porousSimpleFoam` (sezione 3.5) per calcolare il campo di moto all'interno del dominio computazionale. A questo scopo si è impostata come *boundary condition* una perdita di carico espressa come $\Delta p/\rho$ pari a $5.2 \cdot 10^{-3}\ m^2/s^2$, necessaria ad ottenere una velocità di inlet al reattore di $2.62 \cdot 10^{-4}\ m/s$.
In Figura 4.2 è riportata una rappresentazione grafica del campo di moto ottenuto

all'interno del reattore ed il grafico del profilo di velocità assiale lungo la coordinata radiale.

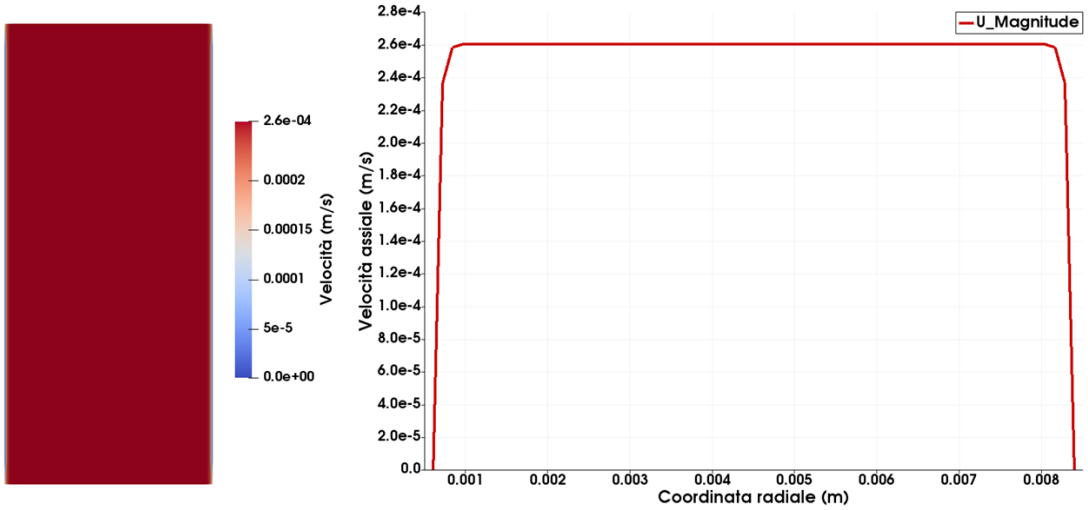


Figura 4.2: Sezione longitudinale del reattore simulato per la visualizzazione del campo di moto ottenuto e grafico del profilo radiale della velocità.

Il profilo di velocità è stato successivamente utilizzato come *boundary condition*, insieme alle concentrazioni delle specie chimiche in ingresso, per le simulazioni finalizzate a modellare il processo di APR. I valori delle concentrazioni inseriti come condizioni iniziali per la simulazione vengono riportati nella Tabella 4.3:

Tabella 4.3: Concentrazioni iniziali delle specie chimiche coinvolte nel processo di APR.

Specie chimica	Concentrazione (mol/m^3)
Glicerolo	529
Anidride carbonica	0
Idrogeno	0
Propilen-glicole	0
Etilen-glicole	0
Propanolo	0

Per la risoluzione dell'equazione di advezione, dispersione e reazione (3.5) per ogni specie chimica coinvolta è necessario fornire come parametro fisico il valore del coefficiente di dispersione D, il cui calcolo è stato fatto partendo dai valori di diffusione molecolare delle specie. Il rapporto tra dispersione e diffusione, infatti, è

esprimibile come una funzione del numero adimensionato di Péclet [38]:

$$Pe = \frac{ud_g}{D_m} \tag{4.1}$$

dove u è la velocità interstiziale del fluido, d_g è una lunghezza caratteristica del sistema, che in questo caso è stata presa pari al diametro equivalente delle particelle di solido, e D_m è il coefficiente di diffusione molecolare. Analizzando l'andamento del rapporto D/D_m al variare del numero di Péclet per il sistema studiato si ottiene il seguente comportamento:

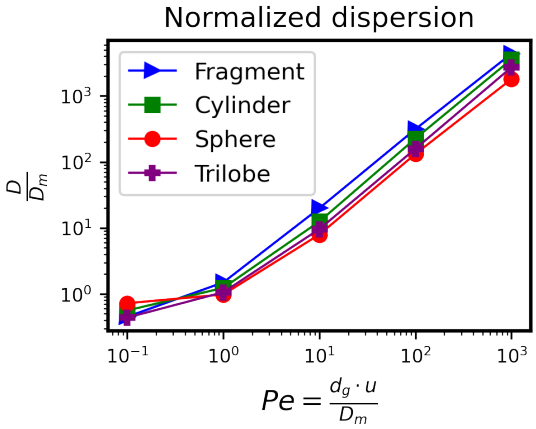


Figura 4.3: Esempio dell'andamento del rapporto tra dispersione e diffusione molecolare in funzione del numero di Péclet per diverse morfologie di pellet catalitici - immagine adattata da [39].

Per valori del numero di Péclet sufficientemente elevati, il rapporto assume una dipendenza a legge di potenza. Per calcolare le dispersioni delle sei specie chimiche presenti nel sistema si è valutato per ognuna il valore del Péclet, inserendo i seguenti parametri all'interno della formula 4.1:

Tabella 4.4: Valori dei parametri per il calcolo del numero di Péclet per le specie coinvolte.

Parametro	Unità di misura	Valore
d_g	mm	1.40
u	m/s	$2.62 \cdot 10^{-4}$
$D_{m,Gli}$	m^2/s	$1.69 \cdot 10^{-9}$
D_{m,CO_2}	m^2/s	$3.13 \cdot 10^{-9}$
D_{m,H_2}	m^2/s	$4.00 \cdot 10^{-9}$
$D_{m,PG}$	m^2/s	$1.77 \cdot 10^{-9}$
$D_{m,EG}$	m^2/s	$1.10 \cdot 10^{-9}$
$D_{m,1P}$	m^2/s	$0.87 \cdot 10^{-9}$

Considerando le particelle di solido come delle sfere, si è ottenuto dal grafico in Figura 4.3 il rapporto tra dispersione e diffusione per ogni specie, partendo dai numeri di Péclet calcolati; moltiplicando infine per la diffusione molecolare si ricava la dispersione. I valori del numero di Péclet e della dispersione relativi alle specie considerate sono i seguenti:

Tabella 4.5: Valori del numero di Péclet e delle dispersioni ottenuti per le sei specie.

Specie chimica	Péclet	$D\ (m^2/s)$
Glicerolo	218	$5.24 \cdot 10^{-7}$
Anidride carbonica	118	$5.63 \cdot 10^{-7}$
Idrogeno	92	$4.40 \cdot 10^{-7}$
Propilen-glicole	208	$5.66 \cdot 10^{-7}$
Etilen-glicole	334	$6.05 \cdot 10^{-7}$
Propanolo	423	$6.18 \cdot 10^{-7}$

Le dispersioni calcolate sono state inserite nel file `transportProperties` contenuto nella *case* della simulazione. Nello stesso file, inoltre, è riportato il valore della temperatura assoluta T del sistema, pari a 513 K.
Un ultimo aspetto importante da considerare prima di eseguire la simulazione è la coerenza delle unità di misura dei *reaction rates* ottenuti implementando le *Neural* ODEs. L'impostazione dei *training* dei modelli sviluppati fornisce, come *output* della predizione, le cinetiche di reazione in $mol/m_L^3 s$, dove m_L^3 si riferisce al volume di liquido del reattore utilizzato per la raccolta dei dati. Per poter inserire tali modelli nell'equazione di advezione, dispersione e reazione come indicato in 3.31, è necessario correggere i *reaction rates* predetti dalle NODEs utilizzando una costante moltiplicativa che permetta di ottenere le cinetiche riferite al volume di reattore simulato. La costante c ha la seguente forma:

$$c = \frac{V_L \rho_{cat}(1 - \epsilon_0)}{m_{cat}} \tag{4.2}$$

dove:

V_L è il volume di liquido nel reattore utilizzato per la raccolta dei dati sperimentali (m^3);

ρ_{cat} è la densità del catalizzatore (kg/m^3);

ϵ_0 è la porosità del letto catalitico;

m_{cat} è la massa di catalizzatore utilizzato nelle prove sperimentali (kg).

I valori dei parametri sopra indicati vengono riportati nella seguente tabella:

Tabella 4.6: Parametri fisici utilizzati per il calcolo della costante moltiplicativa c.

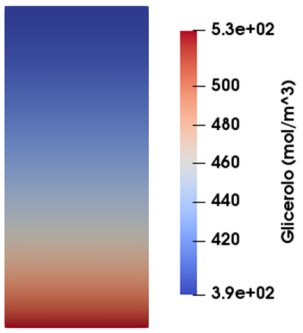
Parametro	Unità di misura	Valore
V_L	m^3	$120 \cdot 10^{-6}$
ρ_{cat}	kg/m^3	859
ϵ_0	-	0.4
m_{cat}	kg	$0.3 \cdot 10^{-3}$

Tale costante, pari a 206.06 m_L^3/m_{PFR}^3, è fornita come *input* alla simulazione ed è riportata anch'essa nel file `transportProperties`.

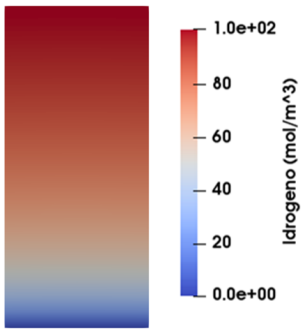
Si procede ora a presentare i risultati ottenuti con le simulazioni CFD svolte utilizzando il *solver* `mytorchODEFoam`, che accoppia i modelli di cinetiche *data-driven* a simulazioni di fluidodinamica computazionale al fine di risolvere l'equazione di trasporto delle specie chimiche coinvolte nel processo simulato.

4.2 Risultati ottenuti

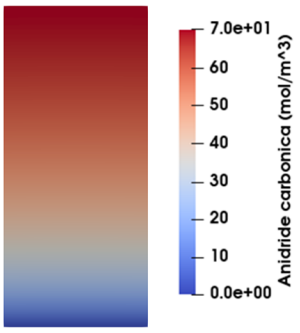
Le simulazioni effettuate riproducono il comportamento del sistema in un intervallo di tempo pari a 650 s, necessario a raggiungere lo stato stazionario all'interno del PFR.
Di seguito si riportano sei sezioni longitudinali del reattore, ognuna delle quali rappresenta il profilo di concentrazione delle specie chimiche considerate:



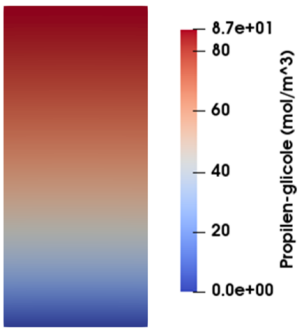
(a) Glicerolo



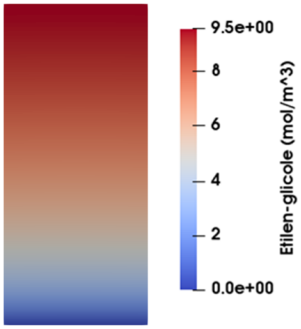
(b) Idrogeno



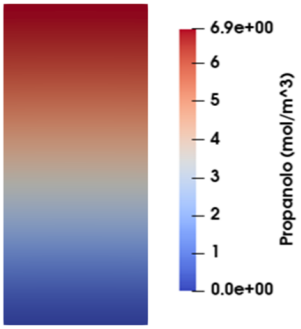
(c) Anidride carbonica



(d) Propilen-glicole



(e) Etilen-glicole



(f) Propanolo

Figura 4.4: Profili di concentrazione delle sei specie ottenuti lungo la coordinata assiale.

Come si può osservare dai *contour plots* riportati in Figura 4.4, il glicerolo diminuisce la propria concentrazione spostandosi all'interno del reattore, in quanto reagente del processo; tutte le altre specie, invece, hanno una concentrazione nulla in ingresso al PFR che aumenta grazie allo sviluppo delle quattro reazioni chimiche considerate nel modello cinetico *data-driven*.
Questi andamenti sono coerenti con quelli predetti dalle *Neural* ODEs allenate sui dati sperimentali, riportati nei grafici 2.12-2.17; questo primo confronto ha permesso di validare i risultati ottenuti con l'accoppiamento tra il modello cinetico e la simulazione CFD.

Per verificare che le variazioni nel tempo delle sei concentrazioni avessero un comportamento fisicamente accettabile, abbiamo riportato l'andamento delle specie lungo la direzione assiale del reattore, ottenendo il grafico riportato di seguito:

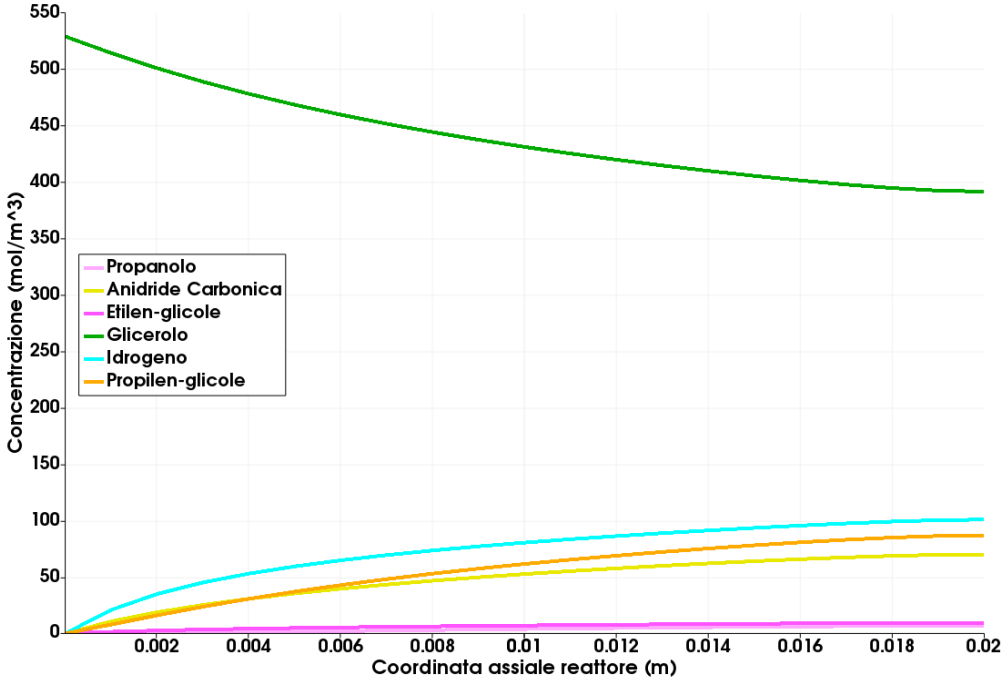


Figura 4.5: Andamento delle concentrazioni lungo la coordinata assiale del reattore simulato.

I profili di concentrazione ottenuti sono fisicamente accettabili: in altre parole, non presentano comportamenti anomali e l'andamento è analogo a quello ottenuto durante i *training* con dati sperimentali. Inoltre, si osserva come l'ordine di grandezza delle concentrazioni di ogni specie rispetti quello dei valori sperimentali utilizzati per sviluppare i modelli cinetici (Tabella 2.2).

Osservando quindi i risultati ottenuti dalle simulazioni CFD in accoppiamento con i modelli cinetici sviluppati, si può affermare che, a livello metodologico, l'implementazione di modelli ottimizzati a partire da dati sperimentali al fine di simulare il termine sorgente delle equazioni di trasporto è possibile e permette di ottenere risultati soddisfacenti, in linea con i valori ottenuti mediante le prove di laboratorio svolte.
Un ultimo importante aspetto da tenere in considerazione quando si parla di modellazione CFD è il tempo computazionale richiesto per eseguire una simulazione. A questo scopo, si è confrontato il tempo impiegato per simulare il processo di APR utilizzando il *solver* `mytorchODEFoam` con il tempo richiesto per eseguire una simulazione analoga implementando il *solver* `myScalarReactionFoamAPR`, dove il termine sorgente dell'equazione di advezione, dispersione e reazione è dato da cinetiche analitiche.
I risultati ottenuti nel primo caso sono significativamente migliori di quelli acquisiti con il secondo *solver*; tuttavia, il costo computazionale dato dall'accoppiamento

delle NODEs con le simulazioni CFD richiede tempi di simulazione maggiori. Nella tabella seguente si riportano i tempi impiegati nei due casi differenti:

Tabella 4.7: Tempi computazionali richiesti dai *solver* implementati per la simulazione del processo di APR.

Solver	Tempo di simulazione richiesto (s)
`mytorchODEFoam`	214.52
`myScalarReactionFoamAPR`	47.95

Come già anticipato, la simulazione che implementa i modelli cinetici *data-driven* è computazionalmente più onerosa, come si evince dai valori riportati in Tabella 4.7. Tuttavia, i risultati ottenuti sono marcatamente migliori rispetto ai profili di concentrazione che si ottengono utilizzando modelli analitici per le cinetiche di reazione.

5 Conclusioni

L'obiettivo del presente lavoro è stato di verificare l'accuratezza delle NODEs quando utilizzate come surrogati di modelli cinetici e la fattibilità dell'accoppiamento tra il modello ottenuto e le simulazioni CFD. A tale scopo, il *workflow* sviluppato presenta i seguenti passaggi principali:

- applicazione delle NODEs, con vincoli cinetici e termodinamici imposti all'interno dell'architettura, finalizzata a costruire un modello cinetico *data-driven* in grado di predire i *reaction rates* del processo di APR del glicerolo, apprendendo da un set di dati raccolti sperimentalmente;

- validazione del modello ottimizzato, eseguita andando a valutare la bontà di approssimazione dei dati sperimentali ottenuta dalle predizioni;

- setup di una simulazione in `OpenFOAM` contenente le informazioni necessarie a simulare il processo di APR all'interno di un PFR (*boundary conditions*, metodi di discretizzazione spaziale e temporale, proprietà fisiche del fluido e del mezzo poroso);

- implementazione del *solver* `mytorchODEFoam`, che risolve l'equazione di advezione, dispersione e reazione per le specie chimiche coinvolte nel processo, modellando il termine sorgente dato dalle reazioni chiamando il modello ottenuto dall'allenamento della NODE;

- esecuzione delle simulazioni, con successiva estrazione e valutazione dei risultati ottenuti.

La prima parte del *workflow* ha consentito di ottenere un modello di cinetica basato sui dati sperimentali, il quale approssima i profili di concentrazione con un errore significativamente minore rispetto a quello ottenuto modellando le cinetiche con funzioni analitiche tradizionali.
Dalla seconda parte del lavoro abbiamo ottenuto buoni risultati dalle simulazioni CFD: i *contour plots* acquisiti, infatti, mostrano andamenti delle concentrazioni lungo la coordinata assiale del reattore in linea con le predizioni ottenute dalla fase di *training* delle NODEs.

Dai risultati ottenuti da questo progetto di tesi si può concludere che:

- l'implementazione di modelli di *deep learning* come le NODEs al fine di approssimare le cinetiche di reazioni coinvolte in processi complessi come l'APR è possibile e consente di ottenere delle predizioni accurate degli andamenti delle concentrazioni registrate durante le prove sperimentali;

- l'imposizione dei vincoli cinetici e termodinamici nell'architettura dei modelli implementati ha permesso di effettuare con successo i *training*, nonostante la dimensione ridotta del *dataset* sperimentale utilizzato;

- l'accoppiamento dei suddetti modelli con simulazioni di CFD è realizzabile e permette di acquisire risultati qualitativamente migliori rispetto a quelli ottenibili modellando le reazioni chimiche mediante funzioni analitiche, a fronte di un aumento del tempo di simulazione richiesto.

Per eventuali progetti futuri, potrebbe essere interessante applicare i modelli che implementano le NODEs a sistemi reattivi ancora più complessi, quali i processi di combustione, che coinvolgono un numero maggiore di reazioni e di specie chimiche. Tali studi permetterebbero di valutare il limite di complessità massimo tale per cui i modelli cinetici *data-driven* sono in grado di approssimare i *reaction rates* in maniera accurata.
Per quanto riguarda l'implementazione di tali modelli nel campo della CFD, invece, si potrebbe lavorare sul miglioramento dell'efficienza computazionale dell'accoppiamento al fine di ridurre il tempo di simulazione richiesto.

Appendice

In questa sezione si riportano i principali codici implementati nello studio svolto.

Modelli di rete implementati

Per il modello a quattro specie lo script utilizzato è il seguente:

```
from torchdyn.core import NeuralODE
import torch

class KineticConstrainedNN_henry(torch.nn.Module):
    def __init__(self, n_hidden=32):
        super(KineticConstrainedNN_henry, self).__init__()
        self.layer_1 = torch.nn.Linear(4, n_hidden)
        self.layer_T = torch.nn.Linear(1, n_hidden)
        self.layer_Tout = torch.nn.Linear(n_hidden, 2)
        self.layer_out = torch.nn.Linear(n_hidden, 2)
        self.layer_H = torch.nn.Linear(1, n_hidden)
        self.layer_Hout = torch.nn.Linear(n_hidden, 1)

        Vliq = 120e-6  # Volume of liquid phase in m^3
        Vgas = 180e-6  # Volume of gas phase in m^3

        self.coef_ = torch.tensor([[-1, -1],
                                   [7*Vliq/Vgas, -1*Vliq/Vgas],
                                   [3*Vliq/Vgas, 0],
                                   [0, 1]]).T.float()


    def forward(self, x):
        concentrations= x[:, :-1]
        temperature=  x[:, -1:]
        T = 1/(temperature*8.31/10000+1/(273.15 + 300))/500
```

```
        R = 8.314  # J/(mol*K)

        c_C3H8O3  = concentrations[:, 0:1]
        c_H2   = concentrations[:, 1:2]
        c_CO2 = concentrations[:, 2:3]
        c_C3H8O2  = concentrations[:, 3:4]

        xh = torch.nn.functional.silu(self.layer_H(temperature))
        xh = torch.exp(self.layer_Hout(xh)).clamp(max=5)
        concentrations_mod = concentrations.clone()
        concentrations_mod[:, 1:2] = c_H2 * xh * R * T

        x_nn = torch.nn.functional.silu(self.layer_1(concentrations_mod))
        x_nn = torch.nn.functional.silu(self.layer_out(x_nn))

        xt = torch.nn.functional.silu(self.layer_T(temperature))
        xt = torch.exp(self.layer_Tout(xt)).clamp(max=5)

        x1 = c_C3H8O3*x_nn[:, 0:1]
        x2 = c_C3H8O3*concentrations_mod[:, 1:2]*x_nn[:, 1:2]

        x_rates = torch.hstack([x1, x2])

        x_rates = x_rates * xt

        out_rates = torch.matmul(x_rates, self.coef_)

        zeros_pad = torch.zeros_like(out_rates[:, :1])
        out = torch.hstack([out_rates, zeros_pad])

        return out

def get_KCNODE_APR_model_henry(solver='dopri5'):
    model_nn = KineticConstrainedNN_henry()
    return NeuralODE(model_nn, sensitivity='autograd',
                     solver=solver, order=1, return_t_eval=False)
```

Per quello a sei specie:

```
from torchdyn.core import NeuralODE
import torch

class KineticConstrainedNN_henry(torch.nn.Module):
    def __init__(self, n_hidden=64):
```

```
        super(KineticConstrainedNN_henry, self).__init__()
        self.layer_1 = torch.nn.Linear(6, n_hidden)
        self.layer_T = torch.nn.Linear(1, n_hidden)
        self.layer_Tout = torch.nn.Linear(n_hidden, 4)
        self.layer_out = torch.nn.Linear(n_hidden, 4)
        self.layer_H = torch.nn.Linear(1, n_hidden)
        self.layer_Hout = torch.nn.Linear(n_hidden, 1)

        Vl = 120e-6  # Volume of liquid phase in m^3
        Vg = 180e-6  # Volume of gas phase in m^3

        self.coef_ = torch.tensor([[-1, -1, -1, 0],
                                   [7*Vl/Vg,-1*Vl/Vg,2*Vl/Vg,-1*Vl/Vg],
                                   [3*Vl/Vg, 0, 1*Vl/Vg, 0],
                                   [0, 1, 0, -1],
                                   [0, 0, 1, 0],
                                   [0, 0, 0, 1]]).T.float()
    def forward(self, x):
        concentrations= x[:, :-1]
        temperature=  x[:, -1:]
        R = 8.314  # J/(mol*K)
        T = 1/(temperature*8.31/10000+1/(273.15 + 300))/500

        c_C3H8O3  = concentrations[:, 0:1]
        c_H2   = concentrations[:, 1:2]
        c_CO2 = concentrations[:, 2:3]
        c_C3H8O2  = concentrations[:, 3:4]
        c_ethg = concentrations[:, 4:5]
        c_prop = concentrations[:, 5:6]

        xh = torch.nn.functional.silu(self.layer_H(temperature))
        xh = torch.exp(self.layer_Hout(xh)).clamp(max=5)
        concentrations_mod = concentrations.clone()
        concentrations_mod[:, 1:2] = c_H2 * xh * R * T

        x_nn = torch.nn.functional.silu(self.layer_1(concentrations_mod))
        x_nn = torch.nn.functional.silu(self.layer_out(x_nn))

        xt = torch.nn.functional.silu(self.layer_T(temperature))
        xt = torch.exp(self.layer_Tout(xt)).clamp(max=5)

        x1 = c_C3H8O3*x_nn[:, 0:1]
        x2 = c_C3H8O3*concentrations_mod[:, 1:2]*x_nn[:, 1:2]
        x3 = c_C3H8O3*x_nn[:, 2:3]
        x4 = c_C3H8O2*concentrations_mod[:, 1:2]*x_nn[:, 3:4]
```

```
51
52          x_rates = torch.hstack([x1, x2, x3, x4])
53
54          x_rates = x_rates * xt
55
56          out_rates = torch.matmul(x_rates, self.coef_)
57
58          zeros_pad = torch.zeros_like(out_rates[:, :1])
59          out = torch.hstack([out_rates, zeros_pad])
60
61          return out
62
63  def get_KCNODE_APR_model_henry(solver='dopri5'):
64      model_nn = KineticConstrainedNN_henry()
65      return NeuralODE(model_nn, sensitivity='autograd',
66                       solver=solver, order=1, return_t_eval=False)
```

Codici del solver OpenFOAM/torchlib

Di seguito vengono riportati i due script principali contenuti all'interno del solver di OpenFOAM implementato per l'accoppiamento del modello ottenuto nel caso con quattro reazioni e sei specie con le simulazioni di CFD. Il primo codice riportato di seguito è `mytorchODEFoam_scaled.C`:

```
1  /*---------------------------------------------------------------------------*\
2    =========                 |
3    \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
4     \\    /   O peration     |
5      \\  /    A nd           | Copyright (C) The OpenFOAM Foundation, Ltd.
6       \\/     M anipulation  |
7  -------------------------------------------------------------------------------
8
9    Modifications and additional contributions:
10
11   MultiForm Group, University of Nottingham
12   Copyright (C) Matteo Icardi and collaborators
13   Contributions to this file are licensed under the same GPLv3 terms as
14   OpenFOAM.
15
16   This work is based on OpenFOAM, with substantial portions copied,
17   modified, or extended under the GNU General Public License (GPLv3).
18   Please refer to the original OpenFOAM copyright notice for the base
19   framework, and to MultiForm Group for new additions or modifications.
```

```
20    For the full terms of
21    this license, see <http://www.gnu.org/licenses/>.
22
23  Application
24      volAPRFoam
25
26  Description
27      Solves the steady or transient reactive transport equation for
28      two volumetric reactions.
29      Reaction occurs everywhere (no zones).
30
31
32  \*---------------------------------------------------------------------------*/
33  /*---------------------------------------------------------------------------*\
34    Application
35        volAPRFoam
36
37    Description
38        Transient reactive transport for 4 species using a TorchScript RHS
39        (no time arg).
40        NN input per cell: [Cg, Ch, Cc, Cp, Tarr];
41        output: [Sg, Sh, Sc, Sp, 0].
42  \*---------------------------------------------------------------------------*/
43
44  #include "fvCFD.H"
45  #include "pimpleControl.H"
46  #include "torchWrapper.H"
47
48  int main(int argc, char *argv[])
49  {
50      #include "postProcess.H"
51      #include "setRootCaseLists.H"
52      #include "createTime.H"
53      #include "createMesh.H"
54
55      pimpleControl pimple(mesh);
56
57      #include "createFields.H"
58
59      Info<<"\nCalculating scalar transport with"
60      "NN reaction source\n"<<endl;
61
62      #include "CourantNo.H"
63
64      //Load the TorchScript model once: expects already-preprocessed Tarr
```

```
    loadModel("myModel.pt");

    // If you need a flag to turn reactions on/off globally:
    //scalar flagReactive = 1.0;

    while (pimple.loop(runTime))
    {
        Info<< "\n\nTime = " << runTime.timeName() << nl << endl;

        while (pimple.loop())
        {
            // Compute NN sources with current fields
            auto S = evaluateTorchSources(Cg, Ch, Cc, Cp, Ce, C1p, Tarr);

            // --- Glycerol ---
            while (pimple.correctNonOrthogonal())
            {
                fvScalarMatrix CgEqn
                (
                    fvm::ddt(Cg)
                  + fvm::div(phi, Cg)
                  - fvm::laplacian(Dg*(1 + (flagReactive*(DcDf-1))), Cg)
                );

                // If NN returns production (+), subtract to consume
                (adjust sign as needed)
                CgEqn += fvm::Su(-rho.value()*S[0], Cg);

                CgEqn.relax();
                CgEqn.solve();
            }

            // Optionally refresh sources (cheap vs solve) after
            each species update
            S = evaluateTorchSources(Cg, Ch, Cc, Cp, Ce, C1p, Tarr);

            // --- Hydrogen ---
            while (pimple.correctNonOrthogonal())
            {
                fvScalarMatrix ChEqn
                (
                    fvm::ddt(Ch)
                  + fvm::div(phi, Ch)
                  - fvm::laplacian(Dh*(1 + (flagReactive*(DcDf-1))), Ch)
                );
```

```
110
111             ChEqn += fvm::Su(-rho.value()*S[1], Ch);
112
113             ChEqn.relax();
114             ChEqn.solve();
115         }
116
117         S = evaluateTorchSources(Cg, Ch, Cc, Cp, Ce, C1p, Tarr);
118
119         // --- Carbon dioxide ---
120         while (pimple.correctNonOrthogonal())
121         {
122             fvScalarMatrix CcEqn
123             (
124                 fvm::ddt(Cc)
125               + fvm::div(phi, Cc)
126               - fvm::laplacian(Dc*(1 + (flagReactive*(DcDf-1))), Cc)
127             );
128
129             CcEqn += fvm::Su(-rho.value()*S[2], Cc);
130
131             CcEqn.relax();
132             CcEqn.solve();
133         }
134
135         S = evaluateTorchSources(Cg, Ch, Cc, Cp, Ce, C1p, Tarr);
136
137         // --- Propylene glycol ---
138         while (pimple.correctNonOrthogonal())
139         {
140             fvScalarMatrix CpEqn
141             (
142                 fvm::ddt(Cp)
143               + fvm::div(phi, Cp)
144               - fvm::laplacian(Dp*(1 + (flagReactive*(DcDf-1))), Cp)
145             );
146
147             CpEqn += fvm::Su(-rho.value()*S[3], Cp);
148
149             CpEqn.relax();
150             CpEqn.solve();
151         }
152
153         S = evaluateTorchSources(Cg, Ch, Cc, Cp, Ce, C1p, Tarr);
154
```

```
155             // --- Ethylene glycol ---
156             while (pimple.correctNonOrthogonal())
157             {
158                 fvScalarMatrix CeEqn
159                 (
160                     fvm::ddt(Ce)
161                   + fvm::div(phi, Ce)
162                   - fvm::laplacian(De*(1 + (flagReactive*(DcDf-1))), Ce)
163                 );
164 
165                 CeEqn += fvm::Su(-rho.value()*S[4], Ce);
166 
167                 CeEqn.relax();
168                 CeEqn.solve();
169             }
170 
171             S = evaluateTorchSources(Cg, Ch, Cc, Cp, Ce, C1p, Tarr);
172 
173             // --- 1-propanol ---
174             while (pimple.correctNonOrthogonal())
175             {
176                 fvScalarMatrix C1pEqn
177                 (
178                     fvm::ddt(C1p)
179                   + fvm::div(phi, C1p)
180                   - fvm::laplacian(D1p*(1+(flagReactive*(DcDf-1))), C1p)
181                 );
182 
183                 C1pEqn += fvm::Su(-rho.value()*S[5], C1p);
184 
185                 C1pEqn.relax();
186                 C1pEqn.solve();
187             }
188         }
189 
190         runTime.write();
191     }
192 
193     Info<< "End\n" << endl;
194     Info<< "ExecutionTime = "<<runTime.elapsedCpuTime()<<"s"<< endl;
195 
196     return 0;
197 }
```

Il secondo script presentato è torchWrapper.C:

```
#include "volFields.H"
#include "fvMesh.H"
#include "torchWrapper.H"

// Undefine OpenFOAM macros that conflict with PyTorch
#ifdef TypeName
#undef TypeName
#endif
#ifdef word
#undef word
#endif

#include <torch/script.h>
#include <vector>
#include <array>

namespace
{
    torch::jit::script::Module model;
    bool modelLoaded = false;
    constexpr float SC = 500.0f;
    // concentration scale used during training
    constexpr float ST = 10800.0f;
    // time scale in seconds used during training
}

void loadModel(const std::string& path)
{
    try {
        model = torch::jit::load(path);
        model.eval();
        modelLoaded = true;
    } catch (const c10::Error& e) {
        FatalErrorInFunction
            << "Error loading Torch model: " << e.what()
            << Foam::exit(Foam::FatalError);
    }
}

// Pack (Cg, Ch, Cc, Cp, Tarr) into [nCells,5]. Forward through model.
// Accept output [nCells,4] or [nCells,5] and ignore the 5th (dT/dt)
// if present.
std::array<Foam::volScalarField,6>
evaluateTorchSources
```

```
(
    const Foam::volScalarField& Cg,
    const Foam::volScalarField& Ch,
    const Foam::volScalarField& Cc,
    const Foam::volScalarField& Cp,
    const Foam::volScalarField& Ce,
    const Foam::volScalarField& C1p,
    const Foam::volScalarField& Tarr
)
{
    if (!modelLoaded)
    {
        FatalErrorInFunction
            << "Torch model not loaded. Call loadModel() first."
            << Foam::exit(Foam::FatalError);
    }

    const Foam::label nCells = Cg.size();

    // Sanity: all fields must share size/mesh
    if (Ch.size() != nCells || Cc.size() != nCells || Cp.size() != nCells
    || Tarr.size() != nCells || Ce.size() != nCells ||
    C1p.size() != nCells)
    {
        FatalErrorInFunction
            << "Field size mismatch among Cg/Ch/Cc/Cp/Ce/C1p/Tarr."
            << Foam::exit(Foam::FatalError);
    }


    // Build input X: [nCells, 5] = [Cg, Ch, Cc, Cp, Ce, C1p, Tarr]
    torch::NoGradGuard no_grad;
    auto X = torch::empty({static_cast<long long>(nCells), 7LL},
    torch::kFloat32).contiguous();
    auto accX = X.accessor<float,2>();
    const float invSC = 1.0f / SC;
    for (Foam::label i = 0; i < nCells; ++i)
    {
        accX[i][0] = static_cast<float>(Cg[i])* invSC;
        accX[i][1] = static_cast<float>(Ch[i])* invSC;
        accX[i][2] = static_cast<float>(Cc[i])* invSC;
        accX[i][3] = static_cast<float>(Cp[i])* invSC;
        accX[i][4] = static_cast<float>(Ce[i])* invSC;
        accX[i][5] = static_cast<float>(C1p[i])* invSC;
        accX[i][6] = static_cast<float>(Tarr[i]);
```

```
90      }
91
92      // Forward pass
93      torch::Tensor Y;
94      try {
95          Y = model.forward({X}).toTensor();
96      } catch (const c10::Error& e) {
97          FatalErrorInFunction
98              << "Torch model forward failed: " << e.what()
99              << Foam::exit(Foam::FatalError);
100     }
101
102     // Expect [nCells,4] or [nCells,5] (5th = dT/dt ~ 0, ignored)
103     if (Y.dim() != 2 || Y.size(0) != nCells || (Y.size(1) != 6
104     && Y.size(1) != 7))
105     {
106         FatalErrorInFunction
107             << "Unexpected Torch model output shape. Expected [nCells,4]
108             or [nCells,5], got ["
109             << (Y.dim() > 0 ? Y.size(0) : -1) << ","
110             << (Y.dim() > 1 ? Y.size(1) : -1) << "]."
111             << Foam::exit(Foam::FatalError);
112     }
113
114     if (!Y.is_contiguous()) Y = Y.contiguous();
115     const float outScale = SC / ST;
116     Y = Y * outScale;
117     // Output fields with dimensions [C]/[t], reuse boundary types
118     std::array<Foam::volScalarField,6> S = {
119         Foam::volScalarField
120         (
121             Foam::IOobject("Sg", Cg.time().timeName(), Cg.db(),
122                            Foam::IOobject::NO_READ,
123                            Foam::IOobject::NO_WRITE),
124             Cg.mesh(),
125             Foam::dimensionedScalar("zero",
126             Cg.dimensions()/Foam::dimTime, 0.0),
127             Cg.boundaryField().types()
128         ),
129         Foam::volScalarField
130         (
131             Foam::IOobject("Sh", Ch.time().timeName(), Ch.db(),
132                            Foam::IOobject::NO_READ,
133                            Foam::IOobject::NO_WRITE),
134             Ch.mesh(),
```

```
135             Foam::dimensionedScalar("zero",
136             Ch.dimensions()/Foam::dimTime, 0.0),
137             Ch.boundaryField().types()
138         ),
139         Foam::volScalarField
140         (
141             Foam::IOobject("Sc", Cc.time().timeName(), Cc.db(),
142                            Foam::IOobject::NO_READ,
143                            Foam::IOobject::NO_WRITE),
144             Cc.mesh(),
145             Foam::dimensionedScalar("zero",
146             Cc.dimensions()/Foam::dimTime, 0.0),
147             Cc.boundaryField().types()
148         ),
149         Foam::volScalarField
150         (
151             Foam::IOobject("Sp", Cp.time().timeName(), Cp.db(),
152                            Foam::IOobject::NO_READ,
153                            Foam::IOobject::NO_WRITE),
154             Cp.mesh(),
155             Foam::dimensionedScalar("zero",
156             Cp.dimensions()/Foam::dimTime, 0.0),
157             Cp.boundaryField().types()
158         ),
159         Foam::volScalarField
160         (
161             Foam::IOobject("Se", Ce.time().timeName(), Ce.db(),
162                            Foam::IOobject::NO_READ,
163                            Foam::IOobject::NO_WRITE),
164             Ce.mesh(),
165             Foam::dimensionedScalar("zero",
166             Ce.dimensions()/Foam::dimTime, 0.0),
167             Ce.boundaryField().types()
168         ),
169         Foam::volScalarField
170         (
171             Foam::IOobject("S1p", C1p.time().timeName(), C1p.db(),
172                            Foam::IOobject::NO_READ,
173                            Foam::IOobject::NO_WRITE),
174             C1p.mesh(),
175             Foam::dimensionedScalar("zero",
176             C1p.dimensions()/Foam::dimTime, 0.0),
177             C1p.boundaryField().types()
178         )
179     };
```

```
180
181     // Copy first 4 columns into fields
182     auto accY = Y.accessor<float,2>();
183     for (Foam::label i = 0; i < nCells; ++i)
184     {
185         S[0][i] = static_cast<Foam::scalar>(accY[i][0]); // dCg/dt
186         S[1][i] = static_cast<Foam::scalar>(accY[i][1]); // dCh/dt
187         S[2][i] = static_cast<Foam::scalar>(accY[i][2]); // dCc/dt
188         S[3][i] = static_cast<Foam::scalar>(accY[i][3]); // dCp/dt
189         S[4][i] = static_cast<Foam::scalar>(accY[i][4]); // dCe/dt
190         S[5][i] = static_cast<Foam::scalar>(accY[i][5]); // dC1p/dt
191         // ignore accY[i][6] if present (dT/dt)
192     }
193
194     return S;
195 }
```

Bibliografia

[1] Opeoluwa Owoyele e Pinaki Pal. «ChemNODE: A neural ordinary differential equations framework for efficient chemical kinetic solvers». In: *Energy and AI* 7 (2022), p. 100118 (cit. a p. 1).

[2] Weiqi Ji, Franz Richter, Michael J Gollner e Sili Deng. «Autonomous kinetic modeling of biomass pyrolysis using chemical reaction neural networks». In: *Combustion and Flame* 240 (2022), p. 111992 (cit. alle pp. 1, 2).

[3] Aleksandr Fedorov, Anna Perechodjuk e David Linke. «Kinetics-constrained neural ordinary differential equations: Artificial neural network models tailored for small data to boost kinetic model development». In: *Chemical Engineering Journal* 477 (2023), p. 146869 (cit. alle pp. 1, 2, 26, 31).

[4] Tadbhagya Kumar, Anuj Kumar e Pinaki Pal. «A Physics-Informed Autoencoder-NeuralODE Framework (Phy-ChemNODE) for Learning Complex Fuel Combustion Kinetics». In: *NeurIPS Machine Learning and the Physical Sciences Workshop*. 2024, p. 1 (cit. alle pp. 1, 4).

[5] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt e David K Duvenaud. «Neural ordinary differential equations». In: *Advances in neural information processing systems* 31 (2018) (cit. alle pp. 1, 22, 23).

[6] Weiqi Ji e Sili Deng. «Autonomous discovery of unknown reaction pathways from data by chemical reaction neural network». In: *The Journal of Physical Chemistry A* 125.4 (2021), pp. 1082–1092 (cit. a p. 2).

[7] Maziar Raissi, Paris Perdikaris e George E Karniadakis. «Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations». In: *Journal of Computational physics* 378 (2019), pp. 686–707 (cit. a p. 2).

[8] Gabriel S. Gusmão, Adhika P. Retnanto, Shashwati C. da Cunha e Andrew J. Medford. «Kinetics-informed neural networks». In: *Catalysis Today* 417 (2023). Transient Kinetics Seminar, p. 113701 (cit. a p. 2).

[9] André O Menezes, Michelly T Rodrigues, Adriana Zimmaro, Luiz EP Borges e Marco A Fraga. «Production of renewable hydrogen from aqueous-phase reforming of glycerol over Pt catalysts supported on different oxides». In: *Renewable Energy* 36.2 (2011), pp. 595–599 (cit. a p. 3).

[10] Guodong Wen, Yunpeng Xu, Huaijun Ma, Zhusheng Xu e Zhijian Tian. «Production of hydrogen by aqueous-phase reforming of glycerol». In: *International journal of hydrogen energy* 33.22 (2008), pp. 6657–6666 (cit. alle pp. 3, 25).

[11] Irene Coronado, M Stekrova, Matti Reinikainen, Pekka Simell, L Lefferts e J Lehtonen. «A review of catalytic aqueous-phase reforming of oxygenated hydrocarbons derived from biorefinery water fractions». In: *International Journal of Hydrogen Energy* 41.26 (2016), pp. 11003–11032 (cit. alle pp. 3, 26, 38).

[12] Esmail MA Mokheimer, Muhammad Ibrar Hussain, Shakeel Ahmed, Mohamed A Habib e Amro A Al-Qutub. «On the modeling of steam methane reforming». In: *Journal of Energy Resources Technology* 137.1 (2015), p. 012001 (cit. a p. 3).

[13] Marcos Roberto Monteiro, Cristie Luis Kugelmeier, Rafael Sanaiotte Pinheiro, Mario Otávio Batalha e Aldara da Silva César. «Glycerol from biodiesel production: Technological paths for sustainability». In: *Renewable and Sustainable Energy Reviews* 88 (2018), pp. 109–122 (cit. a p. 3).

[14] H.J. Alessio, G.L. Pestana, R.A. Comelli e J.M. Grau. «Hydrogen production via aqueous phase reforming of glycerol over Ni-Co/Al2O3 catalysts: Effect of support modification with lanthanides and alkaline earth metals». In: *Fuel* 404 (2025), p. 136217 (cit. a p. 4).

[15] Arthur L Samuel. «Some studies in machine learning using the game of checkers». In: *IBM Journal of research and development* 3.3 (1959), pp. 210–229 (cit. a p. 7).

[16] Jafar Alzubi, Anand Nayyar e Akshi Kumar. «Machine learning from theory to algorithms: an overview». In: *Journal of physics: conference series*. Vol. 1142. IOP Publishing. 2018, p. 012012 (cit. a p. 7).

[17] N d Sandhya e KR Charanjeet. «A review on machine learning techniques». In: *International Journal on Recent and Innovation Trends in Computing and Communication* 4.3 (2016), pp. 451–458 (cit. alle pp. 7, 8).

[18] Eduardo F Morales e Hugo Jair Escalante. «A brief introduction to supervised, unsupervised, and reinforcement learning». In: *Biosignal processing and classification using computational learning and intelligence*. Elsevier, 2022, pp. 111–129 (cit. alle pp. 8, 9).

[19] Trevor Hastie, Robert Tibshirani, Jerome Friedman et al. *The elements of statistical learning*. 2009 (cit. a p. 8).

[20] Jesper E Van Engelen e Holger H Hoos. «A survey on semi-supervised learning». In: *Machine learning* 109.2 (2020), pp. 373–440 (cit. a p. 9).

[21] Ludwig Fahrmeir, Thomas Kneib, Stefan Lang e Brian D Marx. «Regression models». In: *Regression: Models, methods and applications*. Springer, 2022, pp. 21–71 (cit. alle pp. 9, 12).

[22] Abhishek V Tatachar. «Comparative assessment of regression models based on model evaluation metrics». In: *International Research Journal of Engineering and Technology (IRJET)* 8.09 (2021), pp. 2395–0056 (cit. alle pp. 10, 11).

[23] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep Learning*. MIT Press, 2016 (cit. alle pp. 13, 20).

[24] Kevin Gurney. *An Introduction to Neural Networks*. 1ª ed. CRC Press, 1997 (cit. a p. 13).

[25] Osval Antonio Montesinos López, Abelardo Montesinos López e José Crossa. «Chapter 10: Fundamentals of Artificial Neural Networks and Deep Learning». In: *Multivariate Statistical Machine Learning Methods for Genomic Prediction*. Springer, Cham, 2022, pp. 379–425 (cit. a p. 14).

[26] Frank Rosenblatt. «The perceptron: a probabilistic model for information storage and organization in the brain.» In: *Psychological review* 65.6 (1958), p. 386 (cit. a p. 15).

[27] Johannes Lederer. «Activation functions in artificial neural networks: A systematic overview». In: *arXiv preprint arXiv:2101.09957* (2021) (cit. a p. 16).

[28] Stefan Elfwing, Eiji Uchibe e Kenji Doya. «Sigmoid-weighted linear units for neural network function approximation in reinforcement learning». In: *Neural networks* 107 (2018), pp. 3–11 (cit. a p. 17).

[29] Rolf Sander. «Compilation of Henry's law constants (version 5.0. 0) for water as solvent.» In: *Atmospheric Chemistry & Physics* 23.19 (2023) (cit. alle pp. 19, 28).

[30] Michael A Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA, 2015 (cit. a p. 21).

[31] Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama e Jinkyoo Park. «Torchdyn: A neural differential equations library». In: *arXiv preprint arXiv:2009.09346* (2020) (cit. a p. 25).

[32] Nianjun Luo, Xianwen Fu, Fahai Cao, Tiancun Xiao e Peter P Edwards. «Glycerol aqueous phase reforming for hydrogen generation over Pt catalyst–Effect of catalyst composition and reaction conditions». In: *Fuel* 87.17-18 (2008), pp. 3483–3489 (cit. alle pp. 25, 26, 38).

[33] Suresh Kumar Sahani e Binod Kumar Sah. «An In-Depth Stability and Convergence Analysis of the Runge-Kutta 4th Order Method for Nonlinear Ordinary Differential Equations». In: *Panamerican Mathematical Journal* 34.2 (2024), pp. 300–312 (cit. a p. 31).

[34] A Yu Beliaev e Serguei M Kozlov. «Darcy equation for random porous media». In: *Communications on pure and applied mathematics* 49.1 (1996), pp. 1–34 (cit. a p. 48).

[35] William C. Lyons. «Chapter 1 - Basic Principles, Definitions, and Data». In: *Working Guide to Reservoir Engineering*. A cura di William C. Lyons. Gulf Professional Publishing, 2010, pp. 1–95 (cit. a p. 49).

[36] F Moukalled L Mangani M Darwish. *The finite volume method in computational fluid dynamics*. 2016 (cit. a p. 49).

[37] Joel H Ferziger, Milovan Perić e Robert L Street. *Computational methods for fluid dynamics*. springer, 2019 (cit. alle pp. 50, 51).

[38] Kaishuo Yang et al. «Quantitative tortuosity measurements of carbonate rocks using pulsed field gradient NMR». In: *Transport in Porous Media* 130.3 (2019), pp. 847–865 (cit. a p. 62).

[39] Alice De Matteis. «Analisi CFD di flusso e dispersione in reattori a letto impaccato: ottimizzazione e scale-up = CFD analysis of flow and dispersion in packed-bed reactors: optimization and scale-up». Supervisors: Daniele Marchisio, Gianluca Boccardo. Corso di laurea magistrale in Ingegneria Chimica e dei Processi Sostenibili. Master's Thesis. Torino, Italy: Politecnico di Torino, 2024 (cit. a p. 62).

Ringraziamenti

Arrivata alla conclusione di questo progetto di tesi, vorrei ringraziare in primo luogo i Professori Gianluca Boccardo e Daniele Marchisio per avermi dato l'opportunità di approcciare il mondo del Machine Learning e della CFD e per avermi accompagnato nello sviluppo di questo lavoro. Un ringraziamento speciale va alla Dott.ssa Agnese Marcato, per avermi affiancato in questi mesi con una preparazione, una pazienza e una gentilezza uniche. Ringrazio il Dott. Diego Fida per aver condiviso e messo a disposizione i risultati ottenuti dal proprio lavoro di ricerca per permettermi di lavorare a questo progetto.

Vorrei ringraziare tutti i miei amici, per avermi supportato e sopportato non solo in questi ultimi mesi di duro lavoro, ma nella vita di tutti i giorni: vi ringrazio perché in ognuno di voi scopro continuamente dei tesori nascosti, che mi fanno sentire fortunata di avervi nella mia vita.

Grazie a te, Jacopo, per tutto quello che hai fatto in questi anni. Non ti arrendi mai, almeno con me, trovi sempre il modo per strapparmi un sorriso, anche quando capisci che tutto mi va di fare tranne che scherzare; è un privilegio averti nella mia vita.

Ringrazio immensamente i miei quattro nonni per aver sempre avuto un sorriso da regalarmi accompagnato dalle storie della vostra vita, che porterò sempre con me come preziosi insegnamenti e ricordi meravigliosi.

Ringrazio il mio Papà, che in questi ultimi anni ha visto il progetto di una vita intera sgretolarsi in un battito di ciglia, per aver fatto in modo che la mia di vita e quella dei miei fratelli potesse comunque continuare. A te va tutto il mio riconoscimento per i sacrifici che hai fatto.

Ringrazio mio fratello, Giovanni, e mia sorella, Elisa: credetemi se non trovo le parole per esprimere quello che provo quando vi penso. Siamo tre persone completamente diverse, ma allo stesso tempo con gli stessi valori; nulla mi rende più serena che sapervi accanto a me.

L'ultimo grazie va a te, Mamma. Ti rivedo sempre nei sorrisi e negli sguardi gentili delle persone che mi stanno accanto; non passa giorno in cui io non ti senta qui con me. A te, Mamma, devo tutto.