

Analisi e Visualizzazione della Stabilità dei Sistemi Dinamici

Riva Daniele

31/10/2024 - 04/04/2025

Abstract

In questo elaborato presento alcuni strumenti e tecniche per analizzare e visualizzare la stabilità di un sistema dinamico. In tale contesto per ‘stabilità’ non si intende la convergenza o la divergenza di specifici stati in punti di equilibrio, piuttosto la *sensibilità* di una certa regione di configurazioni del sistema a delle perturbazioni dopo l’evoluzione di tale regione per un certo tempo. Inoltre tengo subito a precisare che questa analisi viene fatta dopo la simulazione del sistema. Poiché gli strumenti presentati non permettono di analizzare a priori il comportamento del sistema l’utilità di questa tecnica è pressoché nulla.

Indice

1 Fondamenti Teorici	2
1.1 Introduzione Teorica	2
1.1.1 Il Sistema Dinamico e l'Operatore di Stabilità	3
1.1.2 Interpretazione Intuitiva di $\mathcal{C}(t)$	8
1.1.3 Formalizzazione della Sensibilità	13
1.1.4 Esempi Illustrativi e Verifica	17
1.2 Metodi di Discretizzazione	21
1.2.1 Definizione del Sistema Dinamico	21
1.2.2 Integrazione nel Sistema Dinamico	23
1.2.3 Impegno Computazionale	27
1.2.4 Esempio di Calcolo	29
2 Implementazione	34
2.1 Implementazione in Python	34
2.1.1 Definizione del Sistema Dinamico	35
2.1.2 Visualizzazione dello Spazio delle Fasi	40
2.1.3 Utilizzo di CUDA	42
2.2 Visualizzazioni dei Risultati	45
2.2.1 Simulazione della Competizione Interspecifica	47
2.2.2 Simulazione del Modello di Hodgkin-Huxley	53
2.3 La Funzione $\mathcal{C}_{\vec{p}}(t)$	57
2.3.1 Pendolo semplice	58
2.3.2 Lotka-Volterra	62
3 Conclusioni e Riflessioni	65
3.1 L'Operatore di Stabilità	65
3.1.1 Analisi e osservazioni	65
3.1.2 L'(in)utilità dell'operatore	66
3.2 Implementazione dell'Operatore di Stabilità	66
3.2.1 Python e C++	66
3.2.2 IA	67

Fondamenti Teorici

In questo capitolo viene introdotto il concetto di sensibilità e viene definito l'operatore di stabilità, ovvero lo strumento in grado di analizzare la sensibilità del sistema di un sistema dinamico.

Nella prima sezione l'operatore viene definito in modo teorico, in seguito viene presentata l'idea sulla quale si fonda in modo più intuitivo.

La seconda sezione, invece, tratta la discretizzazione dell'operatore, di come viene definito il sistema dinamico sul quale agisce e dell'integrazione che opera.

1.1 Introduzione Teorica

Alla base di tutto quello che viene presentato in questo documento vi è il concetto di sensibilità. Per sensibilità si intende la variazione nel tempo di una configurazione in seguito ad una piccola variazione (una perturbazione nello stato iniziale).

Il Concetto di Regione

Al fine di quantificare la sensibilità di uno stato è necessario utilizzare il concetto di regione.

Se il modello (o sistema dinamico) è definito solamente da un grado di libertà, allora la regione di una configurazione coincide con le due configurazioni che differiscono in egual misura dalla prima. Se i gradi di libertà sono due allora la regione di una configurazione coincide con le infinite configurazioni che differiscono in egual misura dalla prima. In generale per regione di una configurazione si intende l'insieme di tutte quelle configurazioni del modello che hanno una differenza fissata dalla configurazione di riferimento.

La costante c e c'

È necessario chiarire che d'ora in poi tutti i sistemi dinamici presi in considerazione hanno due gradi di libertà; in tali sistemi c è una costante che dipende solamente dalla differenza fissata per le regioni. Questa differenza dovrebbe

essere pensata come molto piccola:

$$c = 2\pi\epsilon$$

ϵ è il valore che corrisponde alla differenza tra gli stati appartenenti ad una regione e la sua configurazione di riferimento.

Si vuole ora analizzare come muta, durante l'evoluzione del sistema, l'insieme di tutte quelle configurazioni che appartengono alla regione. È chiaro che ogni elemento appartenente alla regione si evolverà in modo differente, ma sempre più simile alla configurazione di riferimento, per un ϵ che diventa sempre più piccolo. c' , un valore associato al cambiamento della regione nel tempo, verrà definito formalmente in seguito.

Il Rapporto $\frac{c'}{c}$

Quello che l'operatore di stabilità genera è un campo scalare, dove lo scalare associato ad ogni stato del sistema è il rapporto $\frac{c'}{c}$. Essendo $c, c' \geq 0$ il loro rapporto sarà necessariamente maggiore di 0. Se è compreso tra 0 e 1 la configurazione a cui è associato questo valore sarà considerata stabile, se invece il valore sarà maggiore di 1 quella configurazione sarà considerata instabile.

È fondamentale ricordare però che la stabilità/instabilità della configurazione è sempre relativa ad un certo istante t . Gli stati di un modello sono classificati come stabili o instabili utilizzando il tempo t di evoluzione nel sistema come parametro. Se uno stato viene classificato come stabile per $t = 2.3$ non significa che possa divergere nel senso classico del termine¹.

1.1.1 Il Sistema Dinamico e l'Operatore di Stabilità

Un sistema dinamico è un modello matematico che descrive l'evoluzione nel tempo di un fenomeno. Per farlo si avvale di un certo numero di gradi di libertà che permettono di descrivere tutte le possibili configurazioni del sistema.

In ogni istante è possibile descrivere completamente lo stato del sistema attraverso queste variabili.

Definizione del Sistema Dinamico

Un sistema dinamico viene solitamente descritto da un sistema di equazioni differenziali che permette di calcolare l'evoluzione di uno stato nel tempo. I modelli che vengono qui considerati sono definiti come segue

$$\begin{cases} \frac{dx}{dt} = f(x, y) \\ \frac{dy}{dt} = g(x, y) \end{cases}$$

¹Questa è una delle tante ragioni che mi portano a concludere che questo operatore è poco efficace.

Nonostante il sistema di equazioni in sé sia completamente deterministico, spesso non è possibile trovare una soluzione analitica a queste equazioni.

Si prenda in considerazione il sistema di equazioni differenziali che descrivono il moto di un pendolo semplice:

$$\begin{cases} \frac{d\theta}{dt} = \omega \\ \frac{d\omega}{dt} = -\frac{g}{l} \sin \theta \end{cases}$$

Si ricava banalmente che il sistema può essere scritto come:

$$\ddot{\theta} + \frac{g}{l} \sin \theta = 0$$

Nonostante l'apparente semplicità dell'equazione non esiste una soluzione analitica esatta.

Stato di un Sistema Dinamico

Per stato o configurazione \vec{s} di un sistema dinamico si intende la descrizione del sistema in un preciso istante attraverso tutte le variabili che lo definiscono. Lo stato è costituito essenzialmente da un insieme ordinato di numeri reali, in questo caso due, in cui ogni elemento che appartiene allo stato esprime quantitativamente il valore assunto da un grado di libertà. Possono essere scritti nella forma $[i, j]$ con $i, j \in \mathbb{R}$ in questo modo:

$$\vec{s} = [i, j]$$

La differenza tra due stati $\vec{s} = [i, j]$ e $\vec{s}' = [i', j']$ può essere misurata come:

$$d = \sqrt{(i' - i)^2 + (j' - j)^2}$$

Questa misura vuole esprimere quanto due stati siano diversi tra loro.

Spazio delle Fasi

Per rappresentare l'evoluzione del sistema dinamico si può utilizzare lo spazio delle fasi: questo è uno spazio matematico all'interno del quale ogni punto descrive univocamente uno stato del sistema. Ogni grado di libertà del sistema dinamico è rappresentato da un asse in uno spazio multidimensionale. Poiché i sistemi qui considerati hanno due gradi di libertà, lo spazio delle fasi è detto piano delle fasi. Un punto $\vec{p} = (x, y)$ nel piano delle fasi può essere rappresentato anche come un vettore. Ad esso è univocamente associato uno stato \vec{s} :

$$\vec{s} \leftrightarrow \vec{p}^2$$

²Immagino che questo sia considerato abuso di notazione.

La differenza tra due stati è graficamente rappresentata come la misura del segmento che collega i punti \vec{p} e \vec{p}' . Più uno stato è simile ad un altro minore sarà la distanza geometrica tra i punti che rappresentano questi due stati.

Stabilito un certo punto \vec{p} se ne può calcolare l'evoluzione \vec{p}' dopo un tempo infinitesimo dt . Collegando questi punti con un segmento e ripetendo questa operazione più volte, si ottiene la traiettoria di quel punto. La traiettoria descrive l'evoluzione di uno stato nel tempo.

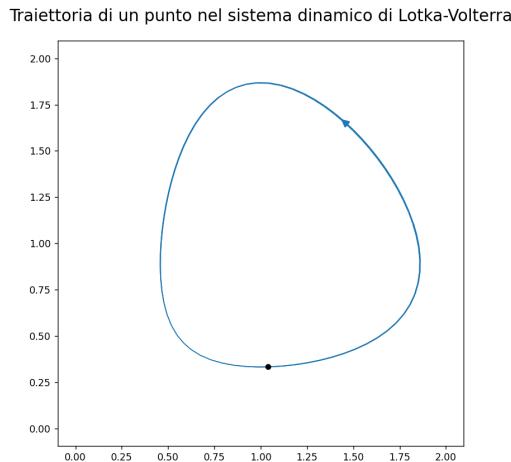


Figura 1.1: La traiettoria del punto $(1.04, 0.33)$ nel sistema dinamico Lotka-Volterra. L'asse orizzontale corrisponde alla popolazione delle prede mentre l'asse verticale rappresenta la popolazione dei predatori. La traiettoria mostra come la popolazione di una specie di predatore e di prede interagisce ed evolve nel tempo.

Se invece si associa ad ogni punto un vettore \vec{p}' , ma si ripete l'operazione una sola volta normalizzando adeguatamente i vettori nel piano della fase, si ottiene quello che potrebbe essere considerato il campo vettoriale \vec{F} del sistema dinamico

$$\vec{F}(\vec{p}) = \frac{d\vec{p}}{dt}$$

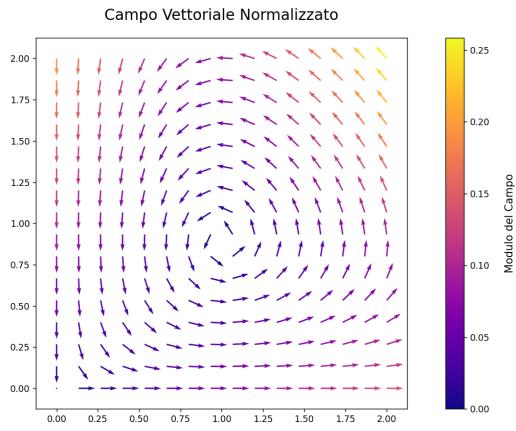


Figura 1.2: Il campo vettoriale del modello della figura 1.1, calcolato per $\text{dt} = 0.001$.

Il sistema di equazioni differenziali utilizzato per le immagini 1.1 e 1.2 è

$$\begin{cases} \frac{dx}{dt} = x \left(\frac{2}{3} - \frac{3}{4}y \right) \\ \frac{dy}{dt} = y(x - 1) \end{cases} \quad (1.1)$$

Introduzione a $\mathcal{C}(t)$

È possibile illustrare graficamente il concetto di regione. Considerando un punto (x, y) , la regione di questo punto corrisponderà graficamente ad una circonferenza di centro (x, y) e di raggio ϵ

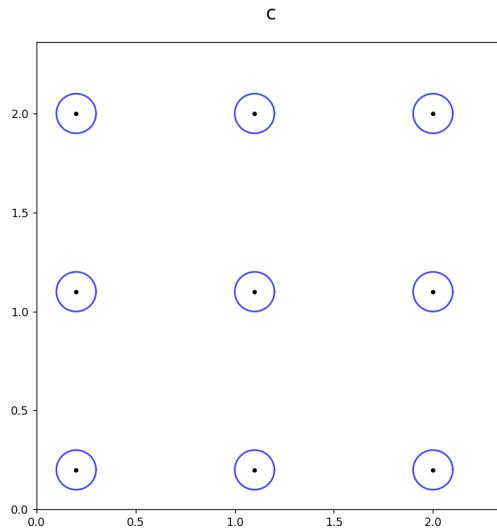


Figura 1.3: Un insieme di 9 punti distribuiti linearmente nell'intervallo $[0.2, 2]$ su entrambi gli assi e le loro regioni.

Nella figura 1.3 le regioni dei punti hanno raggio $\epsilon = 0.1$, di conseguenza si ricava che $c = \frac{\pi}{5}$. Come c rappresenta la lunghezza della circonferenza di centro (x, y) , così c' indica la lunghezza della stessa circonferenza dopo che i suoi infiniti punti si sono evoluti nel sistema per un certo tempo t .

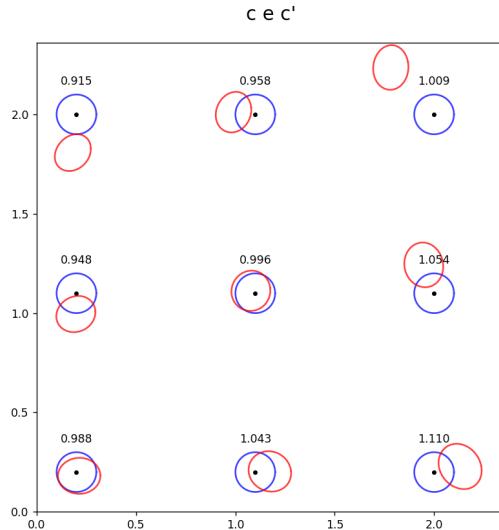


Figura 1.4: Evoluzione delle regioni dei 9 punti per $t = 0.125$ e $dt = 0.00001$. Le regioni sono approssimate con poligoni regolari di 500000 lati. Le circonference blu rappresentano le regioni di partenza, quelle rosse rappresentano l'evoluzione delle stesse dopo t .

Il valore approssimato al millesimo sopra ogni punto in figura 1.4 è il rapporto $\frac{c'}{c}$, ovvero il rapporto tra la lunghezza della circonferenza deformata rossa e la circonferenza blu.

Aumentando sufficientemente la densità della distribuzione dei punti sulla griglia e diminuendo sensibilmente il raggio delle regioni dei punti si ottiene un'approssimazione di quello che l'operatore $\mathcal{C}(t)$ restituisce

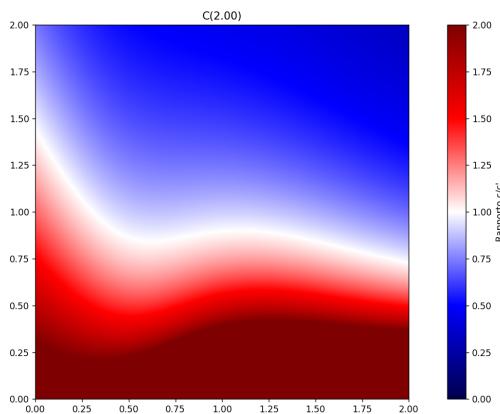


Figura 1.5: Il campo scalare che restituisce $\mathcal{C}(2)$. La griglia è suddivisa in 250^2 punti. Le circonference sono approssimate da poligoni regolari di 50 lati, l'integrazione è stata eseguita per $dt = 0.001$.

Nell'immagine sono state effettuate anche delle operazione di *mapping* e di *clipping*.

In primo luogo il campo scalare viene rappresentato nel grafico utilizzando una mappa di colori che associa i valori assunti dal campo ad un intervallo di colori che va dal blu al rosso, evidenziando però con una sottile fascia bianca i punti dove la stabilità cambia, ovvero i punti in cui $\mathcal{C}(2)$ cambia da valori compresi tra 0 e 1 a valori superiori a 1.

Per migliorare l'interpretabilità del grafico la mappa di colore esegue un'operazione di *clipping* simile a:

$$\min(\mathcal{C}(t), 2)$$

Così facendo vengono messi in risalto i punti dove il sistema cambia stabilità.

Per quanto riguarda la notazione, in $\mathcal{C}(t)$ viene sottinteso il campo sul quale l'operatore viene applicato, infatti è specificato solamente il tempo di evoluzione. Se fosse necessario usare l'operatore su sistemi dinamici differenti in una singola espressione, allora si può anteporre il campo $\vec{\mathbf{F}}_k$ al tempo di evoluzione

$$\mathcal{C}(\vec{\mathbf{F}}_1, 2.75) - \mathcal{C}(\vec{\mathbf{F}}_2, 0.23) = \mathcal{C}(\vec{\mathbf{F}}_3, 15.23)$$

1.1.2 Interpretazione Intuitiva di $\mathcal{C}(t)$

In questa sezione si esplicita il significato reale, non matematico o geometrico, di $\mathcal{C}(t)$. Per farlo si utilizzano tre esempi, il primo che prende in esame il modello di Lotka-Volterra, il secondo il modello di un pendolo semplice ed il terzo il modello ridotto di Hodgkin-Huxley. Gli esempi che seguono sono molto generali e non hanno la pretesa di voler approfondire ulteriormente la connotazione matematica o geometrica dell'operatore, bensì vogliono chiarire la corrispondenza tra il campo scalare $\mathcal{C}(t)$ e il significato dei suoi singoli valori nel sistema dinamico in sé.

Analogia del Modello Preda-Predatore

Il modello utilizzato nelle figure precedenti descrive l'evoluzione della popolazione di due specie animali³, una di predatori e l'altra di prede che interagiscono in un ambiente chiuso. Si tratta dunque di un'astrazione priva di fattori esterni come la limitazione di risorse.

In primo luogo è necessario chiarire cosa rappresenta una regione quando riferita ad un modello come questo. È fondamentale tenere presente che il raggio ϵ della regione va sempre pensato come molto piccolo. La regione dello stato che conta 15000 prede e 5000 predatori sarebbe semplicemente quell'insieme di coppie (*#prede*, *#predatori*) che sono molto simili al relativo stato. Questo corrisponderebbe all'insieme degli stati in cui si è verificata una piccolissima variazione nel numero di prede, nel numero di predatori o in entrambi.

³Vedi figura 1.1.

Se il numero di prede fosse 10000 e il numero di predatori 2300, si potrebbe considerare appartenente alla regione di $(10000, 2300)$ lo stato del sistema dove sono presenti 10001 prede e 2300 predatori o 9999 prede e 2300 predatori. Ovviamente si potrebbe variare anche il numero di predatori, non importa al fine dell'esempio che ϵ sia *esattamente* lo stesso in tutti i punti della regione, anche perché in questo esempio concreto le due variabili sarebbero discrete, non ha senso parlare di 2300.52 predatori, l'importante è che ci sia una variazione molto piccola nel numero di prede o nel numero di predatori. In breve, la regione di una configurazione è l'insieme delle situazioni che le sono molto simili.

L'operatore confronta questi nuovi stati della regione dopo la loro evoluzione nel sistema. Se gli stati $(10001, 2300)$ e $(9999, 2300)$ si sono evoluti verso una simile soluzione, allora significa che un'ipotetica piccola variazione nelle condizioni iniziali $(10000, 2300)$ non avrebbe influenzato di molto l'evoluzione del sistema e che col tempo si sarebbe stabilizzata. Viceversa, se questa variazione porta i due nuovi stati a divergere e a differenziarsi sempre di più, allora significa che quella regione è instabile. L'aggiunta anche di una sola preda o di un solo predatore porterebbe a delle situazioni molto differenti per quell'istante. Si considerino le seguenti due regioni:

	\vec{s}_1	\vec{s}_t
A	9999, 2300	2600, 1000
B	10001, 2299	2590, 1000

	\vec{s}_2	\vec{s}_t
A	10000, 1000	7000, 1500
B	10005, 1005	7000, 1500

Nella prima tabella sono riportati due punti (A e B) che appartengono ad una certa regione \vec{s}_1 . La popolazione delle prede e dei predatori dopo l'evoluzione per un certo tempo t è descritta da \vec{s}_t . Si noti che:

$$\Delta \vec{s}_1 < \Delta \vec{s}_t$$

Dove

$$\begin{aligned}\Delta \vec{s}_1 &= \vec{s}_{1_B} - \vec{s}_{1_A} \\ \Delta \vec{s}_t &= \vec{s}_{t_B} - \vec{s}_{t_A}\end{aligned}$$

Essenzialmente la differenza introdotta si è andata ad amplificare, per questo motivo \vec{s}_1 si dice instabile per t .

Nella seconda tabella invece la situazione è invertita

$$\Delta \vec{s}_2 > \Delta \vec{s}_t$$

In questo caso quindi, dato che la differenza introdotta è andata via via scomparendo, si dice che \vec{s}_2 è stabile per t . In questi esempi l'utilizzo di \vec{s} al posto di \vec{p} è semplicemente un modo per sottolineare la concretezza dell'esempio, \vec{p} è un vettore che in sé è privo di significato, \vec{s} invece descrive una situazione concreta facendo riferimento ad una precisa configurazione di un modello. È la relazione stabilità tra \vec{p} e \vec{s} che permette di attribuire un significato reale ad elementi astratti come vettori e funzioni.

Questa piccola variazione nelle condizioni iniziali si può intendere anche come una debole perturbazione. Il termine ‘perturbazione’ sarebbe più adatto in realtà ad un contesto come quello della meccanica classica. In questo caso per perturbazione di un sistema si intende una modifica allo stato del sistema dovuto a dei fattori esterni al modello, come ad esempio il rumore termico o il moto browniano. Questi tipi di fattori vengono spesso ignorati perché non influenzano in modo significativo l’evoluzione del sistema; l’operatore $\mathcal{C}(t)$ tenta semplicemente di misurare quanto questi fattori avrebbero influito sul sistema se fossero stati inclusi nel modello.

Analogia del Modello di un Pendolo Semplice

In questo sottoparagrafo il sistema dinamico preso in considerazione è quello di un pendolo semplice. Come nel caso precedente, anche in questo, al fine dell’esempio, non è significativa una rigorosa descrizione del sistema, ma si vuole semplicemente evidenziare quali sono i collegamenti tra un modello ben noto e quello che è possibile dedurre dall’operatore di stabilità.

Si prendano in considerazione 2 stati del sistema: nel primo il pendolo è in un equilibrio stabile, nel secondo invece è in un equilibrio instabile. Se si tiene in considerazione anche l’attrito, per un valore sufficientemente elevato di t , qualsiasi stato convergerà nella posizione di equilibrio se soggetto ad una piccola variazione nelle condizioni iniziali del sistema. Assegnando a t un valore adeguato l’operatore potrebbe rivelare dettagli non del tutto evidenti o banali sulla stabilità del sistema.

Nel primo caso, una piccola variazione, sia nell’angolo sia nella velocità angolare, non porta il pendolo ad uno stato molto diverso da quello che aveva all’inizio: si pensi ad un pendolo in equilibrio stabile lasciato all’aria aperta, le deboli correnti d’aria agiscono in maniera del tutto trascurabile su un sistema in equilibrio stabile come questo. Il pendolo, per qualsiasi t scelto, rimarrà dunque molto vicino al punto di equilibrio stabile e tenderà sempre ad avvicinarsi.

Viceversa, nel secondo caso, una qualsiasi variazione nello stato del sistema porta il pendolo a cadere o a destra o a sinistra, in questo caso ogni variazione, per quanto piccola, non è trascurabile.

Analogia del Modello di Hodgkin-Huxley

Il modello di Hodgkin-Huxley è un sistema dinamico molto complesso, fu proposto nel 1952 per descrivere matematicamente il comportamento dei neuroni e valse ad Alan Hodgkin e ad Andrew Huxley il nobel per la fisiologia nel 1963.

Il modello originale utilizza quattro variabili di stato per descrivere l’evoluzione di un neurone; data l’impossibilità di visualizzare un sistema con così

tante variabili sono stati sviluppati molti modelli semplificati a due variabili di stato che permettono di simulare tre comportamenti di un neurone⁴:

1. Ritorno all'equilibrio stabile: il neurone riceve un input elettrico ma questo non lo porta a rispondere, è come se il segnale venisse direttamente assorbito dal neurone.
2. Singolo picco: il neurone, in seguito ad un input elettrico sufficientemente forte, risponde con un picco nel potenziale elettrico.
3. Ciclo di picchi: il neurone, in seguito ad un input elettrico ben preciso, entra in un ciclo che fa produrre al neurone stesso una serie di picchi nel potenziale elettrico.

Il modello riesce a spiegare una delle più importanti caratteristiche dei neuroni, ovvero la dipendenza non solo dalle condizioni in cui si trova il sistema in un dato momento, ma anche la dipendenza da momenti precedenti. Due neuroni identici nella loro struttura, a parità di condizioni esterne, possono manifestare comportamenti completamente differenti in seguito ad uno stesso input.

Un neurone può essere semplificato come un circuito elettrico nel quale sono presenti componenti che resistono e immagazzinano carica elettrica. La membrana del neurone separa l'ambiente intracellulare da quella extracellulare, questi due ambienti hanno una differenza di cariche e il doppio strato lipidico si comporta come un condensatore. Sono presenti inoltre dei canali nella membrana che permettono agli ioni di entrare o uscire dal neurone, modificando la carica totale interna del neurone stesso. Questi canali dipendono dal voltaggio e dalla distribuzione delle cariche all'interno della membrana. In particolare, nel modello qui trattato, le variabili di stato sono V^5 e n^6 .

In una delle molteplici configurazioni di questo modello è possibile osservare che il campo vettoriale è diviso in due aree, una rossa e una blu, come riportato in figura 1.6. Le due aree sono separate da una linea detta *limit cycle* (lett. “ciclo limite”), se lo stato del neurone risulta essere interno al *limit cycle* allora esso entra in un ciclo che lo porta a produrre una serie di scariche elettriche. Se invece lo stato del neurone corrisponde ad un punto esterno al *limit cycle* allora il neurone convergerà verso il punto di equilibrio stabile, detto ‘resting state’ o ‘stable node’.

⁴Questa è una grandissima semplificazione che non rende assolutamente giustizia al modello, ma approfondire questo argomento va decisamente oltre le finalità di questa relazione.

⁵Differenza di potenziale.

⁶Valore che varia tra 0 e 1 che rappresenta la percentuale di canali ionici del potassio aperti.

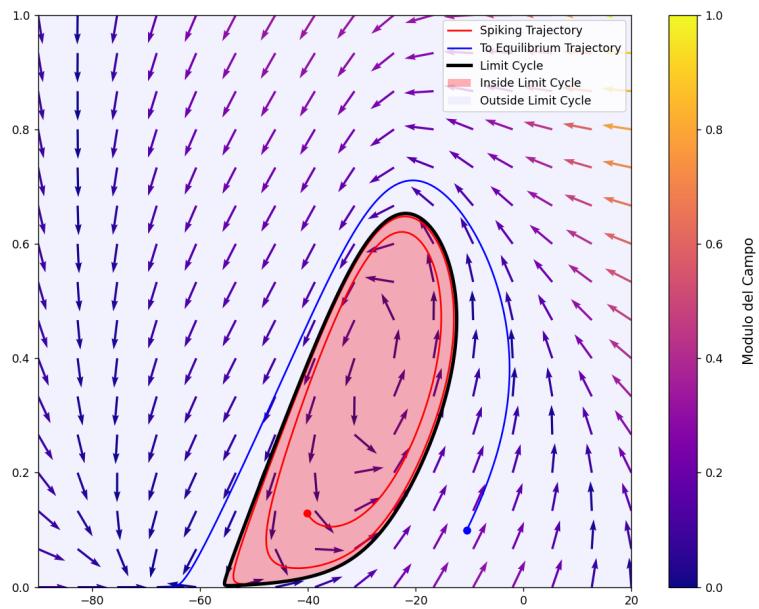


Figura 1.6: *Limit cycle* e le traiettorie di due punti, uno interno alla regione e uno esterno.

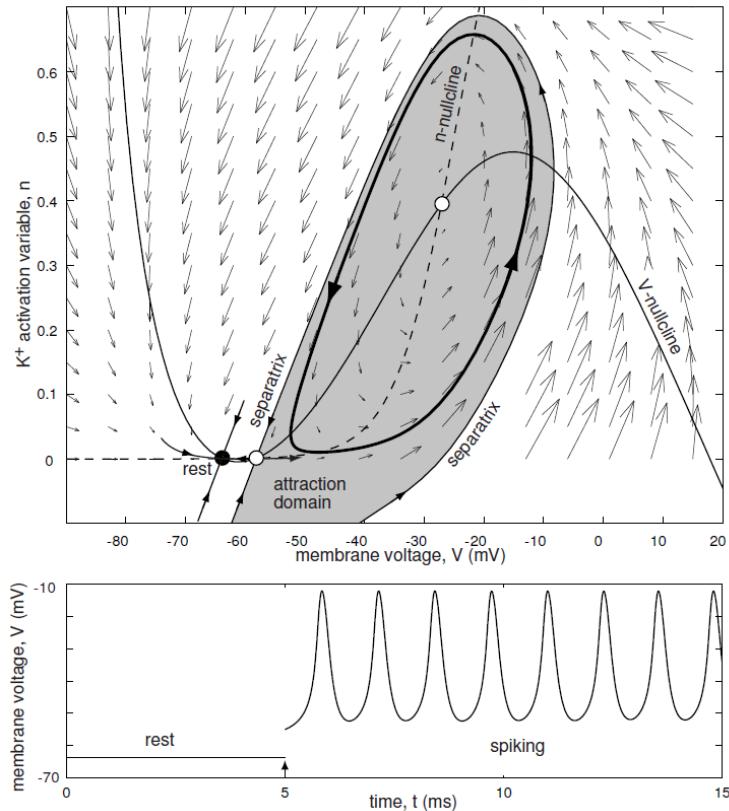


Figura 1.7: Grafici tratti da (Izhikevich, E. M. 2010) che mostrano, oltre al *limit cycle* e al campo vettoriale, la differenza di potenziale rispetto al tempo di evoluzione nel modello.

L'operatore di stabilità, in questo caso, mostrerà, per un tempo adatto, che la zona interna al *limit cycle* è instabile, mentre quella esterna è stabile. In realtà, aumentando sufficientemente il tempo di evoluzione nel modello, anche i punti interni al *limit cycle* alla fine convergono sul *limit cycle* stesso. Entrambe le regioni quindi sono caratterizzate da una convergenza: la regione esterna si può considerare convergente in senso stretto (tutti i punti in questa regione convergono in un unico punto di equilibrio); i punti nella regione interna convergono anch'essi, ma una piccola perturbazione dello stato non altera in modo significativo la traiettoria del punto. Questo stato finirà comunque all'interno del *limit cycle* se la perturbazione non lo porta nella regione esterna.

1.1.3 Formalizzazione della Sensibilità

In questa sezione si presenta la formalizzazione dell'operatore $\mathcal{C}(t)$ di un sistema dinamico con due gradi di libertà.

Definizione

Siano i e j i due gradi di libertà propri del sistema. Sia P lo spazio delle fasi in $\mathbb{R} \times \mathbb{R}$ che rappresenta ogni possibile stato del sistema. Siano x e y le due variabili di P che rappresentano i e j

$$[i, j] \leftrightarrow (x, y)$$

Sia $\vec{\mathbf{p}}$ un generico punto in P

$$\vec{\mathbf{p}} = (x, y)$$

L'operatore $\mathcal{C}(t)$ associa ad ogni punto $\vec{\mathbf{p}}$ in P uno scalare s a formare il campo scalare S .

$$\mathcal{C}(t) : \mathbb{R}^2 \rightarrow \mathbb{R}$$

Regione

Sia ϵ un valore arbitrariamente piccolo, sia c definito come segue

$$c = 2\pi\epsilon$$

Sia C l'insieme dei punti $\vec{c} \in \mathbb{R}^2$ generati da

$$\vec{c}_k(x, y) = \left(\epsilon \cdot \cos \frac{2\pi \cdot k}{n} + x, \epsilon \cdot \sin \frac{2\pi \cdot k}{n} + y \right)$$

Con $n \in \mathbb{N}$ e $k \in [0, n]$ ⁷ per $n \rightarrow \infty$ al variare di k .

$$C_{(x,y)} = \{ \vec{c}_k(x, y) \mid k \in [0, n] \}$$

⁷Immagino che debba specificare anche che questo intervallo sia $\in \mathbb{N}$

Sia $l(C_{(x,y)})$ la funzione che associa all'insieme $C_{(x,y)}$ lo scalare c

$$l(C_{(x,y)} : C \rightarrow \mathbb{R}$$

Sia $l(C_{(x,y)})$ definita come

$$l(C_{(x,y)}) = \lim_{n \rightarrow \infty} \sum_{k=0}^n |\vec{c}_k(x, y) - \vec{c}_{k+1}(x, y)|$$

Dimostrazione di $l(C_{(x,y)}) = c$

Si dimostra ora quanto segue

$$\lim_{n \rightarrow \infty} \sum_{k=0}^n |\vec{c}_k(x, y) - \vec{c}_{k+1}(x, y)| = 2\pi\epsilon$$

Per prima cosa si verifica che $|\vec{c}_k - \vec{c}_{k+1}|$ è costante al variare di k , in modo da sostituire la sommatoria con una moltiplicazione. Sia $d = |\vec{c}_k(x, y) - \vec{c}_{k+1}(x, y)|$

$$d = \left| \left(\epsilon \cdot \cos \frac{2\pi \cdot k}{n} + x, \epsilon \cdot \sin \frac{2\pi \cdot k}{n} + y \right) - \left(\epsilon \cdot \cos \frac{2\pi \cdot (k+1)}{n} + x, \epsilon \cdot \sin \frac{2\pi \cdot (k+1)}{n} + y \right) \right|$$

Si effettua ora una traslazione per riportare il centro di C nell'origine

$$\vec{v} = \begin{cases} x' = c_{k_x} - x \\ y' = c_{k_y} - y \end{cases}$$

Dove c_{k_x} e c_{k_y} sono le componenti x e y di $\vec{c}_k(x, y)$. In seguito alla trasformazione l'equazione diventa

$$d = \left| \left(\epsilon \cdot \cos \frac{2\pi \cdot k}{n}, \epsilon \cdot \sin \frac{2\pi \cdot k}{n} \right) - \left(\epsilon \cdot \cos \frac{2\pi \cdot (k+1)}{n}, \epsilon \cdot \sin \frac{2\pi \cdot (k+1)}{n} \right) \right|$$

Si effettua ora la trasformazione in coordinate polari $(x, y) \rightarrow (r, \theta)$

$$\begin{cases} x = r \cos \theta \\ y = r \sin \theta \end{cases}$$

Per ottenere

$$d = \left| \left(\epsilon, \frac{2\pi \cdot k}{n} \right) - \left(\epsilon, \frac{2\pi \cdot (k+1)}{n} \right) \right|$$

Sapendo che la distanza tra due punti nel sistema di coordinate polari è data da

$$d = \sqrt{r_1^2 + r_2^2 - 2r_1 r_2 \cos(\theta_2 - \theta_1)}$$

E utilizzando

$$\begin{aligned} r_1 &= r_2 = \epsilon \\ \theta_1 &= \frac{2\pi \cdot k}{n} \\ \theta_2 &= \frac{2\pi \cdot (k+1)}{n} \end{aligned}$$

Si ottiene

$$\begin{aligned} d &= \sqrt{\epsilon^2 + \epsilon^2 - 2\epsilon^2 \cos\left(\frac{2\pi \cdot k}{n} - \frac{2\pi \cdot (k+1)}{n}\right)} \\ d &= \sqrt{2\epsilon^2 - 2\epsilon^2 \cos\left(\frac{2\pi \cdot k - 2\pi \cdot (k+1)}{n}\right)} \\ d &= \sqrt{2\epsilon^2 - 2\epsilon^2 \cos\left(\frac{2k\pi - 2k\pi + 2\pi}{n}\right)} \\ d &= \sqrt{2\epsilon^2 - 2\epsilon^2 \cos\left(\frac{2\pi}{n}\right)} \end{aligned}$$

Effettivamente si trova un'espressione che non dipende da k , si può dunque concludere che

$$\lim_{n \rightarrow \infty} \sum_{k=0}^{n-1} d = \lim_{n \rightarrow \infty} \sum_{k=0}^{n-1} \sqrt{2\epsilon^2 - 2\epsilon^2 \cos\left(\frac{2\pi}{n}\right)} = \lim_{n \rightarrow \infty} n \cdot \sqrt{2\epsilon^2 - 2\epsilon^2 \cos\left(\frac{2\pi}{n}\right)}$$

Si procede dunque con il dimostrare che

$$\begin{aligned} 2\pi\epsilon &= \lim_{n \rightarrow \infty} n \cdot \sqrt{2\epsilon^2 - 2\epsilon^2 \cos\left(\frac{2\pi}{n}\right)} \\ 2\pi\epsilon &= \lim_{n \rightarrow \infty} n \cdot \sqrt{2\epsilon^2 \left(1 - \cos\left(\frac{2\pi}{n}\right)\right)} \\ 2\pi\epsilon &= \lim_{n \rightarrow \infty} \sqrt{2\epsilon} \cdot n \cdot \sqrt{\left(1 - \cos\left(\frac{2\pi}{n}\right)\right)} \end{aligned}$$

Utilizzando l'identità

$$2\sin^2(x) = 1 - \cos(2x)$$

Si ottiene

$$2\pi\epsilon = \lim_{n \rightarrow \infty} \sqrt{2\epsilon} \cdot n \cdot \sqrt{2\sin^2 \frac{\pi}{n}}$$

$$2\pi\epsilon = \lim_{n \rightarrow \infty} 2\epsilon \cdot n \cdot \sqrt{\sin^2 \frac{\pi}{n}}$$

$$2\pi\epsilon = \lim_{n \rightarrow \infty} 2\epsilon \cdot n \cdot \left| \sin \frac{\pi}{n} \right|$$

Ma dato che

$$\pi > 0 \wedge n > 1 \Rightarrow 0 < \frac{\pi}{n} < \pi \Rightarrow \sin \frac{\pi}{n} > 0$$

È possibile quindi scrivere l'equazione come

$$2\pi\epsilon = \lim_{n \rightarrow \infty} 2\epsilon \cdot n \cdot \sin \frac{\pi}{n}$$

$$2\pi\epsilon = \lim_{n \rightarrow \infty} 2\epsilon \cdot \cancel{n} \cdot \frac{\sin \frac{\pi}{n}}{\frac{\pi}{n}} \cdot \frac{\pi}{\cancel{n}}$$

$$2\pi\epsilon = \lim_{n \rightarrow \infty} 2\epsilon \cdot \frac{\sin \frac{\pi}{n}}{\frac{\pi}{n}} \cdot \pi$$

$$2\pi\epsilon = \lim_{n \rightarrow \infty} 2\epsilon \cdot \pi$$

$$2\pi\epsilon = 2\pi\epsilon$$

L'uguaglianza $l(C_{(x,y)}) = c$ è così verificata.

Evoluzione della Regione

Sia $\Phi(t, \vec{p})$ la legge di evoluzione del sistema

$$\Phi(0, \vec{p}) = \vec{p} \quad (1.2)$$

Siano $\vec{c}_{k_t}(x, y)$ e $C_t(x, y)$ definiti come segue

$$\vec{c}_{k_t}(x, y) = \Phi(t, \vec{c}_k(x, y))$$

$$C_t(x, y) = \{\vec{c}_{k_t}(x, y) \mid k \in [0, n], \text{ con } n \in \mathbb{N}\}$$

Se per $l(C_{(x,y)})$ era possibile stabilire un valore esatto, per $l(C_{t,(x,y)})$ invece nella maggior parte non è possibile ricavare una soluzione analitica.

Sia

$$l(C_{t,(x,y)}) = c'(x, y)$$

$\mathcal{C}(t)$ è infine definito come

$$\mathcal{C}(t) = \frac{c'(x, y)}{c}$$

1.1.4 Esempi Illustrativi e Verifica

In questa sezione sono presenti alcuni esempi di $\mathcal{C}(t)$ applicato al modello Lotka-Volterra e al pendolo semplice con attrito.

Lotka-Volterra

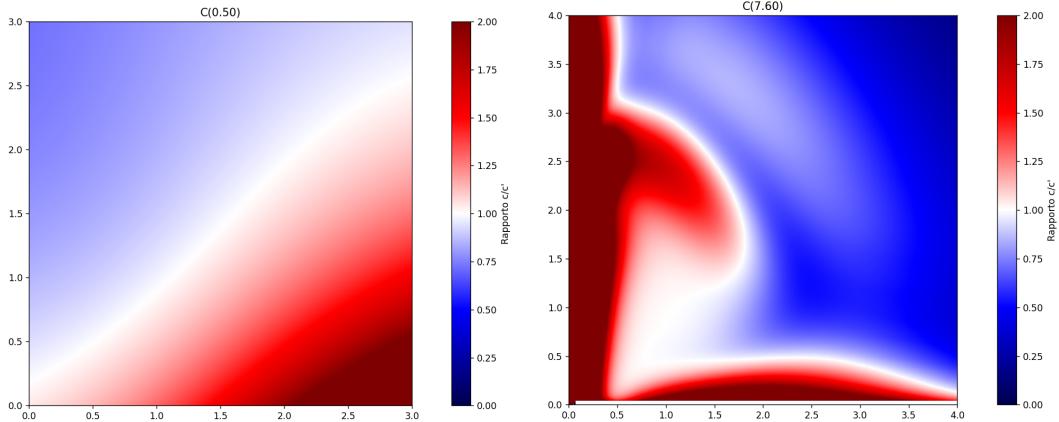


Figura 1.8: Campo scalare della stabilità per $t=0.5$.

Figura 1.9: Campo scalare della stabilità per $t=7.6$.

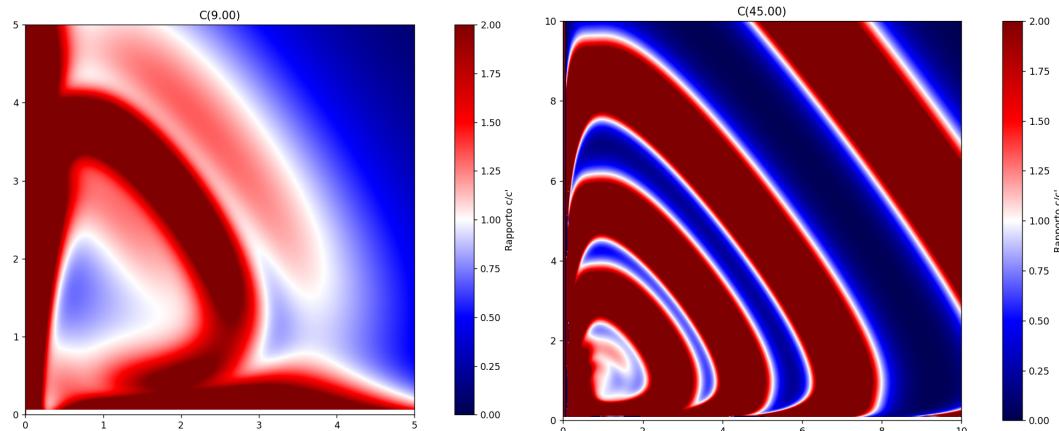


Figura 1.10: Campo scalare della stabilità per $t=9$.

Figura 1.11: Campo scalare della stabilità per $t=45$.

Il sistema dinamico sul quale è stato applicato l'operatore di stabilità è quello descritto dalle equazioni 1.1. Nella prima figura, dove il tempo di evoluzione è relativamente breve, si nota come i punti di maggiore instabilità sono quelli dove il numero di prede è molto superiore a quello dei predatori. In una situazione del genere la crescita delle prede è pressoché esponenziale e quindi una piccola variazione nel numero dei predatori influisce molto, anche nel breve termine.

Nella seconda figura invece si può notare come la parte di instabilità riguarda i punti dove il numero dei predatori è molto superiore a quello delle prede.

In effetti, con un tempo di evoluzione pari a 7.6, i punti in questione si spostano rapidamente verso l'origine, questo avviene perché non ci sono sufficienti prede per sfamare tutti i predatori e questo porta ad un calo nella popolazione dei predatori. Ora però il numero delle prede torna a crescere in maniera esponenziale, e avendo come unico fattore limitante la popolazione dei predatori ed essendo già questo molto ridotto, un qualsiasi cambiamento nel loro numero, influisce di molto sullo stato finale di quei punti.

Nelle ultime due immagini si nota l'evidente carattere ciclico del sistema.

Pendolo semplice

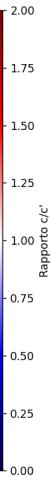
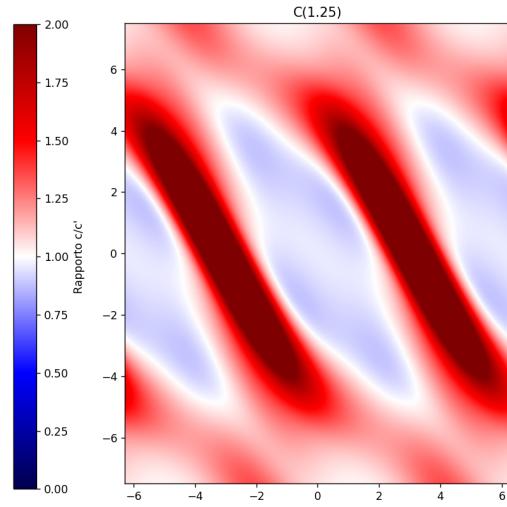
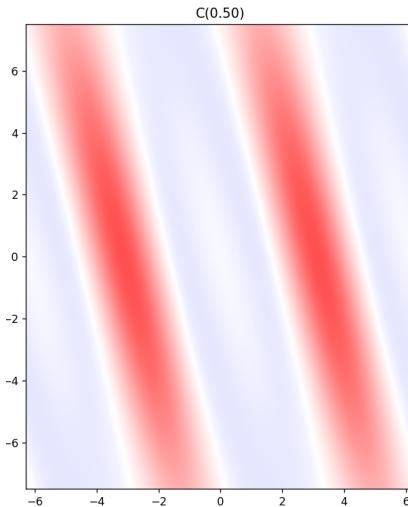


Figura 1.12: Campo scalare della stabilità per $t=0.5$.

Figura 1.13: Campo scalare della stabilità per $t=1.25$.

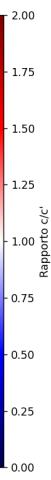
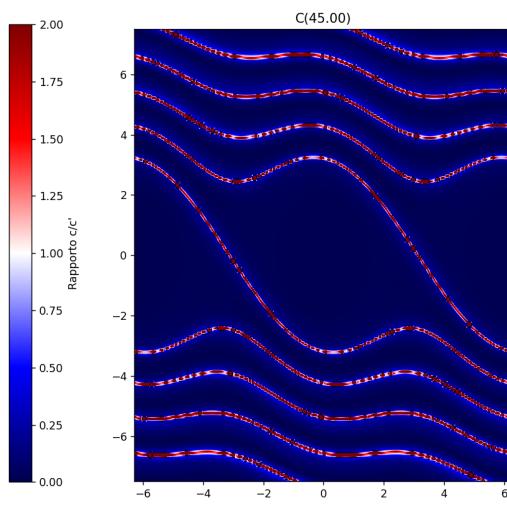
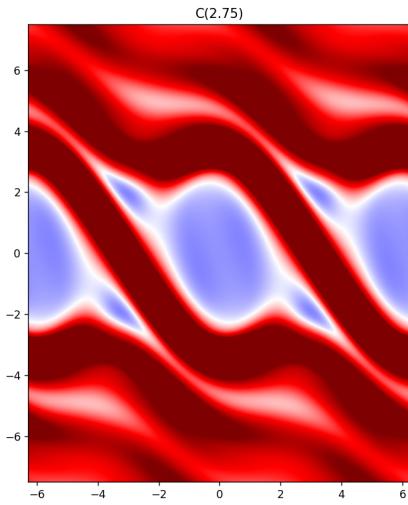


Figura 1.14: Campo scalare della stabilità per $t=2.75$.

Figura 1.15: Campo scalare della stabilità per $t=45$.

Come si può intuire dalle figure nel modello usato per descrivere il movimento di un pendolo semplice è stato introdotto un termine per simulare l'attrito⁸. Questo termine è proporzionale alla velocità, il sistema di equazioni differenziali utilizzato è il seguente

$$\begin{cases} \frac{d\theta}{dt} = \omega \\ \frac{d\omega}{dt} = -\frac{g}{L} \sin \theta - \gamma\omega \end{cases} \quad \text{con } \gamma = 0.2, L = 5 \quad (1.3)$$

Nella prima figura risaltano subito le regioni del sistema più instabili, sono quelle dove $\theta \approx \pm\pi$. Le zone sono oblique perché $\dot{\theta}$ riesce a bilanciare il pendolo, come verrà poi dimostrato nella sezione 2.3.1, portandolo molto vicino a $\pm\pi$, un punto di equilibrio instabile. I punti dove invece il sistema risulta essere più stabile sono quelli dove il pendolo ha una posizione e velocità tali che lo riportano alla posizione di equilibrio stabile. L'intervallo su cui viene visualizzato θ è $[-2\pi, 2\pi]$, che corrisponde ad un giro completo del pendolo verso destra e verso sinistra. Se si dovesse restringere il dominio a $[-\pi, \pi]$ allora si noterebbe che le zone più instabili sono quelle limitrofe agli estremi dell'intervallo.

Nella seconda e nella terza figura la regione di spazio stabile sembra ridursi, questo avviene perché il sistema ha più tempo per evolversi e differenziarsi sempre di più, ma non abbastanza per tornare alla posizione di equilibrio stabile.

Nell'ultima immagine il valore del tempo di evoluzione è sufficientemente grande per la maggior parte dei punti nel piano delle fasi di tornare nel punto di equilibrio stabile. Si notano però molte regioni di instabilità che si alternano a regioni di stabilità. Il primo punto di stabilità vicino a $(0, 4)$ è la configurazione del pendolo per la quale questo ha velocità sufficiente per completare un giro completo. In modo del tutto simile il punto vicino a $(0, 4.75)$ ha una velocità tale da permettere al pendolo di eseguire due giri completi. Tra questi due punti si trova una regione di instabilità, quella dove il pendolo può completare solamente un giro e mezzo, arrivando vicino ad una posizione di equilibrio instabile.

⁸Se non ci fosse attrito, per `t=45`, il campo non avrebbe valori così bassi.

Modello di Hodgkin-Huxley ridotto

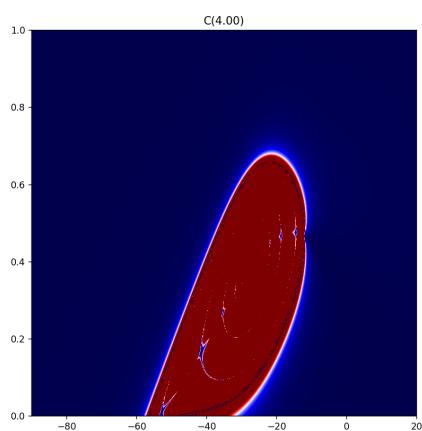


Figura 1.16: Campo scalare della stabilità per $t=4$. Il modello è lo stesso utilizzato per la figura 1.6.

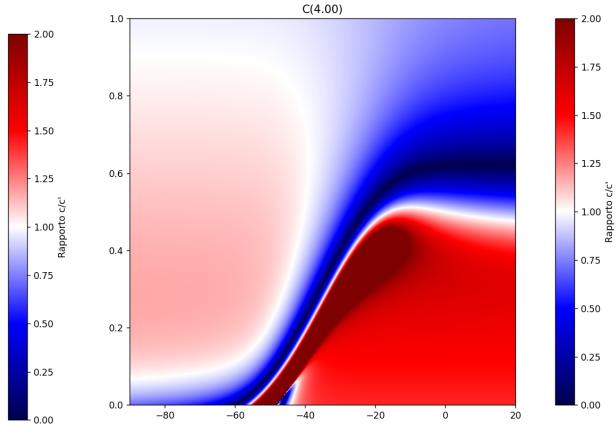


Figura 1.17: Stesso tempo di evoluzione, ma con parametri diversi, in particolare si tratta del modello rappresenta la biforcazione di Hopf supercritica.

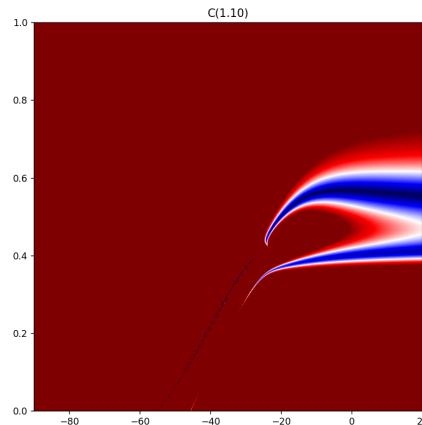


Figura 1.18: Campo scalare della stabilità per $t=1.1$. Il modello in questo caso rappresenta una biforcazione di tipo SNIC (*Saddle Node on Invariant Circle*).

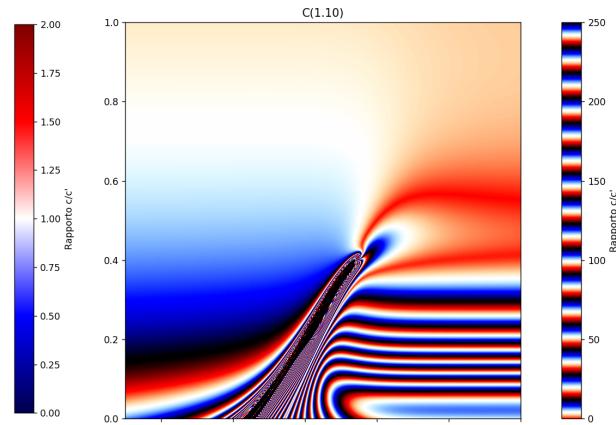


Figura 1.19: Stesso modello dell'immagine precedente, è stata modificata solo la scala di colori. Questa rivela una complessità che prima rimaneva nascosta.

È evidente la chiara divisione tra la regione stabile e instabile nella figura 1.16. Nell'immagine successiva viene utilizzato invece il modello con la biforcazione di Hopf supercritica. Una biforcazione è un punto critico dove la stabilità di un sistema cambia al variare di uno o più parametri. In questo caso varia I_{ext} , un parametro che rappresenta una corrente esterna applicata al neurone, una sorta di input elettrico arbitrario. Aumentando questo valore si può simulare ad esempio che il neurone stia ricevendo continuamente dei segnali elettrici da altri neuroni. Questa variazione nei parametri modifica in modo qualitativo il comportamento del neurone, si potrebbe passare, ad esempio, da un compor-

tamento del tipo di singolo picco ad uno che porta il neurone a produrre una serie di picchi.

Al di là dei dettagli tecnici relativi al modello, si nota che nell'ultima figura i colori non hanno alcun significato particolare e servono semplicemente ad evidenziare la natura estremamente complessa di questo modello. In questo caso i valori di $\mathcal{C}(1.1)$ variano moltissimo e risulta difficile darne una rappresentazione che mantenga tutte le informazioni. L'utilità dell'operazione di *clipping* in questo caso è particolarmente dubbia, questa va infatti a nascondere moltissimi dettagli.

1.2 Metodi di Discretizzazione

Come già accennato in precedenza, risolvere $\mathcal{C}(t)$ in maniera del tutto analitica è spesso un compito impossibile. $\mathcal{C}(t)$ ha una difficoltà di risoluzione superiore a quella di trovare la legge di evoluzione del sistema in sé semplicemente perché la risoluzione di $\Phi(t, \vec{p})$, ovvero la soluzione al sistema di equazioni differenziali, è implicata dall'operatore stesso.

Per calcolare la lunghezza della circonferenza deformata infatti è necessario prima risolvere il sistema di equazioni differenziali per ricavarne le soluzioni e successivamente svolgere il seguente integrale:

$$\int \sqrt{1 + \left(\frac{dy(t)}{dx(t)}\right)^2} dx$$

Risolvere un integrale di questo tipo è molto complicato anche nel caso in cui il quadrato delle derivate sia una funzione relativamente semplice, la presenza della radice complica il calcolo notevolmente. Per questo motivo, al fine di calcolare effettivamente $\mathcal{C}(t)$, si rende necessario discretizzare l'operatore. Così facendo è inoltre possibile sfruttare l'eccezionale potenza computazionale dei moderni *hardware*, che possono eseguire anche decine di miliardi di operazioni al secondo.

In questa sezione, oltre a mostrare come vengono definiti i parametri necessari per calcolare l'operatore di stabilità, viene mostrato l'algoritmo utilizzato per integrare le equazioni del sistema. Inoltre verrà analizzata brevemente la differenza tra la soluzione esatta e la soluzione approssimata al variare di dt . Successivamente si calcola l'impegno computazionale dell'operatore e la complessità dello stesso espressa tramite la notazione O-grande. Infine viene calcolato analiticamente il valore di $\mathcal{C}(t)$ di un sistema dinamico estremamente semplice.

1.2.1 Definizione del Sistema Dinamico

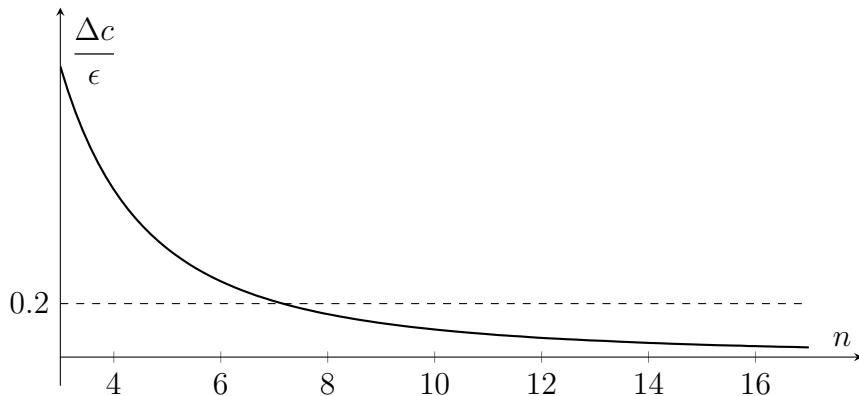
In primo luogo è necessario rendere discreto lo spazio sul quale viene applicato l'operatore. Per fare questo si stabilisce prima di tutto un intervallo nelle due

direzioni che corrispondono ai due gradi di libertà del modello, successivamente si distribuiscono linearmente p punti negli intervalli scelti. La distanza minima tra due punti è data da

$$d_m = \frac{\min(L_x, L_y)}{p - 1}$$

Dove L_x e L_y corrispondono al modulo della differenza degli estremi del dominio lungo le rispettive direzioni.

Viene poi resa discreta anche la circonferenza di centro (x, y) generata da \vec{c}_k : invece di utilizzare $n \rightarrow \infty$ si pone n uguale ad numero finito, non è necessario che sia eccessivamente grande, è difficile notare differenze significative per $n > 10$. La differenza tra il valore esatto di c e quello approssimato da un poligono regolare di n lati è mostrata nel grafico che segue:



Dato che il perimetro di un poligono regolare di n lati inscritto nella circonferenza unitaria è dato da:

$$2p = 2n \sin\left(\frac{\pi}{n}\right)$$

Si può ricavare la funzione che esprime la differenza tra il valore esatto e l'approssimazione:

$$\frac{\Delta c}{\epsilon} = 2\pi - 2n \sin\left(\frac{\pi}{n}\right)$$

$$\Delta c = 2\pi\epsilon - 2n\epsilon \sin\left(\frac{\pi}{n}\right)$$

È importante comunque ricordare che Δc dipende da ϵ , un valore molto piccolo. Per $n = 7$, ad esempio, $\Delta c \approx 0.2\epsilon$.

Oltre a n anche ϵ viene sostituito con un numero finito molto piccolo. Generalmente $\epsilon = 0.0001$ offre risultati sufficientemente precisi, ma ovviamente questo dipende da d_m , l'ideale sarebbe utilizzare questi parametri in modo che $\epsilon \propto d_m$, come ad esempio $\epsilon = \frac{d_m}{10}$ o qualcosa di simile.

Segue il confronto fra due campi scalari della stabilità relativi al modello di Hodgkin-Huxley, la prima generata con un valore di ϵ (il corrispondente di ϵ

nell'implementazione in Python) sufficientemente piccolo, la seconda invece è stata generata con un valore di r decisamente troppo grande:

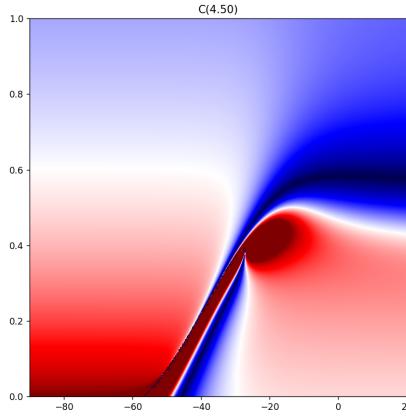


Figura 1.20: Campo scalare della stabilità con $r=0.000001$.

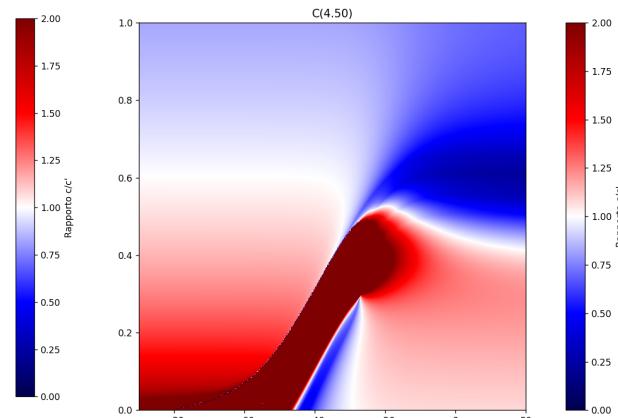


Figura 1.21: Campo scalare della stabilità con $r=0.1$.

Come si può vedere, utilizzando un valore di r eccessivamente grande il campo risulta distorto e presenta artefatti. La struttura generale del campo in questo caso è preservata; questo fatto non vale però in generale, dipende completamente dal tipo di sistema dinamico considerato.

1.2.2 Integrazione nel Sistema Dinamico

Anche l'integrazione dei singoli punti viene resa discreta. Se per esempio il sistema è descritto dalle equazioni

$$\begin{cases} x' = -y \\ y' = x \end{cases}$$

e si vuole calcolare la posizione di $(2, 3)$ dopo $t = 5$ con $dt = 10^{-4}$ allora si potrà calcolare dx e dy scrivendo il sistema come:

$$\begin{cases} dx = -y \cdot 10^{-4} \\ dy = x \cdot 10^{-4} \end{cases}$$

Sommendo poi dx e dy a $(2, 3)$ ed eseguendo questo passaggio $\frac{t}{dt}$ volte si ottiene la posizione finale del punto. Ovviamente al diminuire di dt diminuisce anche la differenza tra la soluzione approssimata e quella esatta.

In questo caso la soluzione esatta si può ricavare abbastanza semplicemente risolvendo l'equazione differenziale che si ottiene dal sistema

$$\begin{aligned}y' &= x \\y'' &= x' \\y'' &= -y \\y'' + y &= 0\end{aligned}$$

Il cui polinomio caratteristico è

$$P(\lambda) = \lambda^2 + 1$$

Le cui radici sono

$$\lambda = \pm i$$

Poiché le soluzioni sono complesse coniugate nella forma

$$\alpha \pm i\beta \quad \text{con} \quad \alpha = 0, b = 1$$

La soluzione è del tipo

$$\begin{aligned}y(t) &= c_1 e^{\alpha t} \cos(\beta t) + c_2 e^{\alpha t} \sin(\beta t) \\y(t) &= c_1 \cos(t) + c_2 \sin(t)\end{aligned}$$

Per trovare $x(t)$ è sufficiente calcolare la derivata di $y(t)$ dato che $y' = x$

$$y'(t) = x(t) = c_2 \cos(t) - c_1 \sin(t)$$

Per $t = 0$

$$y(0) = c_1 = 3$$

$$x(0) = c_2 = 2$$

Quindi le equazioni soluzione del sistema sono

$$\begin{cases}x(t) = 2 \cos(t) - 3 \sin(t) \\y(t) = 3 \cos(t) + 2 \sin(t)\end{cases}$$

Il punto $(2, 3)$ dopo $t = 5$ avrà dunque coordinate

$$(2 \cos 5 - 3 \sin 5, 3 \cos 5 + 2 \sin 5) \approx (3.44, -1.07)$$

Conoscendo dunque il risultato esatto è possibile analizzare quanto questo differisca dal risultato ottenuto tramite integrazione discreta

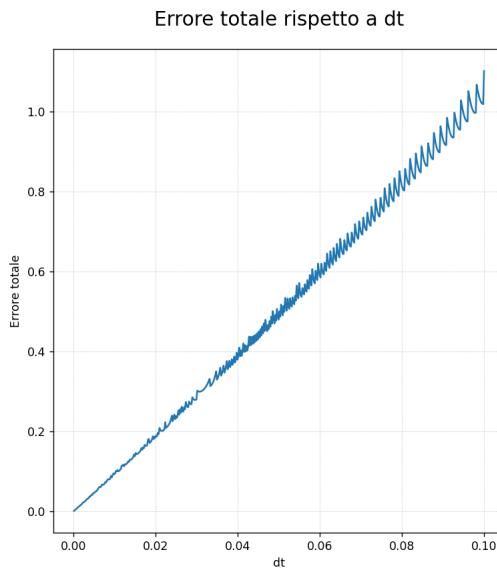


Figura 1.22: Errore dell'approssimazione di $\Phi[5, (2, 3)]$ per $dt \in [10^{-4}, 10^{-1}]$

Per $dt = 10^{-4}$ l'errore è uguale a 0.00097.

Il metodo di approssimazione descritto in breve sopra corrisponde al metodo di integrazione di Eulero

$$y_{n+1} = y_n + dt f(t_n, y_n)$$

Questo metodo però può essere usato solamente quando è possibile ricavare

$$\frac{dy}{dt} = f(t, y)$$

Se l'equazione differenziale non permette di calcolare le derivate esplicite per le variabili di stato allora questo metodo non è adoperabile. Inoltre, come si può vedere dal grafico, il metodo di Eulero ha un errore globale di ordine $\mathcal{O}(\Delta t)$, che necessita quindi l'utilizzo di un passo molto piccolo per ottenere dei risultati precisi. Poiché si era detto nel paragrafo 1.1.1 che i sistemi considerati sono definiti tramite le derivate dei due gradi di libertà del sistema, il problema che rimane è quello della precisione del metodo. Nonostante esistano molti altri metodi, come quello di Runge-Kutta, che offrono una precisione molto maggiore, il metodo di Eulero ha la caratteristica di essere estremamente parallelizzabile in virtù della sua semplicità strutturale e dell'indipendenza dei calcoli che l'operatore deve svolgere.

Il problema principale rimane comunque quello della propagazione dell'errore. In sistemi dinamici che presentano elementi di non linearità o che sono caotici il metodo di Eulero diventa sempre meno adatto all'aumentare del tempo di integrazione. Si prenda per esempio il sistema:

$$\begin{cases} x' = 10x \\ y' = 1 \end{cases}$$

Le cui soluzioni sono:

$$\begin{aligned}
 x' &= 10x \\
 \frac{dx}{x} &= 10dt \\
 \int \frac{dx}{x} &= \int 10dt \\
 \ln|x| &= 10t + C_1 \\
 e^{\ln|x|} &= C_1 e^{10t} \\
 x(t) &= C_1 e^{10t}
 \end{aligned}
 \quad
 \begin{aligned}
 y' &= 1 \\
 \int dy &= \int dt \\
 y(t) &= t + C_2
 \end{aligned}$$

Per il punto $(2, 3)$ le soluzioni diventano:

$$\begin{cases} x(0) = C_1 = 2 \\ y(0) = C_2 = 3 \end{cases} \Rightarrow \begin{cases} x(t) = 2e^{10t} \\ y(t) = t + 3 \end{cases} \Rightarrow \Phi(5, (2, 3)) = (2e^{50}, 8)$$

Nella seguente tabella sono riportati gli errori assoluti dell'incremento dt confrontando la soluzione esatta con il metodo di Eulero con una semplice differenza:

$|\text{soluzione esatta} - \text{approssimazione}|$

Incremento	Errore Eulero
10^{-1}	2.137×10^{13}
10^{-2}	1.562×10^{13}
10^{-3}	2.775×10^{12}
10^{-4}	3.180×10^{11}
10^{-5}	3.203×10^{10}

L'errore in questo caso è molto grande nonostante il tempo di integrazione sia relativamente piccolo. Metodi più complessi, come RK4 (Runge-Kutta di quarto ordine), sono di gran lunga più precisi rispetto al metodo di Eulero.

Ciononostante per generare le immagini del campo scalare è stato sempre utilizzato il metodo di Eulero perché per eseguire i calcoli è necessario svolgere la stessa identica operazione di integrazione per ogni punto nel dominio, dove l'integrazione di un certo punto ha la caratteristica di non influenzare l'integrazione in un altro punto. È proprio grazie a questa inter-indipendenza computazionale che è possibile sfruttare la potenza di calcolo delle GPU (*Graphic Processing Unit*), hardware altamente specializzate nell'elaborazione di grandissime quantità di operazioni elementari, per eseguire questi calcoli in tempi brevissimi. Questo aspetto sarà discusso in maniera più approfondita nel paragrafo 2.1.3.

1.2.3 Impegno Computazionale

In questa sezione viene analizzato come varia l'impegno computazionale totale per calcolare il campo scalare al variare dei parametri che lo definiscono. In particolare viene calcolato il numero di operazioni che è necessario svolgere per elaborare il campo scalare in base a suddetti parametri.

Stabilito il dominio di integrazione, il numero totale di punti è pari a $p \cdot p$, dove p corrisponde al numero di vertici per lato. Un incremento lineare nella densità dei vertici nei lati della griglia corrisponde ad un aumento quadratico del numero di operazioni. Probabilmente nel caso in cui vengano utilizzate dimensioni diverse per il dominio sull'asse orizzontale e verticale, sarebbe meglio distribuire i punti in modo che

$$d_m = \frac{L_x}{p_x - 1} = \frac{L_y}{p_y - 1}$$

Dove p_x e p_y corrispondono al numero di punti sui rispettivi assi. In questo modo la distanza tra i punti sarebbe la stessa in entrambe le direzioni.

Ad ogni punto è poi associata una regione. Ogni regione corrisponde ad un poligono regolare di n lati. Per ogni punto bisogna calcolare l'evoluzione di n stati. Il numero totale di punti da integrare nel sistema sale quindi a $n \cdot p^2$.

Ogni punto deve essere integrato secondo il sistema di equazioni differenziali che definisce il sistema. Il numero di passi necessari per integrare un singolo punto equivale al rapporto tra il tempo totale di integrazione e il passo temporale

$$n_{dt} = \left\lceil \frac{t_{tot}}{dt} \right\rceil = \left\lceil \frac{t_f - t_0}{dt} \right\rceil$$

Questo porta il numero totale di operazioni a

$$\boxed{n_{dt} \cdot n \cdot p^2} \quad (1.4)$$

Notazione O-grande

Per esprimere la complessità dell'operatore si può utilizzare la notazione O-grande, introdotta da Paul Bachmann nel 1894. La definizione formale è la seguente

Siano f e g due funzioni definite su qualche sottoinsieme dei numeri reali. Diciamo che

$$f(x) \in O[g(x)] \text{ per } x \rightarrow \infty$$

se e solo se

$$\exists x_0 \in \mathbb{R}, c > 0 : |f(x)| \leq c|g(x)|, \quad \forall x > x_0$$

In questo caso la funzione $f(x)$, che esprime la complessità dell'operatore di stabilità, si può ricavare dall'equazione 1.4. Ogni parametro contribuisce essenzialmente ad aumentare la complessità dell'algoritmo, infatti:

$$f(x) = cx \cdot x \cdot x^2 = cx^4$$

La prima x rappresenta il numero di passi di integrazione (la costante c tiene semplicemente conto del tempo totale di evoluzione del sistema, corrisponde alla variabile t in $\mathcal{C}(t)$, l'operatore viene valutato per un singolo valore di t , c dunque non tende all'infinito), la seconda il numero di vertici dei poligoni e x^2 il numero di punti su cui calcolare $\mathcal{C}(t)$.

Si può infine concludere che

$$f(x) \in O(x^4)$$

In parole, l'operatore di stabilità ha una complessità in tempo dell'ordine di x^4 .

Tutte le immagini nel documento sono state elaborate con:

$$\begin{aligned} p &\geq 250 & n &\geq 25 \\ dt &\leq 0.001 & t &\leq 45 \end{aligned}$$

Il numero totale di operazioni per la figura 1.15 ad esempio è:

$$\left\lceil \frac{45}{0.001} \right\rceil \cdot 30 \cdot 350^2 = 165375000000 = 1.65 \times 10^{11}$$

In cui ogni operazione corrisponde ad eseguire il calcolo:

$$\begin{cases} x_{n+1} = x_n + y \cdot dt \\ y_{n+1} = y_n + \left(-\frac{9.81}{5} \cdot \sin x - 0.2y \right) \cdot dt \end{cases}$$

Il tempo stimato per eseguire queste operazioni con una CPU⁹ (*Central Processing Unit*) è di circa 44 ore. Questo valore è ottenuto dalla media di una serie di previsioni che eseguono solamente una parte del numero totale di operazioni, ma questo viene compensato moltiplicando il tempo parziale ottenuto per il fattore di riduzione delle operazioni. Ad esempio si potrebbe calcolare il tempo per elaborare 1.65×10^5 operazioni e poi moltiplicare il tempo ottenuto per 10^6 .

Eseguire questo tipo di operazioni in serie è semplicemente insostenibile. Utilizzando invece una GPU¹⁰ e parallelizzando i calcoli, il tempo necessario per svolgere la stessa operazione si riduce a meno di due minuti.

Al di là di come viene calcolato esattamente il tempo stimato per la CPU, che dipende altamente dal tipo di processore utilizzato, la cosa importante da notare è l'enorme differenza in tempo di calcolo tra una CPU e una GPU nell'eseguire queste operazioni.

⁹La CPU utilizzata è *AMD Ryzen 5 5600H Radeon Graphics*, non è stata utilizzata la tecnica del *multithreading*.

¹⁰La GPU utilizzata è *NVIDIA GeForce RTX 3050 Laptop GPU*.

NVIDIA Nsight Compute

Attraverso il software *NVIDIA Nsight Compute* (NNC) è possibile analizzare e misurare esattamente le prestazioni e l'esecuzione delle operazioni allocate alle singole componenti della GPU.

Per poter comprendere le metriche generate da NNC è necessario chiarire alcuni concetti alla base della computazione in parallelo eseguita su una GPU: la funzione, o blocco di istruzioni, che si vuole calcolare è detta *kernel*. Il lavoro totale del *kernel* viene diviso in vari *Streaming Multiprocessors* (SM), delle sub-unità interne alla GPU che svolgono i calcoli definiti dal *kernel* contemporaneamente, in parallelo. Un ciclo (o *cycle* in inglese) non è altro che un'unità di misura temporale che corrisponde all'intervallo di tempo che trascorre tra due computazioni negli SM.

Dal file di *profiling* generato grazie a NNC è possibile ricavare ad esempio la metrica `smsp__inst_executed.sum`. Questa metrica contiene il numero di istruzioni eseguite in totale. In questo caso vale 2.5×10^{11} . Il fatto che questo valore sia maggiore di quello teorico calcolato in precedenza è normale poiché questa metrica dipende dal numero totale di istruzioni, che non sono necessariamente matematiche e che dipendono dalla struttura del *kernel* stesso.

Un'altra metrica interessante è quella che misura quanta informazione è stata processata in termini di byte. In questo caso la somma di tutte le operazioni di scrittura e lettura, globale e locale è di 1.570 GB. Questa cifra rappresenta una quantità enorme di dati, che corrisponde all'incirca alla quantità di informazione che si avrebbe con più di 300 film in full HD. Grazie alle efficienti prestazioni delle moderne GPU è possibile processare più di 6.5 GB di informazione ogni secondo.

1.2.4 Esempio di Calcolo

In questo paragrafo viene calcolato analiticamente $\mathcal{C}(2)$ in $(0, 0)$ di un sistema dinamico molto semplice.

Il sistema in questione è il seguente:

$$\begin{cases} x' = 1 \\ y' = 0 \end{cases}$$

Per prima cosa è necessario trovare le funzioni $x(t)$ e $y(t)$ che descrivono la posizione di un punto al variare di t :

$$\begin{aligned} \begin{cases} \frac{dx}{dt} = 1 \\ \frac{dy}{dt} = 0 \end{cases} & \quad \begin{cases} \int dx = \int dt \\ \int dy = \int 0 \end{cases} \\ \begin{cases} dx = dt \\ dy = 0 \end{cases} & \quad \begin{cases} x = t + C_1 \\ y = C_2 \end{cases} \end{aligned}$$

Le soluzioni generali sono dunque:

$$\begin{cases} x(t) = t + C_1 \\ y(t) = C_2 \end{cases}$$

Per trovare le costanti è sufficiente calcolare $x(0)$ e $y(0)$:

$$\begin{cases} x(0) = x_0 = C_1 \\ y(0) = y_0 = C_2 \end{cases}$$

Quindi:

$$\Phi(t, \vec{p}) = \begin{cases} x(t) = x_0 + t \\ y(t) = y_0 \end{cases}$$

Si può ora trovare l'equazione della circonferenza γ' per $t = 2$

$$\gamma : x^2 + y^2 = \epsilon^2$$

$$\begin{cases} x(t) = x_0 + t \\ y(t) = y_0 \end{cases} \rightarrow \begin{cases} x_0 = x - 2 \\ y_0 = y \end{cases}$$

$$\gamma' : (x - 2)^2 + y^2 = \epsilon^2$$

Come dimostrato nella sezione 1.1.3 $c = 2\pi\epsilon$. Per calcolare $\mathcal{C}(2)$ è necessario trovare la lunghezza della curva γ' .

In effetti il sistema dinamico non fa altro che traslare verso destra i punti della circonferenza senza deformarli, è una trasformazione isometrica, sarebbe sufficiente applicare una traslazione per riportare il centro della circonferenza nell'origine per poi concludere che il rapporto tra c e c' è 1. Tuttavia, poiché lo scopo di questa sezione è quello di mostrare che trovare una soluzione analitica è abbastanza complesso anche in un caso estremamente banale, si utilizza ora il metodo descritto brevemente nell'introduzione della sezione 1.2.

Per trovare la lunghezza della circonferenza γ' utilizzando l'integrale di una curva è necessario ricavare una funzione differenziabile. Dato che γ' non è

altro che una circonferenza traslata verso destra si può ricavare la funzione che descrive la semicirconferenza superiore:

$$f(x) = \sqrt{\epsilon^2 - (x - 2)^2}$$

Per calcolare la lunghezza della circonferenza è sufficiente moltiplicare per 2 la lunghezza della semicirconferenza. La circonferenza inferiore ha equazione:

$$g(x) = -\sqrt{\epsilon^2 - (x - 2)^2}$$

Le loro derivate sono:

$$f'(x) = \frac{-(x - 2)}{\sqrt{\epsilon^2 - (x - 2)^2}} \quad g'(x) = \frac{x - 2}{\sqrt{\epsilon^2 - (x - 2)^2}}$$

I due integrali di linea sono equivalenti:

$$\begin{aligned} \int \sqrt{1 + [f'(x)]^2} dx &= \int \sqrt{1 + [g'(x)]^2} dx \\ \int \sqrt{1 + \left[\frac{-(x - 2)}{\sqrt{\epsilon^2 - (x - 2)^2}} \right]^2} dx &= \int \sqrt{1 + \left[\frac{x - 2}{\sqrt{\epsilon^2 - (x - 2)^2}} \right]^2} dx \\ \int \sqrt{1 + \frac{(x - 2)^2}{\epsilon^2 - (x - 2)^2}} dx &= \int \sqrt{1 + \frac{(x - 2)^2}{\epsilon^2 - (x - 2)^2}} dx \end{aligned}$$

È possibile concludere che:

$$\begin{aligned} c' &= \int_a^b \sqrt{1 + [f'(x)]^2} dx + \int_a^b \sqrt{1 + [g'(x)]^2} dx \\ &= \int_a^b \sqrt{1 + [f'(x)]^2} dx + \int_a^b \sqrt{1 + [f'(x)]^2} dx \\ &= 2 \cdot \int_a^b \sqrt{1 + [f'(x)]^2} dx \end{aligned}$$

Il dominio è:

$$D = 2 - \epsilon \leq x \leq 2 + \epsilon$$

Il valore esatto di c' si può dunque calcolare come:

$$c' = 2 \cdot \int_{2-\epsilon}^{2+\epsilon} \sqrt{1 + \left[\frac{-(x - 2)}{\sqrt{\epsilon^2 - (x - 2)^2}} \right]^2} dx$$

L'integrale è improprio, infatti negli estremi dell'integrale il denominatore si annulla, sono presenti due asintoti verticali. È possibile semplificare ulteriormente l'integrale notando che la funzione integranda è simmetrica rispetto alla retta $x = 2$:

$$f(x) = f(-x + 2 \cdot 2)$$

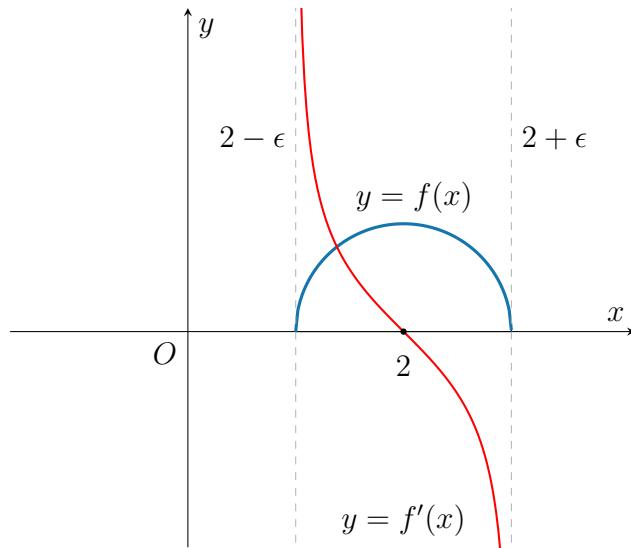
$$\begin{aligned} \sqrt{1 + \left[\frac{-(x-2)}{\sqrt{\epsilon^2 - (x-2)^2}} \right]^2} &= \sqrt{1 + \left[\frac{-(-x+4-2)}{\sqrt{\epsilon^2 - (-x+4-2)^2}} \right]^2} \\ 1 + \left[\frac{-(x-2)}{\sqrt{\epsilon^2 - (x-2)^2}} \right]^2 &= 1 + \left[\frac{x-2}{\sqrt{\epsilon^2 - (2-x)^2}} \right]^2 \\ \frac{(x-2)^2}{\epsilon^2 - (x-2)^2} &= \frac{(x-2)^2}{\epsilon^2 - (2-x)^2} \end{aligned}$$

In virtù di questa simmetria è possibile calcolare c' come:

$$c' = 4 \cdot \int_2^{2+\epsilon} \sqrt{1 + \left[\frac{-(x-2)}{\sqrt{\epsilon^2 - (x-2)^2}} \right]^2} dx$$

È comunque presente una discontinuità in $x = 2 + \epsilon$, che rende l'integrale improprio di seconda specie, ed è dunque valida la seguente uguaglianza:

$$4 \int_2^{2+\epsilon} \sqrt{1 + \left[\frac{-(x-2)}{\sqrt{\epsilon^2 - (x-2)^2}} \right]^2} dx = 4 \lim_{b \rightarrow 2+\epsilon^-} \int_2^b \sqrt{1 + \left[\frac{-(x-2)}{\sqrt{\epsilon^2 - (x-2)^2}} \right]^2} dx$$



$$\begin{aligned}
c' &= 4 \int_2^{2+\epsilon} \sqrt{1 + \left[\frac{-(x-2)}{\sqrt{\epsilon^2 - (x-2)^2}} \right]^2} dx \\
&= 4 \int_2^{2+\epsilon} \sqrt{1 + \frac{(x-2)^2}{\epsilon^2 - (x-2)^2}} dx \\
&= 4 \int_2^{2+\epsilon} \sqrt{\frac{(x-2)^2 + \epsilon^2 - (x-2)^2}{\epsilon^2 - (x-2)^2}} dx \\
&= 4 \int_2^{2+\epsilon} \sqrt{\frac{\epsilon^2}{\epsilon^2 - (x-2)^2}} dx \\
&= 4\epsilon \int_2^{2+\epsilon} \frac{dx}{\sqrt{\epsilon^2 - (x-2)^2}}
\end{aligned}$$

Ponendo $u = x - 2$ quindi $dx = du$

Se $x = 2 \rightarrow u = 0$

Se $x = 2 + \epsilon \rightarrow u = \epsilon$

$$\begin{aligned}
&= 4\epsilon \int_0^\epsilon \frac{du}{\sqrt{\epsilon^2 - u^2}} \\
&= 4\epsilon \lim_{b \rightarrow \epsilon^-} \left[\arcsin \frac{u}{\epsilon} \right]_0^b \\
&= 4\epsilon \left(\arcsin \frac{\epsilon}{\epsilon} - \arcsin \frac{0}{\epsilon} \right) \\
&= 4\epsilon [\arcsin(1) - \arcsin(0)]
\end{aligned}$$

$$= 4\epsilon \left(\frac{\pi}{2} + 0 \right)$$

$$= 2\pi\epsilon$$

$$\mathcal{C}(2) = \frac{c'}{c} = \frac{2\pi\epsilon}{2\pi\epsilon} = 1$$

È evidente la difficoltà di risolvere $\mathcal{C}(t)$ per sistemi dinamici più complessi, anche se non presentano elementi di non linearità. Inoltre si noti che quello calcolato è sì il valore di $\mathcal{C}(2)$ del campo scalare, ma il fatto che il sistema descriva semplicemente una traslazione, e quindi che $\mathcal{C}(2)$ sia costante, ha permesso essenzialmente di ricondurre il calcolo dell'intero campo a quello di un singolo punto.

Implementazione

Nella prima sezione di questo capitolo viene mostrato come l'operatore di stabilità può essere calcolato e visualizzato con Python, inoltre verrà discusso in modo più approfondito della tecnologia CUDA e di come viene usata nello specifico per aumentare l'efficienza di calcolo.

Nella seconda sezione verranno mostrati altri esempi e visualizzazioni dell'operatore, facendo particolare riferimento al modello ridotto di Hodgkin-Huxley e al modello competitivo di Lotka Volterra, una variazione del modello di Lotka-Volterra che modella la popolazione di due specie in competizione per una risorsa comune. Infine verrà discusso dell'applicazione creata appositamente per poter interagire in tempo reale con l'operatore di stabilità. Questa applicazione permette infatti all'utente di navigare nel campo scalare attraverso traslazioni, omotetie e dilatazioni. Il fatto che l'intero codice sorgente dell'applicazione sia scritto in C++ e utilizzi, come nel codice Python, il calcolo in parallelo attraverso delle implementazioni di CUDA, permette all'applicazione di elaborare il campo scalare in una frazione di secondo.

Infine, nella terza ed ultima sezione, viene analizzata una funzione legata all'operatore di stabilità che utilizza essenzialmente gli stessi concetti descritti nel capitolo precedente, ma elaborati in modo tale da offrire una visione forse più utile, ma più specifica, riguardo ad una precisa configurazione. Questa funzione viene osservata nel modello del pendolo e quello di Lotka-Volterra.

2.1 Implementazione in Python

Inizialmente avrei voluto utilizzare C++ come linguaggio per risolvere discretamente l'operatore di stabilità, data la sua notoria velocità di calcolo. Rispetto a Python, per esempio, nel conteggio fino a un miliardo, C++ impiega circa 20 millisecondi, mentre Python 1 minuto e 20 secondi. Purtroppo a causa di difficoltà tecniche derivanti l'utilizzo di alcune librerie specifiche per interagire con la GPU ho optato per sviluppare la quasi totalità degli script in Python.

Sviluppare un'applicazione che interagisca con una GPU in C++ non è un compito molto difficile di per sé, ma non ho né le conoscenze né le competenze

tecniche specifiche per svolgere un lavoro di questo tipo in autonomia. Ho già avuto modo di lavorare in C++, ma per compiti decisamente più semplici e non senza problemi. Solamente in data 15/01 sono riuscito, o meglio, Gemini Experimental¹ è riuscito a risolvere questi problemi.

A differenza di quanto avevo ipotizzato inizialmente, la differenza di elaborazione del campo scalare tra i due linguaggi è notevole: se in Python una singola immagine impiega circa due minuti per essere elaborata, in C++, utilizzando gli stessi parametri, il tempo si riduce a circa 5 secondi. Diminuendo alcuni parametri, come il numero di vertici per poligono e/o la risoluzione della griglia e utilizzando alcune tecniche di ottimizzazione, è comunque possibile raggiungere stabilmente 10-20 FPS (*frame per second*). Nonostante questo Python rimane comunque un'ottima scelta per *script* di *testing* per via della semplicità del linguaggio (se confrontato con C++) e del tempo necessario per impostare l'ambiente di esecuzione del codice.

In questa sezione viene mostrato com'è possibile definire un sistema dinamico in Python, gestirne l'integrazione numerica (già discussa brevemente nella sezione 1.2.2) per trovare le traiettorie o il relativo spazio delle fasi e visualizzarlo con vari metodi. Infine viene approfondito l'ultimo paragrafo della sezione 1.2.3 relativo a NNC.

NOTA: parte del codice riportato in questa sezione è stato generato da un'intelligenza artificiale. Le funzioni `lambda` e la gestione degli `array` in `numpy` ne sono un esempio.

2.1.1 Definizione del Sistema Dinamico

I parametri che gestiscono il sistema dinamico sono i seguenti:

```
equal = False
domain_x = [-90,20]
domain_y = [0,1]
num_sides = 30
radius = 0.000001
grid_size = 250
dt = 0.0001
t = 1.1
```

La variabile `equal` in caso sia impostata su `True` rende il dominio verticale del sistema dinamico uguale a quello orizzontale, non è particolarmente rilevante, è solamente una comodità per gestire il dominio velocemente quando è necessario cambiare tra vari sistemi.

```
if equal:
    domain_y = domain_x
```

¹ `gemini-2.0-flash-exp` e `gemini-exp-1206` sono gli autori di circa il 92% del codice sorgente dell'applicazione.

`domain_x` e `domain_y` determinano il dominio sull'asse orizzontale e sull'asse verticale, mentre `grid_size` corrisponde al numero di vertici per lato su cui viene calcolato $\mathcal{C}(t)$.

`num_sides` invece è la variabile che determina il numero di lati del poligono regolare che approssima la circonferenza, viene utilizzato insieme a `radius` dalla funzione `generate_polygon(center, radius, num_sides)` per generare la lista di punti corrispondente ai vertici di tale poligono di centro `center`. `center` è una lista di punti generata nel seguente modo

```
x_centers = np.linspace(domain_x[0], domain_x[1], grid_size)
y_centers = np.linspace(domain_y[0], domain_y[1], grid_size)
centers = np.array(np.meshgrid(x_centers, y_centers)).T.reshape(-1, 2)
```

`np.linspace` distribuisce in modo lineare `grid_size` punti dal primo al secondo argomento, `np.array` e gli altri metodi assicurano il corretto formato dell'`array`.

La funzione che genera i punti del poligono regolare inscritto in una circonferenza è così definita:

```
def generate_polygon(center, radius, num_sides):
    angles = np.linspace(0, 2 * np.pi, num_sides, endpoint=False)
    x = center[0] + radius * np.cos(angles)
    y = center[1] + radius * np.sin(angles)
    return np.column_stack((x, y))
```

Il calcolo di $l(C_{t,(x,y)})$ è eseguito dalla funzione `perimeter(vertices)`:

```
def perimeter(vertices):
    perim = 0
    for i in range(len(vertices)):
        x1, y1 = vertices[i]
        x2, y2 = vertices[(i + 1) % len(vertices)]
        perim += np.sqrt((x2 - x1)**2 + (y2 - y1)**2)
    return perim
```

Questa funzione esegue semplicemente il calcolo della distanza tra le coppie di punti successivi all'interno di `vertices`, corrisponde a:

$$\sum_{k=0}^n |\vec{c}_k(x, y) - \vec{c}_{k+1}(x, y)|$$

Successivamente si creano tanti poligoni quanti sono i punti sulla griglia:

```
polygons = []
for center in centers:
    polygon = generate_polygon(center, radius, num_sides)
    polygons.append(polygon)
```

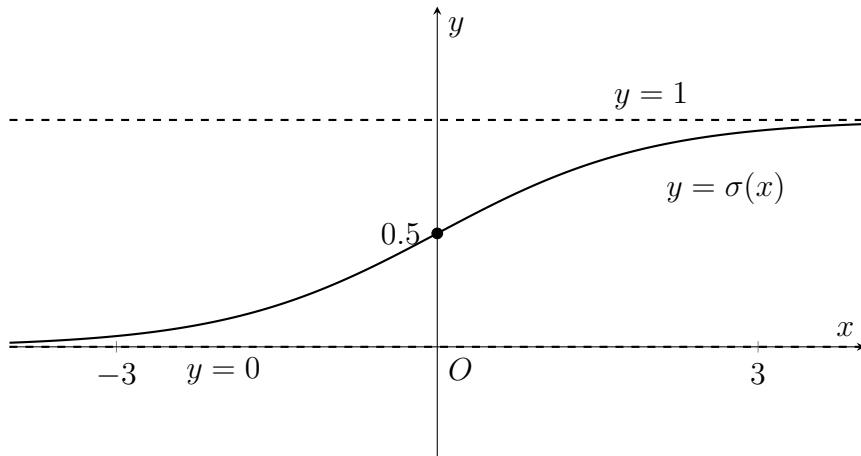
E vengono infine mandati alla GPU per poter eseguire l'integrazione numerica:

```
vertices_device = cuda.to_device(vertices)

integrate[blocks_per_grid, threads_per_block](vertices_device, dt, num_steps)
```

Il metodo `to_device` sposta i dati di tutti i vertici dalla CPU alla GPU, mentre la funzione `integrate(vertices_device, dt, num_steps)` esegue l'integrazione numerica con il metodo di Eulero. Segue l'esempio delle funzioni utilizzate per integrare il modello ridotto di Hodgkin-Huxley, la variabile `x` corrisponde alla differenza di potenziale V , mentre la variabile `y` corrisponde alla frazione di canali ionici di potassio aperti n . La funzione `sigmoid` è definita come segue

$$\text{sigmoid}(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$



```

@cuda.jit
def integrate(vertices, dt, num_steps):
    i = cuda.grid(1)
    x = vertices[i, 0]
    y = vertices[i, 1]

    g_Na = 20.0
    g_K = 10.0
    g_L = 8.0
    E_Na = 60.0
    E_K = -90.0
    E_L = -80.0
    k_m = 15.0
    k_n = 5.07
    I_ext = 0

    if bifurcation_type_id == 0: # Saddle-node
        V_mid_n = -25
        V_mid_m = -20
        tau_n = 0.152

    elif bifurcation_type_id == 1: # SNIC
        V_mid_n = -25
        V_mid_m = -20
        tau_n = 1.0

    elif bifurcation_type_id == 2: # Subcritical Hopf
        V_mid_n = -45
        V_mid_m = -30
        tau_n = 1.0

    elif bifurcation_type_id == 3: # Supercritical Hopf
        V_mid_n = -45
        V_mid_m = -20
        tau_n = 1.0

    else:
        return

    for step in range(num_steps):
        m_inf = sigmoid((x - V_mid_m) / k_m)
        n_inf = sigmoid((x - V_mid_n) / k_n)

        dx = I_ext - g_Na * m_inf * (x - E_Na) - g_K * y * (x - E_K) - g_L * (x -
            ↵ E_L)
        dy = (n_inf - y) / tau_n

        x += dx * dt
        y += dy * dt

    vertices[i, 0] = x
    vertices[i, 1] = y

```

Il sistema di equazioni differenziali che modella il neurone è il seguente

$$\begin{cases} V' = I_{\text{ext}} - g_{\text{Na}} \cdot m_{\infty} \cdot (V - E_{\text{Na}}) - g_{\text{K}} \cdot n \cdot (V - E_{\text{K}}) - g_{\text{L}} \cdot (V - E_{\text{L}}) \\ n' = \frac{n_{\infty} - n}{\tau_n} \end{cases}$$

Con

$$m_{\infty} = \sigma \left(\frac{V - V_{\text{mid},m}}{k_m} \right)$$

$$n_{\infty} = \sigma \left(\frac{V - V_{\text{mid},n}}{k_n} \right)$$

I parametri utilizzati e le equazioni del modello sono tratte da (Kirisanov, A. 2024) e da (Izhikevich, E. M. 2010).

Le costanti g corrispondono alla conduttanza dei canali ionici Na, K e L, rispettivamente del sodio, del potassio e di *leak* (conduttanza di dispersione) e misura la facilità con cui gli ioni attraversano un canale ionico. I valori di E invece sono dei valori critici e stabiliscono per i vari tipi di ioni qual è il valore del potenziale della membrana al quale la forza elettrochimica che spinge lo ione all'interno del neurone è bilanciata da quello che lo spinge verso l'esterno. Se il potenziale è uguale al valore di E la corrente di quel tipo di ione attraverso la membrana si annulla.

Le variabili m_{∞} e n_{∞} corrispondono ai valori di stato stazionario delle variabili di *gating* m e n , ovvero le frazioni di canali ionici aperti per il sodio e per il potassio. $V_{\text{mid},m}$ e $V_{\text{mid},n}$ rappresentano i valori del potenziale della membrana in cui sono aperti metà canali ionici. k_n e k_m modellano invece la velocità con cui questi canali si aprono e si chiudono. Uno dei motivi per cui è possibile ridurre il modello di Hodgkin-Huxley da 4 a 2 variabili di stato deriva dalla quasi istantaneità di apertura e di chiusura dei canali ionici del sodio (m). τ_n è un'altra variabile che influenza la velocità del processo di apertura e di chiusura dei canali ionici del potassio.

`x += dx * dt` e `y += dy * dt` è semplicemente l'applicazione del metodo di Eulero. `bifurcation_type_id` è un valore intero che cambia alcuni parametri in base al tipo di biforcazione che si vuole visualizzare.

Tutti i vertici integrati vengono poi riportati sulla CPU per poter calcolare il rapporto $\frac{c'}{c}$

```
vertices_final = vertices_device.copy_to_host()

final_perimeters = np.array([perimeter(vertices_final[i * num_sides:(i + 1) *
                                                               num_sides]) for i in range(len(polygons))])
final_centers = np.array([centroid(vertices_final[i * num_sides:(i + 1) *
                                                               num_sides]) for i in range(len(polygons))])
```

```
ratios = final_perimeters / initial_perimeters
```

2.1.2 Visualizzazione dello Spazio delle Fasi

Nel file principale usato per calcolare l'operatore di stabilità sono presenti quattro distinti modi di visualizzazione:

1. `graph = 0`: mostra il campo scalare della stabilità come nella figura 1.5.
2. `graph = 1`: mostra un numero variabile di poligoni e la loro trasformazione dopo un determinato t e il valore di $\frac{c'}{c}$ per ognuno di questi. La figura 1.4 ne è un esempio.
3. `graph = 2, phase_flow = False`: mostra il campo vettoriale utilizzando il metodo `quiver` dalla libreria `matplotlib`. Un esempio è la figura 1.3.
4. `graph = 2, phase_flow = True`: mostra le traiettorie di alcuni punti nel piano delle fasi utilizzando il metodo `streamplot` dalla libreria `matplotlib`.

Grafico di $\mathcal{C}(t)$

La funzione principale utilizzata per generare il grafico è:

```
ax.imshow(ratios_grid, extent=[domain_x[0], domain_x[1], domain_y[0],
↪ domain_y[1]], origin='lower', cmap=seismic_fine, vmin=0, vmax=2,
↪ interpolation='catrom')
```

La mappa di colore (`cmap`) utilizzata è `seismic`, per aumentarne la definizione ed ottenere immagini che non abbiano dei salti evidente di colore dovuti alla finitezza del numero dei colori (spesso evidenti se i valori della funzione sono simili su un grande intervallo) è necessario importare la *color map* con una definizione più alta

```
seismic_fine = plt.get_cmap("seismic", 5000)
```

Dove `5000` indica il numero di colori importati, il valore di default è `128`.

I valori visualizzati vengono *clippatti* tra 0 e 2, come specificato dagli argomenti `vmin` e `vmax`. In questo modo la striscia di colore bianco caratteristica della mappa di colore `seismic` si trova esattamente sul valore 1 e coincide con la zona di punti in cui la stabilità passa da essere convergente a divergente



Figura 2.1: Mappa di colore `seismic` (Matplotlib Dev. Team 2024).

```

x_range = domain_x[1] - domain_x[0]
y_range = domain_y[1] - domain_y[0]
ax.set_aspect(x_range / y_range)

```

Dato che `x_range` è diverso da `y_range`, per ottenere un *aspect ratio* del grafico 1:1 è necessario impostarlo attraverso il metodo `set_aspect`.

Grafico dei poligoni

```

for i, polygon in enumerate(polygons):
    x, y = polygon[:, 0], polygon[:, 1]
    ax.plot(x, y, color='b', alpha=0.75)
    center_x, center_y = np.mean(x), np.mean(y)
    ax.plot(center_x, center_y, 'o', markersize=3, color='black')
    ax.text(center_x, center_y + 0.15, f'{ratios[i]:.3f}', color='black',
            ha='center')

for i in range(len(vertices_final) // num_sides):
    polygon_final = vertices_final[i * num_sides:(i + 1) * num_sides]
    x, y = polygon_final[:, 0], polygon_final[:, 1]
    ax.plot(x, y, color='r', alpha=0.75)

```

Sono presenti due cicli `for` che iterano su tutti i poligoni generati. Il primo mostra i poligoni iniziali blu (`color='b'`), il centro di questi poligoni e il valore di $C(t)$ in quel punto approssimato al millesimo (`:.3f`). Il secondo invece mostra in rosso (`color='r'`) i poligoni dopo che i loro punti sono stati integrati nel sistema.

Grafico con `quiver`

```

ax.quiver(initial_centers[:, 0], initial_centers[:, 1], Ux_normalized,
           Uy_normalized, magnitudes, cmap='plasma', angles='xy', scale = 30)

```

`initial_centers` corrisponde all'insieme dei punti dove si vuole calcolare il vettore `Ux_normalized` e `Uy_normalized`, che coincide con la direzione del campo vettoriale in quel punto. `magnitudes` è necessario per la colorazione dei vettori in funzione del loro modulo, mentre `cmap='plasma'` stabilisce la mappa di colore da utilizzare per il grafico. `angles` e `scale` sono due argomenti necessari per assicurarsi che i vettori vengano rappresentati della giusta dimensione e nella giusta direzione.

I vettori risultati `U` e `V` vengono normalizzati nel seguente modo

```

magnitudes = np.sqrt(Ux**2 + Uy**2)
Ux_normalized = Ux / (magnitudes + 1e-8)
Uy_normalized = Uy / (magnitudes + 1e-8)

```

Per fare in modo che il vettore \vec{v} abbia un modulo unitario, si divide il vettore stesso per il proprio modulo:

$$\hat{v} = \frac{\vec{v}}{||\vec{v}||}$$

Per evitare divisioni per 0 viene aggiunto a denominatore un valore molto piccolo: 1×10^{-8} .

Grafico con `streamplot`

```
axs.streamplot(X, Y, U, V, density = [1,1], linewidth = lw, color = lw, cmap =
    plasma_fine, integration_direction = 'both')
```

Questo metodo non visualizza semplicemente il campo vettoriale, ma mostra la traiettoria di alcuni punti sul piano. I punti totali nel piano corrispondono alle variabili `X` e `Y`, mentre il valore delle derivate lungo le due direzioni (`dx*dt` e `dy*dt`) corrisponde alle variabili `U` e `V`.

L'argomento `density` stabilisce il numero di punti nella griglia di cui mostrare le traiettorie, sono presenti due valori perché è possibile scegliere due valori differenti per la densità lungo i due assi.

`linewidth` e `color` sono due argomenti che vengono utilizzati per mostrare l'intensità del campo, che viene calcolato come `lw` (*LineWidth*)

```
magnitude = np.sqrt(U**2 + V**2)
lw = 4 * magnitude/np.max(magnitude + 0.5) + 0.5
```

Per `plasma_fine` vale lo stesso discorso fatto riguardo a `seismic_fine`:

```
plasma_fine = plt.get_cmap('plasma', 5000)
```

2.1.3 Utilizzo di CUDA

Per poter passare alla GPU una funzione, prima di tutto è necessario aggiungere dei decoratori, questi sono `@cuda.jit(device=True)` e `@cuda.jit`. Il primo viene utilizzato per definire la funzione $\sigma(x)$, dove il flag `device=True` assicura che l'operazione venga eseguita sulla GPU, mentre il secondo è utilizzato per definire il sistema dinamico e l'integrazione numerica.

```
@cuda.jit(device=True)
def sigmoid(x):
    return 1 / (1 + math.exp(-x))
```

All'interno del *kernel* principale è presente `i = cuda.grid(1)`. Questo permette di ottenere l'indice univoco dei *thread* all'interno della griglia di *thread* CUDA. Un *thread* è la più piccola unità di esecuzione che una GPU può gestire.

Ogni *thread* esegue le stesse istruzioni, ma con dati diversi, questo è il principio che sta a fondamento del calcolo inter-indipendente. I *thread* sono organizzati in blocchi e poi ulteriormente in griglie. Gli *Streaming Multiprocessors* (SM) gestiscono uno o più blocchi.

```
threads_per_block = 416
blocks_per_grid = (vertices.shape[0] + (threads_per_block-1)) // threads_per_block
```

Si rende dunque necessario stabilire quanti *thread* debba svolgere ogni blocco. Questo è un parametro fondamentale e influenza molto l'efficienza di esecuzione, tanto che il *software* NNC permette di visualizzare la *warp occupancy* al variare di questo valore

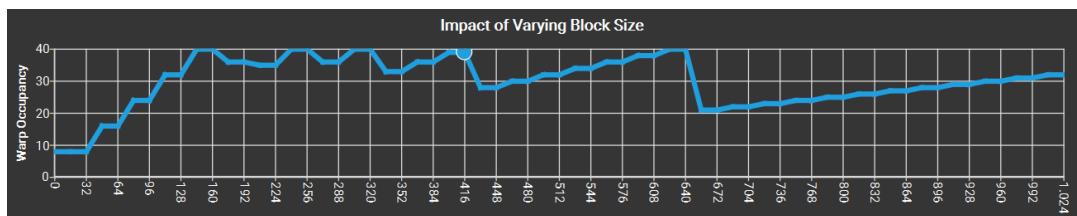


Figura 2.2: Grafico di NVIDIA Nsight Compute che mostra la *Warp Occupancy* al variare del *Block Size*.

La *warp occupancy* è il valore che corrisponde alla percentuale di *warp* (un gruppo di 32 *thread*) utilizzati. Per massimizzare l'efficienza bisognerebbe cerca di portare la *warp occupancy* al valore più alto possibile. Per sfruttare al meglio le risorse è necessario distribuire il carico di lavoro in modo uniforme affinché tutte le componenti lavorino al massimo del potenziale per il maggior tempo possibile.

`blocks_per_grid` infine calcola il numero di blocchi necessari per assicurare che tutti i punti vengano integrati.

```
cuda.synchronize()
vertices_final = vertices_device.copy_to_host()
```

Il primo metodo permette di aspettare che la GPU finisca tutte le operazioni prima di continuare ad eseguire il codice principale sulla CPU. Il secondo invece trasferisce i risultati ottenuti dalla GPU alla CPU.

Una volta integrati i punti nel sistema dinamico è necessario calcolare il perimetro di tutti i poligoni. Dato che il numero di vertici per poligono generalmente è un numero sotto le centinaia, utilizzare la GPU per svolgere questo calcolo potrebbe portare a dei potenziali rallentamenti. La funzione `perimeter(vertices)` infatti si trova all'interno di un *loop* che itera su tutti i punti della griglia. Se `grid_size = 250`, ad esempio, la funzione `perimeter(vertices)` verrà chiamata 62500 volte, il tempo di svolgimento della funzione stessa, che calcola il perimetro di un singolo poligono di meno di un centinaio di lati potrebbe non compensare il tempo impiegato ad inizializzare la funzione. Per questo motivo questa funzione viene svolta direttamente sulla CPU, in questo caso utilizzare il calcolo in parallelo potrebbe non portare a benefici in termini di efficienza.

È altresì vero che si potrebbe integrare il calcolo del perimetro all'interno della funzione `integrate(vertices_device, dt, num_steps)` affinché il loop venga gestito in modo più efficiente, nonostante questo la differenza sarebbe comunque minima, dato che il rapporto tra il numero di operazioni necessarie per integrare i punti è uguale a (1.4) e il numero di operazioni necessarie per calcolare il perimetro è dato da

$$n \cdot p^2$$

il rapporto equivale a

$$n_{dt}$$

Che nel caso in cui $t = 10$ e $dt = 0.0001$, $n_{dt} = 100000$. Questo significa che trovare il perimetro è un'operazione che richiede 100000 volte meno risorse rispetto a integrare tutti i punti.

2.2 Visualizzazioni dei Risultati

Le immagini presenti in questa sezione, a differenza di quelle precedenti, sono relative all'applicazione sviluppata in C++. L'intero codice sorgente dell'applicazione, istruzioni sull'utilizzo e un *installer* per Windows sono disponibili sulla *repository Github* (Riva, D. 2025). Per poter utilizzare il programma è sufficiente disporre di un sistema operativo Windows e di una GPU NVIDIA con *driver* aggiornati. Tutti i requisiti sono comunque specificati sul sito.

L'applicazione permette di visualizzare diversi sistemi dinamici, quelli disponibili sono:

- Pendolo
- Lotka-Volterra (Preda-Predatore)
- Lotka-Volterra (Competizione Intraspecifica)
- Modello di Hodgkin-Huxley ridotto

Per ognuno di questi sistemi è presente una spiegazione generale del sistema e una descrizione di tutte le variabili e dei parametri. È possibile modificare i parametri in tempo reale dall'interfaccia grafica dell'applicazione.

Impostazioni dell'Applicazione

Nel pannello delle impostazioni sulla sinistra è possibile modificare le impostazioni relative all'integrazione dell'operatore di stabilità.



Figura 2.3: Impostazioni utilizzando dell'applicazione `dt`. **Figura 2.4:** Impostazioni utilizzando dell'applicazione `Iterations`.

Dall'applicazione, tramite il pulsante `Usa # iterazioni` è possibile passare da un'integrazione con un passo fissato, `0.1` nell'immagine, ad un'integrazione con un numero fissato di iterazioni, facendo dunque variare il valore di `dt` in base al tempo totale di integrazione. Quest'ultima opzione è particolarmente utile se si sta analizzando una certa regione del modello facendo variare molto il tempo totale di integrazione, infatti, se si tenesse un passo fissato e si

utilizzasse un tempo totale troppo elevato, l'applicazione impiegherebbe molto tempo per elaborare ogni *frame*, rendendo dunque disagevole l'utilizzo dell'applicazione stessa. La seconda opzione permette di ovviare a questo problema imponendo un numero fissato di iterazioni. Ovviamente utilizzare un numero eccessivamente ridotto di iterazioni per valori di T elevati, pur guadagnando in *FPS*, comporta una grande perdita di precisione, che dipende anche dalla complessità del modello scelto.

Il parametro **Block Dim** influisce enormemente sulle prestazioni e sulla risoluzione in output. Un valore pari ad 1 comporta che il campo venga calcolato per ogni pixel sullo schermo, alla massima risoluzione dunque, ma potrebbe risultare eccessivamente lento. Aumentando questo valore l'applicazione risponde molto più velocemente, a discapito però della risoluzione, il campo viene visualizzato infatti più “sgranato”².

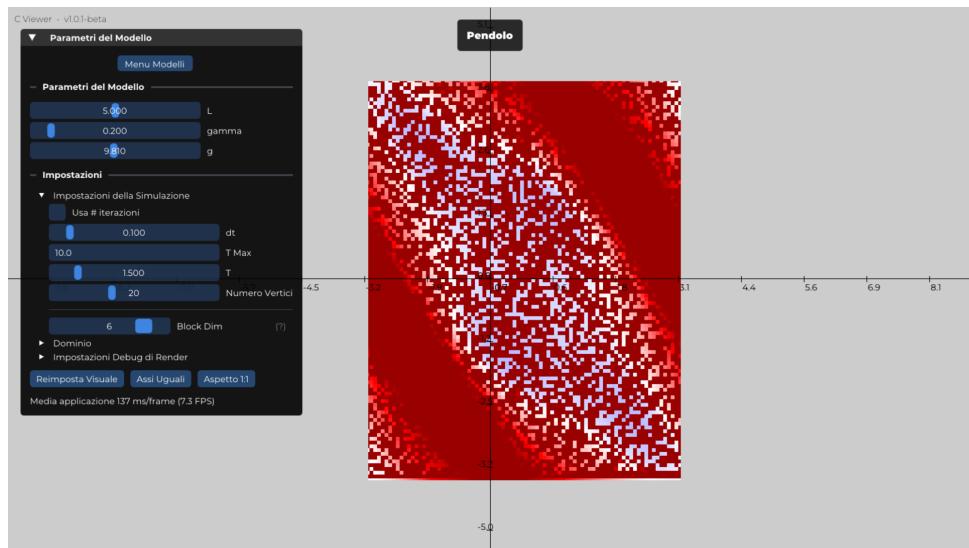


Figura 2.5: Screenshot dell'applicazione *C-Viewer* in beta. In questo caso **Block Dim** è stato impostato su **6**.

Come si vede dalla figura 2.5 l'utilizzo di valori troppo alti di **Block Dim** porta l'applicazione a generare dei campi di stabilità completamente sbagliati e privi di significato. In questo caso i dati passati alla GPU non sono divisi nel modo corretto e probabilmente non tutti i pixel vengono processati correttamente, portando all'emergere di rumore e pixel calcolati erroneamente sull'immagine.

²Questa più che una *feature* è una utile proprietà emergente non pianificata, ancora non mi è ben chiaro il legame reale tra **Block Dim** e la risoluzione dell'output. Con la seconda versione dell'applicazione oltre ad aggiungere altre funzionalità ho intenzione di risolvere questo problema, perché di questo si tratta.

2.2.1 Simulazione della Competizione Interspecifica

Le equazioni di questo modello sono le seguenti

$$\begin{cases} \frac{dN_1}{dt} = r_1 N_1 \left(1 - \frac{N_1 + \alpha_{12} N_2}{K_1} \right) \\ \frac{dN_2}{dt} = r_2 N_2 \left(1 - \frac{N_2 + \alpha_{21} N_1}{K_2} \right) \end{cases}$$

In questo modello N rappresenta la popolazione totale ad un certo tempo t , r corrisponde al tasso di crescita, K è detta capacità portante, ovvero la popolazione in cui il tasso di mortalità bilancia quello di natalità, e α è l'effetto competitivo di una specie sull'altra.

Nella prima metà del ventesimo secolo Georgy Frantsevich Gause, biologo ed evoluzionista Russo, fece degli esperimenti sull'evoluzione della popolazione di due specie di protozoi. Notò che, se lasciate crescere in ambienti separati, le due colture seguivano una crescita di tipo logistica, ma, se fatte sviluppare nello stesso ambiente, allora una delle due specie prevaleva, causando dopo alcuni giorni l'estinzione dell'altra.

Nel primo caso la popolazione di protozoi è limitata solamente da un fattore, ovvero la disponibilità di cibo.

Nel secondo invece le dinamiche sono molto più complesse: l'introduzione di una specie esterna non è solamente un secondo fattore limitante, in quanto ogni interazione rilevante che la prima specie ha con la seconda si riflette direttamente o indirettamente sulla popolazione di entrambe. Questo porta all'emergere di diverse dinamiche qualitativamente differenti.

In particolare si possono presentare quattro scenari (che nell'applicazione *C-Viewer* sono selezionabili tramite il menù a tendina **Presets**):

- **Esclusione Competitiva (1):** la prima specie porta la seconda ad estinguersi. Si ottiene per $K_1 > K_2 \alpha_{21} \wedge K_1 \alpha_{12} > K_2$
- **Esclusione Competitiva (2):** la seconda specie porta la prima ad estinguersi. Si ottiene per $K_1 < K_2 \alpha_{21} \wedge K_1 \alpha_{12} < K_2$
- **Coesistenza:** il fattore di limitazione intraspecifico prevale su quello interspecifico, questo porta alla coesistenza di entrambe le specie. Si ottiene per $K_1 > K_2 \alpha_{21} \wedge K_1 \alpha_{12} < K_2$
- **Esclusione Competitiva (instabile):** il fattore di limitazione interspecifico prevale su quello interspecifico, questo porta all'emergere di un punto di equilibrio instabile. Si ottiene per $K_1 < K_2 \alpha_{21} \wedge K_1 \alpha_{12} > K_2$

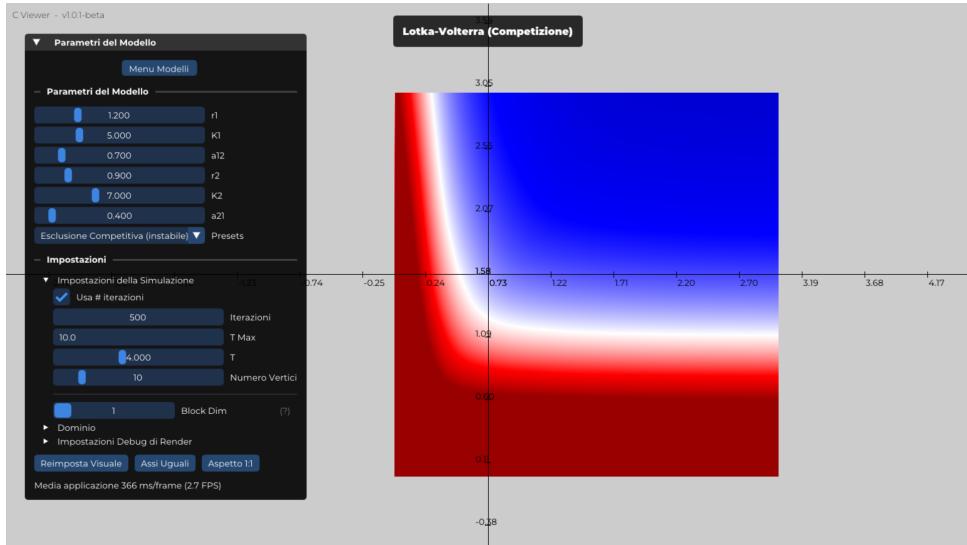


Figura 2.6: Screenshot dell'applicazione *C-Viewer*. Viene mostrato il campo per $t=4$. I parametri sono impostati tramite il menu **Presets** in modo tale da garantire la coesistenza delle due specie.

Nel caso della coesistenza i parametri devono soddisfare le seguenti disequazioni

$$K_1 > K_2 \alpha_{21} \wedge K_1 \alpha_{12} < K_2$$

I valori utilizzati nella figura sono:

$$K_1 = 5$$

$$\alpha_{12} = 0.7$$

$$K_2 = 7$$

$$\alpha_{21} = 0.4$$

Sostituendo si verifica che i parametri verificano le condizioni imposte per la coesistenza:

$$5 > 7 \cdot 0.4 \wedge 5 \cdot 0.7 < 7$$

$$5 > 2.8 \wedge 3.5 < 7$$

Le due isocline che rappresentano i punti dove la popolazione di una specie rimane costante sono delle rette. Per trovare le loro equazioni si pongono le derivate uguali a 0:

$$\begin{cases} 0 = r_1 N_1 \left(1 - \frac{N_1 + \alpha_{12} N_2}{K_1} \right) \\ 0 = r_2 N_2 \left(1 - \frac{N_2 + \alpha_{21} N_1}{K_2} \right) \end{cases}$$

Da cui si ricava:

$$\begin{cases} N_1 = K_1 - \alpha_{12} N_2 \\ N_2 = K_2 - \alpha_{21} N_1 \end{cases}$$

Che si può riarrangiare nella forma:

$$\begin{cases} N_2 = -\frac{N_1}{\alpha_{12}} + \frac{K_1}{\alpha_{12}} \\ N_2 = -\alpha_{21}N_1 + K_2 \end{cases}$$

Dall'intersezione delle rette si ottiene che il punto di equilibrio ha coordinate $\left(\frac{K_1 - \alpha_{12}K_2}{1 - \alpha_{12}\alpha_{21}}, \frac{K_2 + \alpha_{21}K_1}{1 - \alpha_{12}\alpha_{21}}\right)$, che nel caso di prima corrisponde al punto di coordinate $\left(\frac{5}{36}, \frac{125}{18}\right)$. Sostituendo il punto nelle equazioni del sistema per verificare si ottiene effettivamente che:

$$\begin{cases} \frac{dN_1}{dt} = 1.1 \cdot \frac{5}{36} \left(1 - \frac{\frac{5}{36} - 0.7 \cdot \frac{125}{18}}{5}\right) \\ \frac{dN_2}{dt} = 0.8 \cdot \frac{125}{18} \left(1 - \frac{\frac{125}{18} - 0.4 \cdot \frac{5}{36}}{7}\right) \end{cases} \quad \begin{cases} \frac{dN_1}{dt} = 0 \\ \frac{dN_2}{dt} = 0 \end{cases}$$

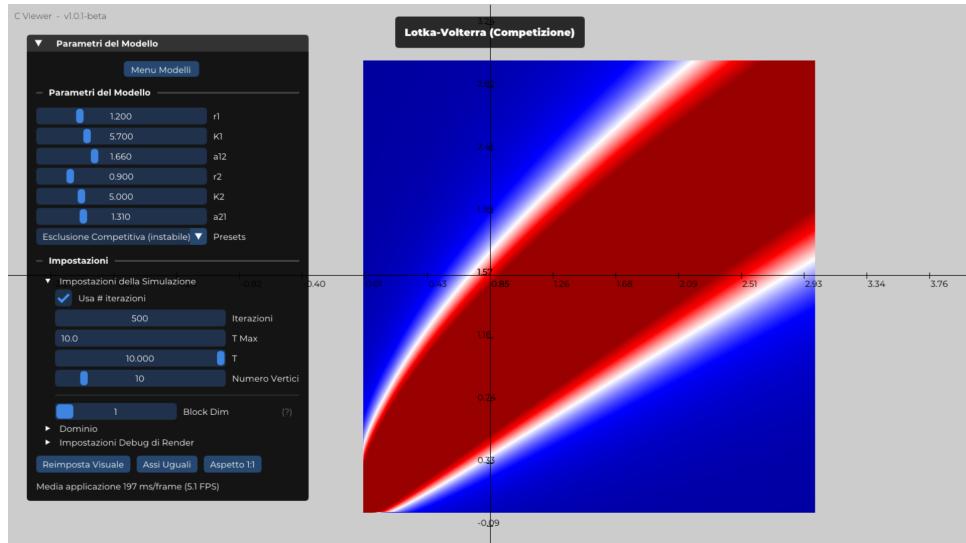


Figura 2.7: In questo screenshot invece i parametri del modello sono configurati in modo tale da garantire un'esclusione competitiva di tipo instabile.

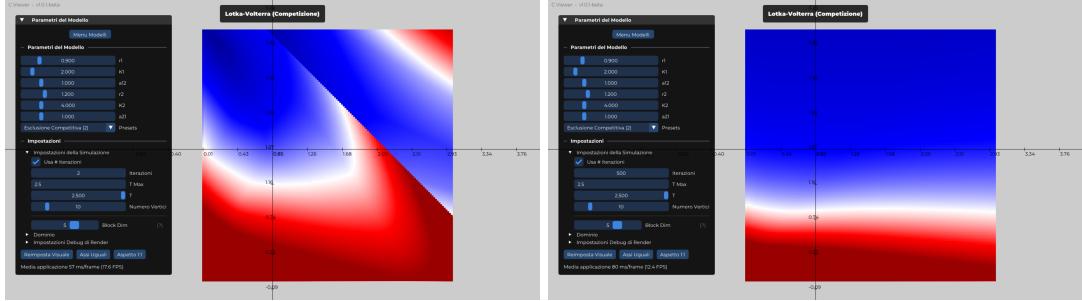


Figura 2.8: In questo esempio il numero di iterazioni è stato ridotto in modo significativo zato con un numero di iterazioni adeguato, si per mostrare quanto il risultato sia incoerente tratta del caso di esclusione competitiva (2), con il risultato corretto mostrato nella figura dove a prevalere è la seconda specie.

2.9.

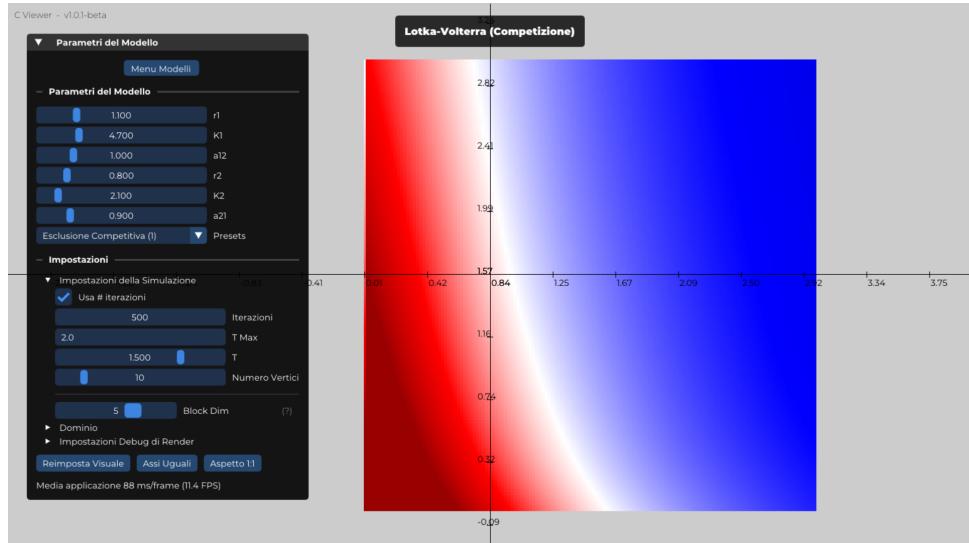


Figura 2.10: Campo della stabilità nel caso di esclusione competitiva (1), qui è la prima specie a prevalere.

Nelle figure 2.9 e 2.10 la situazione è piuttosto evidente: nel primo caso si nota immediatamente una forte instabilità quando la popolazione N_2 è bassa. Poiché il modello rappresenta uno scenario di esclusione competitiva (dove a prevalere è proprio N_2), all'aumentare del tempo di evoluzione la prima specie andrà ad estinguersi ($N_1 \rightarrow 0$), mentre la seconda specie raggiungerà il relativo valore di capacità portante ($N_2 \rightarrow K_2$), nel caso specifico della figura 2.9 è infatti presente un punto di equilibrio stabile in $(0, 4)$.

La porzione più instabile è quella inferiore perché in quella regione la prima specie è limitata per $N_1 < K_1$ (che in questo caso particolare è 2) dal fattore interspecifico, ovvero da $\alpha_{12}N_2$. In un primo momento è poco rilevante poiché la popolazione della seconda specie è molto ridotta e permette alla prima di

crescere, il fattore di limitazione interspecifico è inibito dal numero ridotto della popolazione N_2 . Questo porta la regione ad essere instabile perché ci si sta allontanando dal punto di equilibrio, le traiettorie dei punti tendono ad aprirsi.

Per quanto riguarda $N_1 > K_1$, considerando valori piccoli per N_2 , il fattore limitante è quello intraspecifico, essenzialmente ciò che porta a diminuire la popolazione di entrambe le specie è la mancanza di risorse in rapporto al numero della popolazione delle specie. In questo caso N_1 diminuisce e tende a K_1 . L'instabilità emerge per via dell'andamento della seconda specie. In sintesi, a causa del ridotto numero della popolazione, anche la derivata è altrettanto ridotta e per un certo tempo conserva le alterazioni introdotte da una perturbazione. Successivamente il tasso di variazione di N_2 diventa positivo perché il fattore limitante non è più quello intraspecifico, ma quello interspecifico. Nel caso dell'esclusione competitiva la seconda specie prevale sulla prima e quindi la derivata è positiva. Il fatto che presenti il termine N_2 come fattore moltiplicativo nel calcolo della derivata porta ad un andamento esponenziale e anche una piccola differenza in N_2 viene amplificata, per questo motivo la fascia inferiore è instabile.

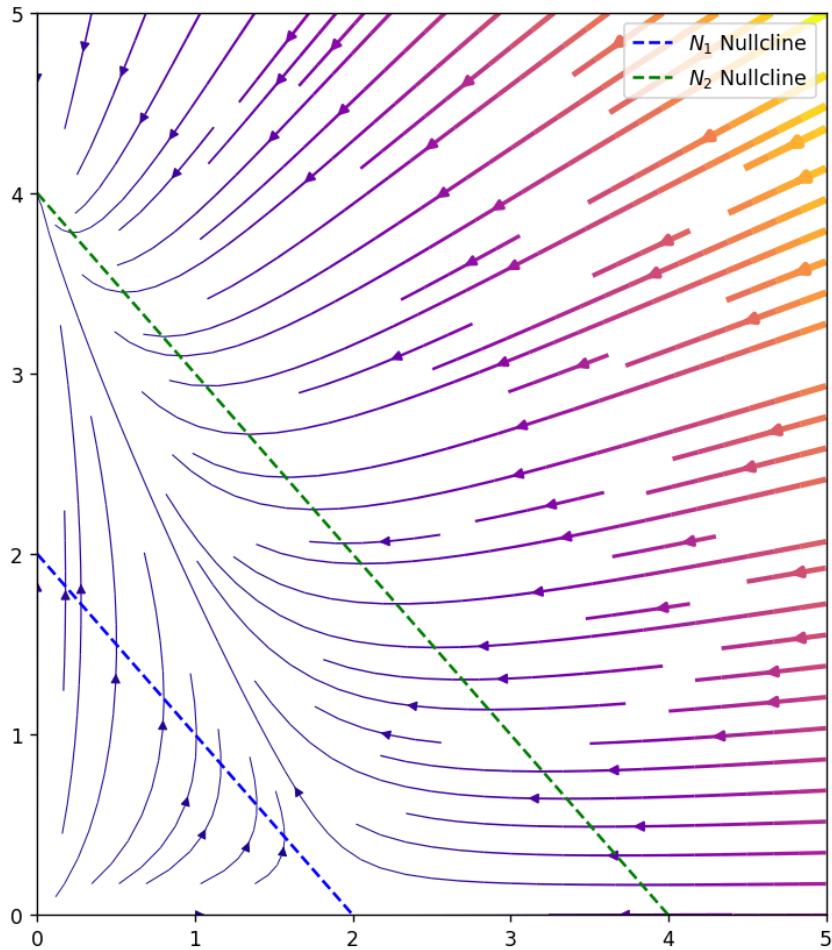


Figura 2.11: Spazio delle fasi del modello relativo a 2.9. I due punti di equilibrio sono $(0, 4)$ e $(2, 0)$, il primo è stabile, il secondo no.

La *nullcline* relativa a N_1 mostra i punti in cui la relativa derivata si annulla, tutti i punti che stanno al di sotto della retta blu hanno la caratteristica di crescere in N_1 , quelli che stanno al di sopra della retta invece hanno una derivata negativa e quindi la popolazione di N_1 decresce. Un discorso analogo si può fare per la *nullcline* relativa a N_2 , combinando le due regioni separate dalle *nullclines* si può riassumere che

- In $(0, 0) - (0, 2) - (2, 0)$ N_1 e N_2 crescono.
- In $(0, 2) - (2, 0) - (4, 0) - (0, 4)$ N_1 decresce e N_2 cresce.
- Nella regione restante entrambe le specie decrescono.

2.2.2 Simulazione del Modello di Hodgkin-Huxley

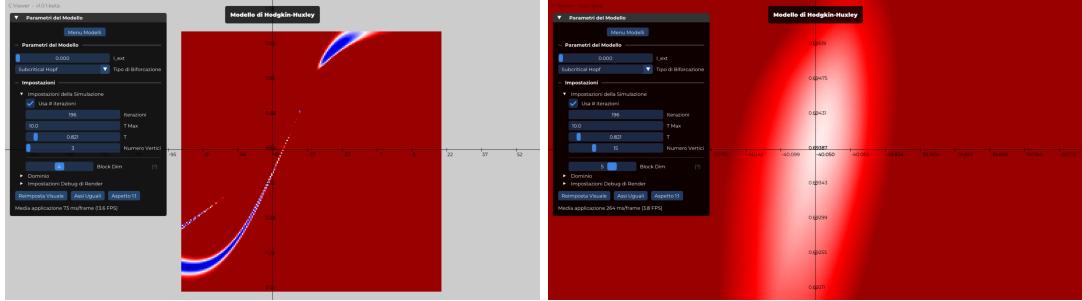


Figura 2.12: Campo della stabilità calcolato per $t = 0.821$, la biforcazione è del tipo Hopf la regione di $(-40, 0.7)$.
sottocritica, non è applicata alcuna corrente esterna.

Come mostrato nelle due figure sopra è possibile eseguire un ingrandimento in qualsiasi regione, arrivando fino ad un fattore massimo pari a 10000. Oltre una certa soglia non avrebbe senso ingrandire poiché gli errori di approssimazione numerici non sarebbero più trascurabili e restituirebbero un campo scalare chiaramente sbagliato. L'applicazione utilizza il tipo di dato `float` a virgola mobile a precisione singola (*single-precision floating-point number*). In generale questo tipo di dato offre una precisione di 7 cifre decimali, per questo motivo ingrandire troppo porterebbe ad errori non trascurabili.

Utilizzando il tipo di dato `double`, come suggerisce il nome, è possibile ottenere una precisione doppia rispetto a `float`, raggiungendo una precisione di 15 decimali. Questi artefatti così facendo vengono decisamente limitati, anche se sono comunque visibili lavorando con numeri eccessivamente grandi o eccessivamente piccoli. Utilizzando questi valori l'applicazione però diventa molto lenta e in generale i benefici introdotti dall'utilizzo del `double` non compensano per gli effetti negativi che questo ha sulle prestazioni. In tutti i sistemi implementati in *C-Viewer*, se si mantiene l'opzione `Utilizza Dominio`, non è possibile notare alcun artefatto di questo tipo. Questa opzione permette di ignorare tutti i valori che sono al di fuori del dominio di integrazione, che è comunque modificabile tramite gli appositi *slider* nella sezione `Dominio`.



Figura 2.14: Impostazioni del dominio. Queste sono le impostazioni di *default* per il modello del pendolo.

Nella figura 2.14 il dominio è $[-\pi, \pi]$ per l’asse delle x , ovvero per θ , che rappresenta l’angolo iniziale del pendolo, e $[-4, 4]$ per l’asse delle y , ovvero per ω , che rappresentava la velocità iniziale del pendolo.

Attivando l’opzione **Utilizza Dominio**, i punti che si trovano all’esterno non subiscono alcuna iterazione e il *kernel* assegna a questi pixel il valore di **-1.0f** terminando la funzione immediatamente per migliorare le prestazioni ed evitare calcoli inutili. Ovviamente questo controllo è eseguito prima dell’integrazione, si perderebbero altrimenti tutti i benefici di questa *feature*. Sarebbe come applicare una maschera a posteriori, andrebbe solamente ad aumentare l’impegno computazionale invece che diminuirlo.

In certi casi, come nel Modello di Hodgkin-Huxley ad esempio, il dominio è utile per visualizzare solamente la regione dello spazio con un reale significato fisico. Non ha senso infatti parlare di valori inferiori a 0 o superiori ad 1 sull’asse verticale nel modello di Hodgkin-Huxley, su quest’asse infatti è rappresentata la percentuale di canali potassio aperti, che in nessun caso potrà mai essere negativa o superiore ad 1. Quindi utilizzare questa funzione permette non solo di aumentare le prestazioni dell’applicazione, ma anche di escludere gli stati del sistema che non hanno un’interpretazione con la realtà.

Per l’asse orizzontale invece non vale la stessa cosa, generalmente i valori utilizzati sono compresi tra -90 e 20 Volt, ma in casi particolari questi limiti potrebbero essere troppo ristretti per descrivere in modo completo la dinamica del modello.

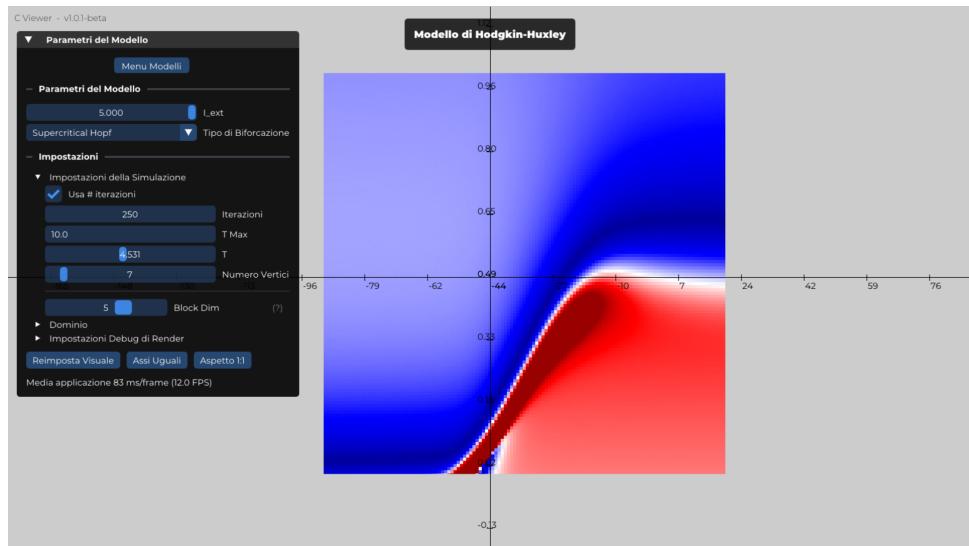


Figura 2.15: Campo della stabilità per $t=10.0$ e $I_{ext} = 5.0$.

L’aggiunta del modello di Hodgkin-Huxley nella libreria dell’applicazione ha reso necessario l’implementazione di tre funzionalità che permettono di gestire al meglio la visualizzazione del dominio. Questa necessità deriva dal fatto che il rapporto tra le dimensioni del dominio sui due assi sul quale viene calcolato il campo della stabilità è elevato. Sull’asse verticale infatti i valori sono compresi in $[0, 1]$ mentre sull’asse orizzontale i valori sono generalmente compresi tra $[-90, 20]$. Utilizzare un valore fissato come unità di misura per entrambi gli assi contemporaneamente è impensabile, per questo motivo nel pannello delle impostazioni, in basso, è presente il pulsante **Aspetto 1:1** che non fa altro che applicare una dilatazione agli assi per fare in modo che l’intera regione venga visualizzata a schermo come un quadrato.

Per applicare l’operazione inversa, ovvero per ripristinare una scala uguale per entrambi gli assi è presente il pulsante **Assi Uguali**, che imposta entrambe le variabili che gestiscono la dilatazione degli assi a **1**.

Un problema di questo modello deriva dalla sua complessità: il numero di operazioni per ogni iterazione sono molti e questo porta ad un rallentamento generale rispetto a modelli più semplici, come quello del pendolo. Questo è un problema perché utilizzando lo zoom la regione di integrazione coprirà necessariamente una porzione maggiore dello schermo, portando a dei rallentamenti ancora più significativi. La soluzione a questo problema³, è utilizzare una maschera, circolare o rettangolare fissata sullo schermo, che permette di saltare l’integrazione di ogni pixel al di fuori di questa regione. Questa operazione di *masking* sarebbe equivalente a quella di usare un dispositivo con una risoluzione dello schermo inferiore in quanto il numero di pixel da processare è ridotto di una certa costante, che dipende dalla grandezza di questa maschera. La conse-

³Issue #3 su (Riva, D. 2025).

guenza sarebbe comunque una riduzione del numero di iterazioni da eseguire, con un netto miglioramento nelle prestazioni.

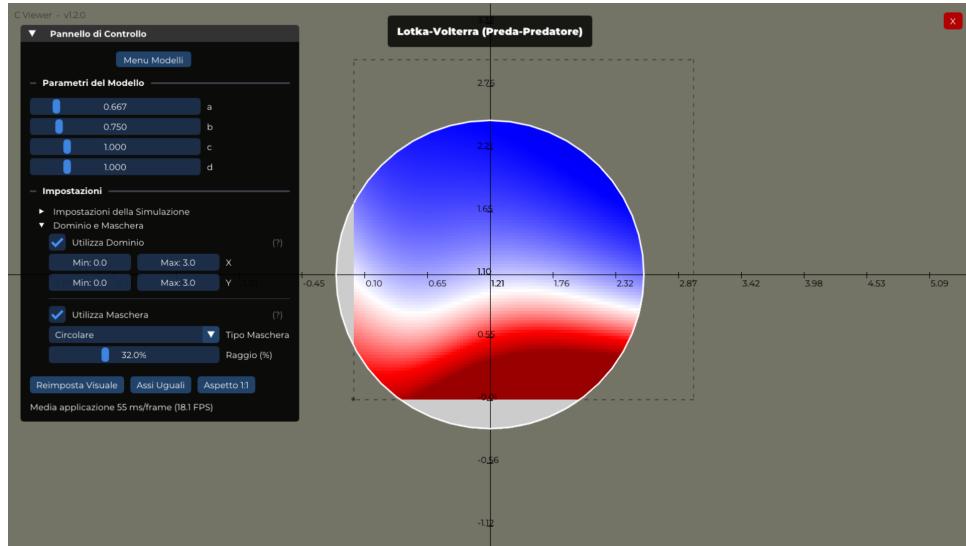


Figura 2.16: In questo screenshot è stata attivata la modalità **Utilizza Maschera**, è stata impostata la maschera di tipo circolare con un raggio pari al 32% della metà della larghezza dello schermo.

Si noti la linea tratteggiata, questa permette di tenere traccia della posizione della regione delimitata dal dominio (se attivato) anche se al di fuori della maschera.

Per poter utilizzare il tipo di maschera rettangolare si possono utilizzare due modalità differenti: la prima (di default) permette di interagire normalmente con il campo scalare, mentre la seconda è necessaria per modificare la posizione e la dimensione della maschera rettangolare. Premendo il tasto ‘M’ è possibile cambiare modalità. Ogni volta che viene premuto questo tasto compare un *pop-up* che descrive la modalità in cui ci si trova, una breve descrizione è presente anche nel pannello delle impostazioni, nella sezione relativa alla maschera.

Una guida all'utilizzo e una spiegazione dettagliata di ogni *feature* dell'applicazione è contenuta nella *repository* (Riva, D. 2025).

2.3 La Funzione $\mathcal{C}_{\vec{p}}(t)$

In questa sezione viene analizzata la funzione $\mathcal{C}_{\vec{p}}(t)$. Questa funzione è definita come segue:

$$\mathcal{C}_{\vec{p}}(t) : I \rightarrow \mathbb{R}^+$$

Con $I \subseteq \mathbb{R}_0^+$, in particolare $I = [0, T_{max}]$, dove $T_{max} \in \mathbb{R}^+$ è un parametro che determina il tempo massimo sul quale viene calcolato il valore della funzione. La funzione è la seguente:

$$C_{x,y}(t) = \{\Phi(t, \vec{c}_{k,x,y}) \mid k \in [0, n], \text{ con } n \in \mathbb{N}\}$$

$$l(C_{x,y}(t)) = c'(t)$$

$$\mathcal{C}_{\vec{p}}(t) = \frac{c'(t)}{c}$$

Essenzialmente l'unica differenza con $C_t(x, y)$ è che x e y sono dei parametri fissati, non variano, mentre la variabile della funzione è t .

Il grafico di questa funzione permette di intuire alcune caratteristiche di un certo stato, in particolare permette di capire se lo stato è caratterizzato da un andamento periodico, esponenziale, asintotico oppure una combinazione di questi per modelli più complessi.

Vale la seguente proprietà:

$$\mathcal{C}_{\vec{p}}(0) = 1$$

Infatti si ricava banalmente da:

$$\mathcal{C}_{\vec{p}}(0) = \frac{c'(0)}{c} = \frac{\phi}{\phi} = 1$$

La seconda uguaglianza è garantita per la proprietà (1.2). Il grafico della funzione quindi parte sempre dal punto $(0, 1)$, indipendentemente dal modello e dai parametri scelti.

Il fatto che la funzione $\mathcal{C}_{\vec{p}}(t)$ per un certo stato \vec{p} sia crescente in un intervallo $[a, b]$ significa che questo stato diventa via via più instabile, più sensibile a delle perturbazioni, questo potrebbe essere indice del fatto che \vec{p} si sta allontanando da un punto di equilibrio stabile.

Se invece $\mathcal{C}_{\vec{p}}(t)$ è decrescente in questo intervallo allora lo stato \vec{p} sta diventando sempre più stabile e potrebbe essere indice di un avvicinamento ad un punto di equilibrio.

2.3.1 Pendolo semplice

Nel caso del sistema dinamico che descrive un pendolo semplice con attrito ci si aspetta che la stabilità per $t \rightarrow \infty$ tenda a 0, questo perché la componente $-\gamma\omega$ che modella l'attrito in funzione della velocità del pendolo porta all'emergere di un punto di equilibrio stabile in $(0, 0)$ in cui ogni stato del sistema è destinato a convergere. Questo punto di equilibrio è presente anche se $\gamma = 0$, solamente che i punti $\vec{p} \neq (0, 0)$ non convergono in questo stato poiché le traiettorie sono chiuse.

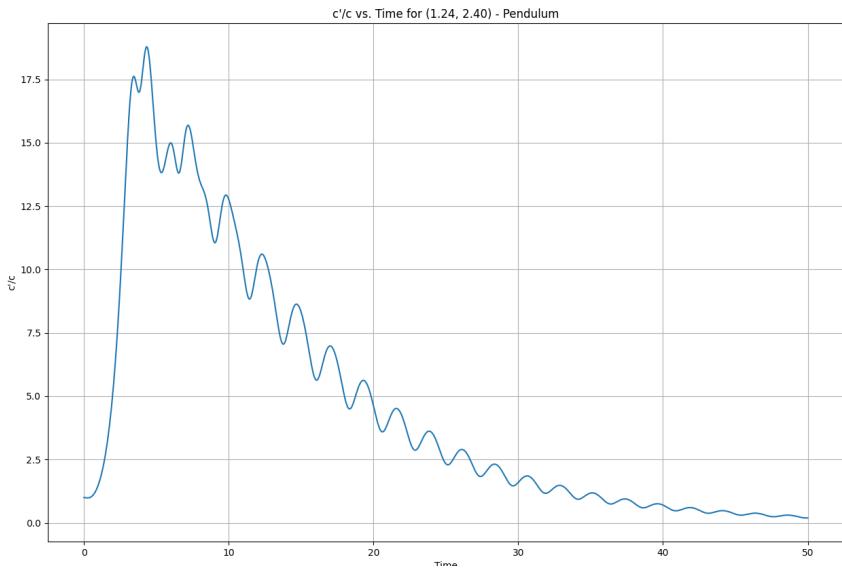


Figura 2.17: Funzione $\mathcal{C}_{\vec{p}}(t)$ per il sistema dinamico del pendolo.

In questo caso i valori iniziali scelti $(1.24, 2.4)$ portano il pendolo vicino al punto di equilibrio instabile, per questo motivo si registra un picco nella stabilità, dal grafico si evince che il sistema è particolarmente instabile nell'intervallo tra $[3, 7]$, una piccola perturbazione nelle vicinanze dell'equilibrio instabile infatti potrebbe il pendolo ad evolvere in due modi completamente differenti.

La funzione $\mathcal{C}_{\vec{p}}(t)$ in effetti esprime quantitativamente l'importanza che una perturbazione in una certa configurazione \vec{p} ha sul modello in un intervallo temporale. Valori elevati potrebbero essere indice del fatto⁴ che in seguito ad una perturbazione iniziale il sistema assuma configurazioni molto distanti per un istante t , anche se poi, come già detto, per $t \rightarrow \infty$ il sistema converge nel punto di equilibrio stabile.

⁴Il fatto che la funzione assuma valori elevati è una condizione necessaria ma non sufficiente per la divergenza delle traiettorie in seguito ad una perturbazione.

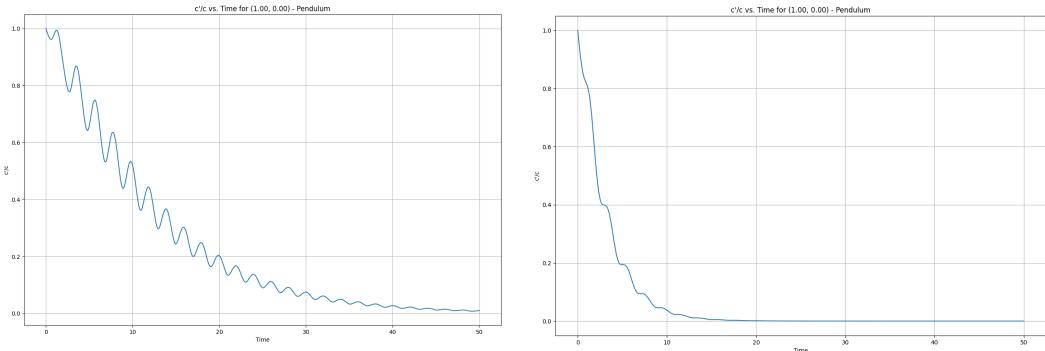


Figura 2.18: Grafico della funzione $\mathcal{C}_p(t)$ con $\gamma = 0.2$. **Figura 2.19:** Grafico della funzione $\mathcal{C}_p(t)$ con $\gamma = 0.7$.

Dal confronto dei due grafici sopra è evidente come il parametro γ , che modella il coefficiente di attrito viscoso, influisca sulla velocità con la quale il sistema raggiunge il punto di equilibrio stabile. Nelle due immagini sopra γ è stato l'unico parametro ad essere modificato.

Per t sufficientemente elevati la stabilità di ogni punto tende a 0. Effettivamente, però esistono delle regioni del modello che portano all'equilibrio instabile, quello in cui lo stato è $(\pm\pi, 0)$ nelle vicinanze delle quali la stabilità è elevata

$$\begin{cases} \dot{\theta} = 0 \\ \dot{\omega} = -\frac{g}{L} \sin(\pm\pi) - \gamma \cdot 0 \end{cases} \quad \begin{cases} \dot{\theta} = 0 \\ \dot{\omega} = 0 \end{cases}$$

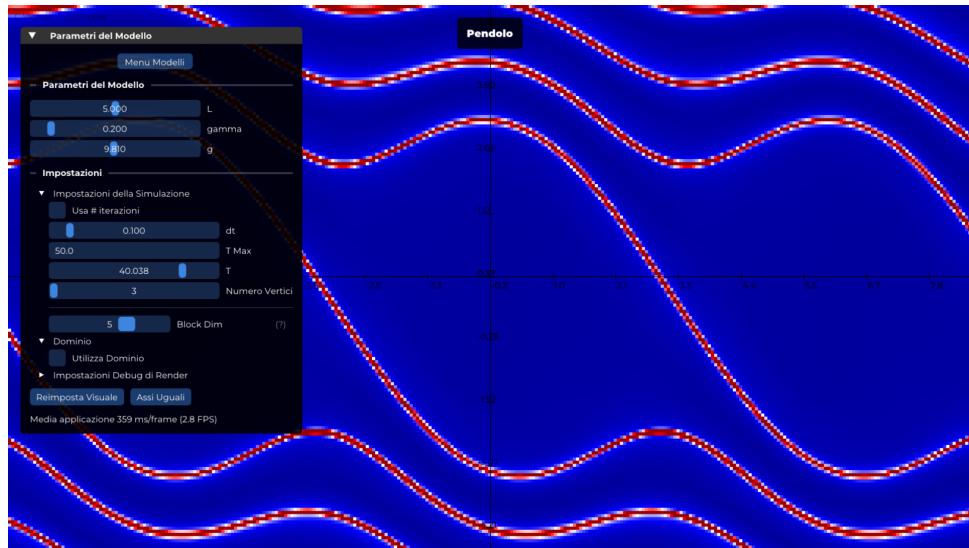


Figura 2.20: Campo della stabilità per $t=50.0$.

Le regioni di instabilità sono evidenti, passano necessariamente per i punti di ordinata 0 e di ascissa $\pi + 2k\pi, k \in \mathbb{Z}$, in quanto tutti punti di equilibrio

instabile. Per $t \rightarrow \infty$ queste regioni vanno a degenerare in una classe di curve φ_k . Tutti i punti $P(x, y) \in \varphi_k$ hanno la proprietà di convergere in un punto di equilibrio instabile. Per ogni angolo θ è quindi possibile trovare almeno un valore di $\dot{\theta}$ che porta il pendolo nel punto di equilibrio instabile $\theta \rightarrow \pi$ per $t \rightarrow +\infty$. In simboli

$$\forall \theta \in \mathbb{R}, \exists \omega \in \mathbb{R} \mid \lim_{t \rightarrow +\infty} \Phi[t, (\theta, \omega)] = (\pi, 0) \quad (2.1)$$

Sia la curva φ il grafico della funzione $\varphi(\theta)$. Questa funzione è suriettiva, ma non iniettiva, non si può escludere che per valori diversi di θ i valori di ω che soddisfano (2.1) siano necessariamente identici.

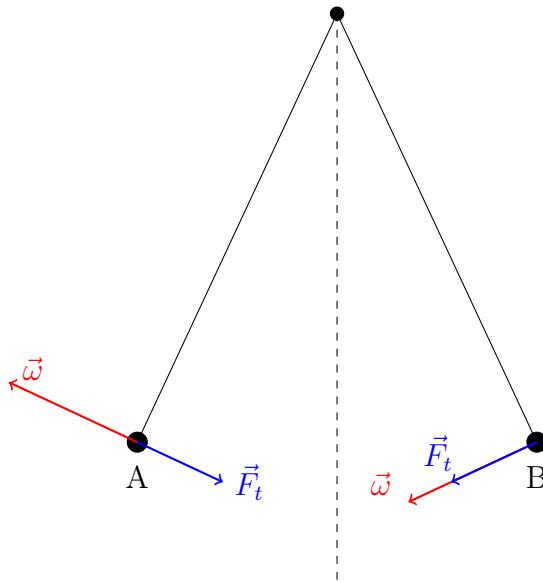


Figura 2.21: Schema di due stati del pendolo semplice.

Nello schema sono rappresentati due stati del modello del pendolo semplice: lo stato A e lo stato B . Si suppone per ipotesi che entrambi gli stati convergano nel punto di equilibrio instabile in $(\pi, 0)$. Con questo esempio si vuole semplicemente palesare in modo del tutto intuitivo e senza alcun tipo di formalità matematica il motivo per cui possono esistere più valori di θ per un unico valore di ω .

Entrambi gli stati A e B hanno, per ipotesi, lo stesso valore di ω , ma differiscono per l'angolo θ . Il concetto sostanziale è che la maggiore distanza che B deve percorrere è compensata dalla forza peso tangente al moto del pendolo, che ha stessa direzione e verso di $\vec{\omega}$. La velocità ω di B infatti aumenterà fino a quando non supererà il punto di equilibrio stabile posto in basso al centro. Quando lo stato B raggiungerà il valore di θ dello stato A allora ω_B sarà aumentato e successivamente diminuito, fino a tornare al valore iniziale. In simboli

$$\Phi[t, (\theta_A, \omega)] = \Phi[t + \Delta t, (\theta_B, \omega)] = (\pi, 0)$$

Si noti che Δt è il tempo che lo stato B impiega per raggiungere lo stato A^5 .

Se il sistema è conservativo però, dimostrare (2.1) potrebbe essere più semplice. Se il sistema del pendolo è conservativo infatti le equazioni del modello sono:

$$\begin{cases} \frac{d\theta}{dt} = \omega \\ \frac{d\omega}{dt} = -\frac{g}{L} \sin \theta \end{cases} \quad (2.2)$$

Inoltre, se due punti appartengono alla stessa traiettoria, l'energia totale dei due stati è la stessa. L'energia totale in questo caso si può calcolare come:

$$E = K + U$$

Con:

$$\begin{cases} K = \frac{1}{2}mv^2 \\ U = mgh \end{cases} \rightarrow \begin{cases} K = \frac{1}{2}m(L\dot{\theta})^2 \\ U = mgL(1 - \cos \theta) \end{cases} \rightarrow E = \frac{1}{2}mL^2\dot{\theta}^2 + mgL(1 - \cos \theta)$$

Poiché conservativo questa quantità rimane costante nel tempo, in particolare il valore dell'energia totale in $(\pi, 0)$ è uguale a:

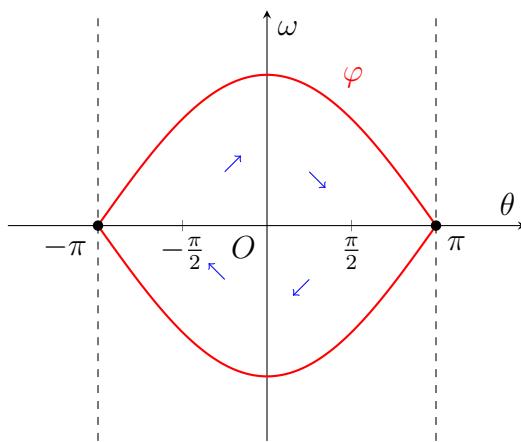
$$E_{\pi,0} = 2mgL$$

Le curva φ in questo caso ha equazione:

$$\begin{aligned} 2mgL &= \frac{1}{2}mL^2\dot{\theta}^2 + mgL(1 - \cos \theta) \\ \frac{1}{2}L\dot{\theta}^2 - g(1 + \cos \theta) &= 0 \end{aligned}$$

Esplicitando per il valore di $\dot{\theta} = \gamma$, la funzione $\varphi(\theta)$ è quindi:

$$\varphi(\theta) = \pm \sqrt{\frac{2g}{L}(\cos \theta + 1)}$$



⁵Sono consapevole del fatto che tutte queste cose andrebbero dimostrate formalmente, in effetti la maggior parte delle osservazioni in questa relazione sono solo delle deduzioni personali che mi sembrano sufficientemente plausibili per essere prese per vere a priori.

Per ogni valore di θ in $(-\pi, \pi)$ esistono due valori di γ che possono portare al punto di equilibrio instabile o in senso orario o nel senso antiorario. In particolare:

$$\lim_{t \rightarrow +\infty} \phi[t, (\theta, \omega)] = \begin{cases} (\pi, 0), & \text{se } \omega > 0 \\ (-\pi, 0), & \text{se } \omega < 0 \end{cases}$$

2.3.2 Lotka-Volterra

Il grafico generato da $\mathcal{C}_{\vec{p}}(t)$ per il modello preda-predatore di Lotka-Volterra è un caso un po' particolare. Il sistema dinamico non è conservativo nel senso classico del termine, infatti se lo fosse il suo rotore dovrebbe essere identicamente nullo:

$$\mathbf{F} \text{ conservativo} \Rightarrow \nabla \times \mathbf{F} = 0$$

$$\mathbf{F} = \begin{cases} \dot{x} = (A - By)x \\ \dot{y} = (Cx - D)y \end{cases}$$

Poiché \mathbf{F} è bidimensionale il calcolo del rotore si riduce a

$$\begin{aligned} |\nabla \times \mathbf{F}| &= \frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y} \\ &= \frac{\partial}{\partial x}(Cx - D)y - \frac{\partial}{\partial y}(A - By)x \\ &= Cy - (-Bx) \\ &= Cy + Bx \neq 0 \end{aligned}$$

Il sistema dunque non è conservativo, ma si dimostra che esiste una quantità descritta da una funzione di energia, l'hamiltoniana, che rimane costante lungo le traiettorie. Questo implica che il sistema non è dissipativo e che quindi le traiettorie sono chiuse⁶. I campi conservativi hanno il difetto di essere particolarmente sensibili a delle perturbazioni, in particolare gli errori di approssimazione numerici possono diventare un problema non trascurabile se si usano metodi di integrazioni con errori globali di ordine relativamente bassi. Il problema è aggravato ulteriormente dall'utilizzo di passi temporali non sufficientemente piccoli.

Questa premessa è necessaria per evidenziare un aspetto che appare contraddittorio osservando il grafico per il modello di Lotka-Volterra.

⁶Non ho voluto approfondire questo punto perché non ho le conoscenze sufficienti per poterne trattare in modo consapevole.

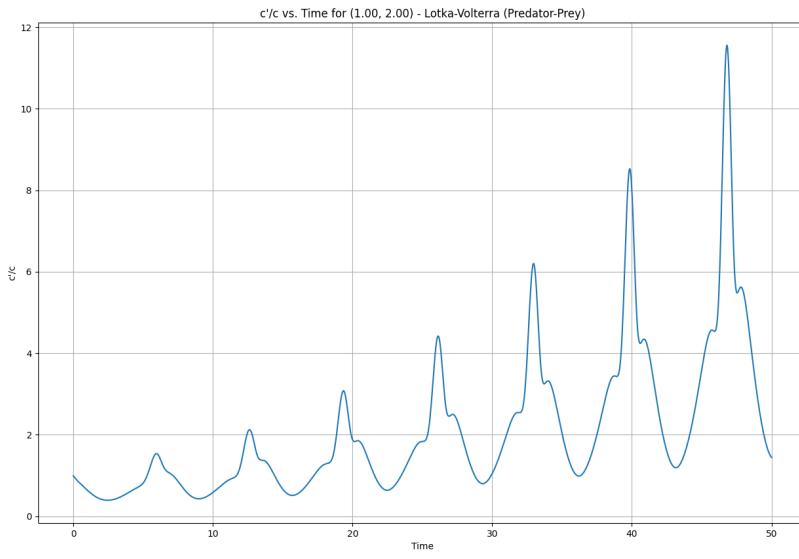


Figura 2.22: Grafico di $\mathcal{C}_{\vec{p}}(t)$ per il sistema di Lotka Volterra, il punto di equilibrio stabile si trova in $(1, 1)$. Il passo di integrazione è pari a 0.017 . Il punto di massimo assoluto nell'intervallo $[0, 50]$ ha un valore compreso tra 10 e 12 .

Si vede chiaramente come il valore della funzione nei punti di massimo locale continui ad incrementare indefinitamente. Questo non è dovuto ad una caratteristica intrinseca del modello, ma solamente ad errori di approssimazione numerici causati, come detto prima, da passi temporali non sufficientemente piccoli. Infatti, diminuendo dt fino a valori come 0.0001 si ottiene il seguente grafico:

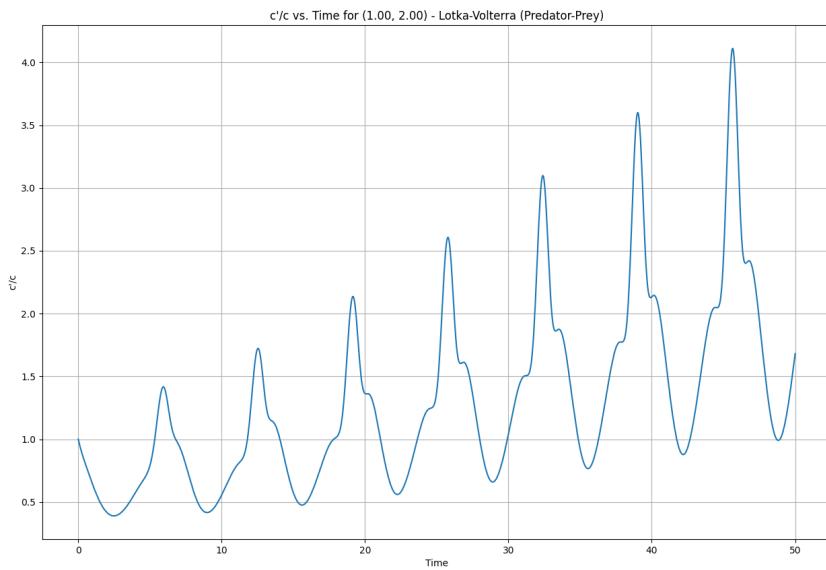


Figura 2.23: Grafico identico a quello in figura 2.22, cambia solamente dt . Il passo di integrazione è pari a 0.0001 . Il punto di massimo assoluto in $[0, 50]$ assume ora un valore tra 3.5 e 4 .

Si nota immediatamente come al diminuire del valore di dt diminuisce anche la differenza tra i punti di massimo assoluto.

Purtroppo questa soluzione ha dei limiti importanti: in primo luogo banalmente il numero di iterazioni crescerà molto rapidamente e conseguentemente anche i tempi di calcolo aumentano. Il secondo tipo di limitazione è decisamente più grave e nasce da un problema più sostanziale, quello della precisione numerica. È perfettamente inutile diminuire il passo d'integrazione sotto una certa soglia se la precisione numerica dei calcoli è finita, andare a ridurre ulteriormente questo valore peggiorerebbe solamente la precisione del grafico; ci sarebbero più punti nel quale la funzione viene calcolata, ma i calcoli sarebbero ancora meno precisi e l'errore introdotto da queste approssimazioni non può essere compensato solamente da un aumento nel numero di iterazioni.

Questa cosa ricorda vagamente il principio di indeterminazione di Heisenberg, in questo caso però l'incertezza è data dal prodotto tra un'incertezza temporale (il numero di passi di integrazione dovrebbe essere infinito teoricamente) e un'incertezza che deriva dalla precisione finita con la quale si può descrivere lo stato del sistema in un istante. A differenza del principio di indeterminazione di Heisenberg, questo tipo di indeterminazione è dettata completamente da limiti pratici e tecnici, non è un principio teorico che vale a priori.

Conclusioni e Riflessioni

3.1 L'Operatore di Stabilità

L'analisi dell'operatore di stabilità presentata in questa relazione ha coinvolto la trattazione di diversi argomenti. È evidente, in tutto il documento, una certa ramificazione dei contenuti, che potrebbe apparire del tutto arbitraria, e in effetti lo è. L'esplorazione dei vari argomenti e dei vari modelli non ha un fine preciso e la relazione stessa manca di un reale obiettivo specifico.

In realtà il fine di questa relazione è la relazione stessa, tale lavoro mi è stato utile per esercitare varie capacità, tra cui quella di scrivere una relazione che ricordi vagamente un documento pseudoscientifico, quella di gestire una *repository* in *Github* e quella di utilizzare un linguaggio tecnico. La relazione comunque non ha la pretesa di essere un trattato formale, piuttosto quello di presentare in modo tendenzialmente didattico, almeno nella forma, un argomento di carattere scientifico.

3.1.1 Analisi e osservazioni

Il modo in cui viene presentato l'operatore di stabilità, pur avendo cercato di strutturare il documento in maniera razionale, è decisamente lontano da quello che un approccio più scientifico richiederebbe. Un'analisi approfondita e sistematica richiederebbe però molto più tempo, lavoro e considerazioni rispetto all'importanza che l'operatore di stabilità stesso possa avere.

La relazione in questione tocca solamente punti che ritenevo sufficientemente interessanti per poterli inserire e discutere. Inoltre avrei voluto analizzare in modo più ortodosso la stabilità classica dei sistemi coinvolti, confrontandola poi con i risultati ottenuti dall'operatore di stabilità; ho tentato di farlo nelle sezioni **1.1.4** e **2.2.1** ma l'ho sempre trattato in maniera molto, forse troppo, qualitativa e approssimativa. Mi rendo conto che tutte le osservazioni che ho fatto sono prive di un reale fondamento matematico e sono piuttosto delle deboli asserzioni frutto di un'intuitività forse un po' troppo creativa. Giustificare ogni osservazione che ho fatto va nel modo più assoluto oltre le mie capacità, ma

anche ammesso che sia in grado di farlo, riportare tutte le dimostrazioni mi richiederebbe troppo tempo.

3.1.2 L’(in)utilità dell’operatore

Per quanto riguarda l’utilità di questo operatore di cui ho già discusso nel primo capitolo vorrei ulteriormente chiarire la questione. In primo luogo esistono strumenti molto più efficaci e coerenti per descrivere la stabilità di un sistema dinamico, il mio infatti è solo un tentativo di rappresentare una specifica sfaccettatura della stabilità di un sistema, argomento che ritengo estremamente interessante. Il modo con cui viene calcolato l’operatore di stabilità è molto specifico, decisamente inadatto per molti modelli (conservativi o che presentano elementi di forte instabilità) e comunque pesante e inefficiente per quanto riguarda il calcolo in sé. L’eccessiva specificità e inefficienza rendono l’operatore di stabilità essenzialmente inutile.

3.2 Implementazione dell’Operatore di Stabilità

3.2.1 Python e C++

La scelta di inserire nel documento una parte consistente e interamente dedicata all’implementazione in Python dell’operatore di stabilità riflette l’effettivo tempo impiegato per analizzare e studiare l’operatore nei vari modelli. Quasi ogni analisi necessaria per la stesura di questa relazione ha alla base un file scritto in Python che analizza una certa caratteristica. Gli *snippet* di codice che ho inserito in questa relazione sono una minima parte di quello che è stato utilizzato nel complesso. Le parti di codice inserite fanno parte del file principale, che permette di analizzare le caratteristiche più generali dei quattro sistemi dinamici discussi.

Inoltre ho inserito la sezione sull’ottimizzazione CUDA solamente per interesse personale e mi rendo conto che non aggiunge effettivamente alcun valore alla relazione in sé o all’analisi dell’operatore, è solo un elenco di nozioni di informatica applicate al caso particolare dell’operatore di stabilità.

Buona parte dell’analisi che sta alla base della relazione si è svolta tramite l’utilizzo dell’applicazione in C++. È comunque servito molto tempo soprattutto per organizzare tutto il materiale necessario all’applicazione in (Riva, D. 2025) e l’utilizzo di un sistema di *versioning* per rendere l’installazione di *C-Viewer* più semplice e organizzata, secondo lo standard del *Semantic Versioning*. Non ho inserito alcun riferimento specifico al codice in C++, principalmente per due motivi: in primo luogo perché avrebbe occupato un’intera sezione della relazione, che non avrebbe aggiunto un valore significativo allo studio dell’operatore, in secondo luogo perché non ho sufficienti conoscenze e familiarità con il linguaggio stesso. Per poter scrivere del codice in C++ avrei dovuto fare in gran parte affidamento ad un’IA, le poche e superficiali conoscenze che ho di

tale linguaggio mi permettono di comprendere in generale il funzionamento dell'applicazione ma non sono neanche lontanamente sufficienti per poter scrivere un testo coerente a riguardo¹.

3.2.2 IA

L'intelligenza artificiale è stato uno strumento necessario per la scrittura del codice sia in Python che in C++ e senza di esso non sarei mai riuscito a completare un'intera applicazione in così poco tempo. Il materiale che l'IA è stata in grado di generare in pochi secondi mi avrebbe richiesto probabilmente dei giorni interi per quanto riguarda Python e mesi per quanto riguarda C++. Nonostante questo l'utilizzo dell'IA è stato limitato esclusivamente al codice, ogni altro aspetto della relazione, dalla struttura all'analisi effettiva dell'operatore non l'ha coinvolta in nessun modo.

Le ragioni per questo sono molteplici, mi limito a riportare quella più rilevante. Come è stato dedotto dallo studio del gruppo di *Microsoft Research* (Lee, H.-P. et al. 2025) «*While GenAI can improve worker efficiency, it can inhibit critical engagement with work and can potentially lead to long-term overreliance on the tool and diminished skill for independent problem-solving*»

Lo studio di *Microsoft Research* sottolinea anche come, oltre ad una generale (ma non universale) diminuzione percepita nell'applicazione del pensiero critico, in particolar modo se si ha poca confidenza con le proprie capacità riguardo ad un argomento particolare o molta confidenza nelle capacità della *GenAI*, l'impegno investito nel pensiero critico si sposta, per esempio, dall'esecuzione concreta di una certa attività alla supervisione e gestione di questa attività svolta dalla *GenAI*.

Utilizzare in modo efficace un'IA per compiti complessi non è così scontato come potrebbe sembrare. La produzione di *C-Viewer* non è frutto di qualche ora di conversazione con Gemini. Come si legge anche in alcuni esempi riportati nell'articolo sopracitato, l'interazione stessa con la *GenAI* e la verifica che il codice generato sia coerente con il *framework* dell'applicazione, fattualmente corretto e funzionante, richiede comunque una quantità di lavoro non indifferente.

Personalmente ritengo che l'insieme delle attività cognitive necessarie per l'utilizzo corretto di un'IA siano importanti, ma non fondamentali, in quanto si pongono ad un livello di astrazione superiore rispetto a quelle che si vengono tradizionalmente a sviluppare senza l'utilizzo di strumenti di questo tipo. La capacità di raccogliere informazioni e sintetizzarle, di problem-solving e di esecuzione di attività specifiche, capacità che possono ora essere delegate ad una *GenAI*, sono il presupposto di quelle attività che si rendono necessarie per l'utilizzo di intelligenze artificiali. Queste attività non dovrebbero diventare, secondo me, primarie, ma essere secondarie e rappresentare un raffinamento

¹Cosa che tra l'altro non sono nemmeno sicuro di aver fatto nel caso di Python.

delle attività cognitive tradizionali. Invertire il rapporto tra queste attività porterebbe ad una totale dipendenza e subordinazione agli strumenti stessi, che diventerebbero gli unici a poter svolgere queste tipologie di attività così fondamentali.

Bibliografia

- Izhikevich, Eugene M. (2010). *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. Cambridge, MA: The MIT Press. ISBN: 9780262514200. URL: <https://mitpress.mit.edu/9780262514200>.
- Kirisanov, Artem (2024). *Elegant Geometry of Neural Computations*. URL: <https://github.com/ArtemKirisanov/Youtube-Videos/tree/main/2024/Elegant%20Geometry%20of%20Neural%20Computations>.
- Lee, Hao-Ping (Hank) et al. (apr. 2025). «The Impact of Generative AI on Critical Thinking: Self-Reported Reductions in Cognitive Effort and Confidence Effects From a Survey of Knowledge Workers». In: *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems*. Honolulu, HI: ACM. URL: <https://www.microsoft.com/en-us/research/publication/the-impact-of-generative-ai-on-critical-thinking-self-reported-reductions-in-cognitive-effort-and-confidence-effects-from-a-survey-of-knowledge-workers/>.
- Matplotlib Development Team (2024). *Choosing Colormaps in Matplotlib*. URL: <https://matplotlib.org/stable/users/explain/colors/colormaps.html>.
- Riva, Daniele (2025). *C-Viewer: Visualizzazione in tempo reale della stabilità di sistemi dinamici*. URL: <https://github.com/Deye27/C>.