Université Nice Côte d'Azur

A Universal Differential Equation Approach to Tumor Growth Modeling.

Bridging Biological Laws and Neural Dynamics

SINADINOVIC Marko

Abstract

This work leverages Universal Differential Equations (UDEs) to uncover hidden dynamics within tumor growth. Our approach integrates the Gompertz mechanistic model with a neural network to approximate unmodeled factors. While standard approaches rely on the Adjoint Method, our analysis reveals its instability for stiff biological systems. Consequently, we propose a high-precision architecture using Forward-Mode Sensitivity Analysis, enabling exact gradient computation via Dual Numbers. Numerical experiments demonstrate that this rigorous numerical strategy allows for the precise identification of the underlying mathematical structure of the missing dynamics, where traditional methods fail.

Contents

Introduction : From Biological Mechanisms to Hybrid Modeling

Context

When we study population dynamics we learn that a population, whether it be rabbits in a ocean or cells in a tissue, grows according to fixed laws. For those who remember their differential equations classes, the Gompertz or Logistic models seem to provide the perfect answer with a curve that starts slowly, accelerates, and eventually stabilizes when resources run out. On paper, it works perfectly. However, when we look closer at clinical reality we realize there is a significant part of the unknown. A tumor is not just a mathematical curve following a predefined path. It is a complex living entity that adapts, fights back against the immune system, and resists treatments in ways that standard equations failed to predict. We find ourselves in a paradoxical situation. We have physical laws that are globally true, but locally insufficient to describe the specific reality of a patient. We are missing a piece of the puzzle.

Instead of choosing between a physical model (too rigid) and a neural network (too opaque) we are going to merge them. We will keep the ordinary differential equation to guide the general behavior, what we know, and we will inject a Neural Network to learn what we don't know like the treatment response or the metabolic situation . This way we confront the rigor of physics with the flexibility of learning.

From Biological Phenomenon to Modeling

To find this missing piece, we are going to "dissect" the dynamics of a tumor to understand and translate it mathematically. We notice that there is a predictable phase which is the natural tendency of cells to multiply, it follows known physical laws like the Gompertz model. The tumor accumulates mass deterministically. But complementary to this, there is a hidden phase which is the response to a medicine or a sudden resistance. Now that we understand the phenomenon we must model it with the mathematical tools at our disposal.

Instead of choosing between a physical model (too rigid) and a neural network (too opaque), we are going to merge them. We will keep the ordinary differential equation to guide the general behavior, what we know, and we will inject a Neural Network to learn what we don't know like the treatment response or the metabolic situation . This way, we confront the rigor of physics with the flexibility of learning.

We are not just trying to predict the future size of a tumor, we are trying to identify the mechanism driving it. This leads us directly to the framework of Universal Differential Equations (UDEs).

Seeing to Understand

Now that we have managed to understand and model the phenomenon, the idea is to truly manipulate it. In advanced mathematics, it is easy to hide behind complex theorems and abstract proofs of convergence. Here we want to take a different path. We want to demystify Scientific Machine Learning.

To do this we will adopt a pragmatic approach. We will start by defining the mathematical playground : the Gompertz equation and the Adjoint method. But we will keep the theory focused on what is strictly necessary to understand the code, we are not going to prove the theorems or properties, that is not the purpose of this work. The core of our work will be experimental. We will set up a digital petri dish and we will watch the neural network learn. We want to see on a graph the moment where the machine understands the hidden law and superimposes its prediction on reality.

1 The Theory of Tumor Growth Modeling

Before introducing complex learning algorithms it is imperative to understand the underlying physical laws we intend to augment. In oncology the growth of a solid tumor is rarely linear. It follows specific dynamics governed by resource availability like oxygen, glucose and spatial constraints.

1.1 The Classical Gompertz Model

Mathematical Formulation and Biological Interpretation

The simplest model for population growth is the Malthusian model, which is a exponential model, where growth is proportional to the current population. But a tumor cannot grow indefinitely, it is limited by the carrying capacity of its host tissue.

To account for this saturation Benjamin Gompertz proposed a model in 1825 where the growth rate decays exponentially over time. The dynamics of the tumor volume $N(t)(t)$ are governed by the following Ordinary Differential Equation (ODE) :

$$(\Psi) : \begin{cases} \frac{dN}{dt} = f(t, N(t)) = rN(t)\ln\left(\frac{K}{N(t)}\right) \\ N(0) = N_0 \end{cases} \tag{1}$$

Where:

- $N(t) \in \mathbb{R}_+^*$ is the tumor size (volume or number of cells) at time t.
- $r > 0$ represents the intrinsic proliferation rate.
- $K > 0$ is the carrying capacity (the maximum size the tumor can reach).

Biologically, the term $\ln(K/N(t))$ acts as a brake. When $N(t)$ is small ($N(t) \ll K$), the term is large, and growth is rapid. As $N(t)$ approaches K, the term tends to 0, and growth halts. Indeed $\ln(K/N(t)) = \ln(K) - \ln(N(t))$.

Analytical Resolution and Stability Analysis

To legitimize the use of this model in a computational framework we must first ensure the problem is well-posed in the sense of Hadamard. It must satisfies three criteria :

- **Existence :** We must ensure that our differential equations actually yield a solution that aligns with the biological parameters we observe
- **Uniqueness :** It is vital that our model produces a single and clear trajectory
- **Stability :** The system must be resilient to small changes. In the real world, biological data is often noisy, so we need to be sure that slight experimental errors won't lead to divergent results

a) Existence and uniqueness We invoke the fundamental theorem of ODEs.

(Theorem) Cauchy-Lipschitz Theorem - Local Version
Consider the Cauchy Problem :

$$\begin{cases} \frac{dy}{dt} = f(t, y) \\ y(t_0) = y_0 \end{cases}$$

Let $\mathcal{O}$ be an open set in $\mathbb{R} \times \mathbb{K}^n$, and let $f : \mathcal{O} \to \mathbb{K}^n$ If f is continuous in t and locally Lipschitz continuous with respect to the state variable y, given any $(t_0, y_0) \in \mathcal{O}$, there exists some $\alpha > 0$ such that the Cauchy Problem $y(t_0) = y_0$ for (Ψ) possesses a unique solution y defined on the interval $[t_0 - \alpha, t_0 + \alpha]$.

Let $f(N) = rN(\ln K - \ln N) \in C^1(0, +\infty)$. Here N represents the state variable (tumor size). To check the Lipschitz condition we differentiate f with respect to the state variable N:

$$\frac{df}{dN} = r\ln(K/N) + rN\left(-\frac{1}{N}\right) = r(\ln(K/N) - 1)$$

Since f is continuously differentiable on the open domain $\mathbb{R}_+^*$ it is locally Lipschitz. Thus, for any initial condition $N_0 > 0$, there exists a unique solution. Since equation (Ψ) is a separable ODE, we can solve it explicitly.

b) Analytical Solution

Let us introduce the change of variable $u(t) = \ln(N(t)/K)$ and by differentiating with respect to time :

$$\frac{du}{dt} = \frac{d}{dt}(\ln N(t) - \ln K) = \frac{1}{N(t)}\frac{dN}{dt}$$

Substituting $\frac{dN}{dt}$ from the Gompertz equation:

$$\frac{du}{dt} = \frac{1}{N(t)}\left(rN(t)\ln\left(\frac{K}{N(t)}\right)\right) = r\ln\left(\frac{K}{N(t)}\right)$$

Noting that $\ln(K/N) = -\ln(N/K) = -u(t)$, this reduces the nonlinear Gompertz equation to a simple linear decay equation:

$$\frac{du}{dt} = -ru(t) \implies u(t) = u(0)e^{-rt}$$

Substituting back $N(t) = Ke^{u(t)}$, we obtain the closed-form solution:

$$N(t) = K\exp\left(\ln\left(\frac{N_0}{K}\right)e^{-rt}\right) \tag{2}$$

c) Fixed Points and Stability Analysis

Now we search for equilibrium states N^* (fixed points) such that : $f(N^*) = 0$ (the growth rate vanishes)

$$rN^*\ln\left(\frac{K}{N^*}\right) = 0$$

This equation yields two solutions:

- A Trivial equilibrium : $N_0^* = 0$ (extinction).
- $\ln(K/N^*) = 0 \implies N^* = K$ (Carrying capacity).

To determine the local stability of the non-trivial equilibrium $N^* = K$, we evaluate the sign of the derivative of the growth function with respect to the state variable N at the point K :

$$\frac{df}{dN} = \frac{d}{dN}\left[rN(\ln K - \ln N)\right] = r(\ln(K/N) - 1)$$

Evaluating at $N = K$:

$$\left.\frac{df}{dN}\right|_{N=K} = r\left(\ln(1) - 1\right) = -r$$

Since $r > 0$, the derivative is negative ($-r < 0$).

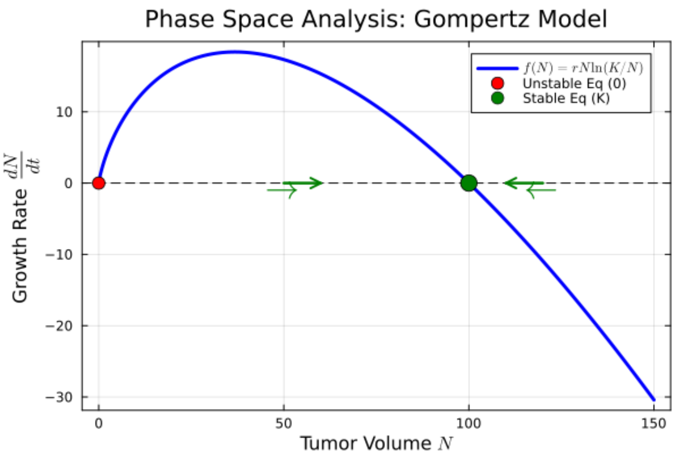

Figure 1:

To conclude rigorously we invoke a fundamental result of dynamical systems theory.

(Theorem) Lyapunov's Indirect Method / Linearization Principle Theorem

Let $\frac{d\mathbf{x}}{dt} = f(\mathbf{x})$ be an autonomous dynamical system where $f : \mathcal{D} \subset \mathbb{R}^n \to \mathbb{R}^n$ is continuously differentiable, $f \in C^1(\mathbb{R}^n)$. Let $\mathbf{x}^*$ be an equilibrium point ($f(\mathbf{x}^*) = 0$).

Define $J = Df(\mathbf{x}^*)$ as the Jacobian matrix of f evaluated at the equilibrium:

$$J_{ij} = \left.\frac{\partial f_i}{\partial x_j}\right|_{\mathbf{x}^*}$$

Let $\sigma(J)$ be the spectrum (set of eigenvalues) of J.

- If for all eigenvalues $\lambda \in \sigma(J)$, $\mathrm{Re}(\lambda) < 0$, then $\mathbf{x}^*$ is locally asymptotically stable, the equilibrium is a sink.
- If there exists at least one eigenvalue $\lambda \in \sigma(J)$ such that $\mathrm{Re}(\lambda) > 0$, then $\mathbf{x}^*$ is unstable (source or saddle).

Let's apply to the scalar case now ($n = 1$). In our one-dimensional Gompertz model, the Jacobian matrix reduces to the scalar derivative $J = f'(N^*)$. The single eigenvalue is $\lambda = f'(N^*)$. Thus the stability condition $\mathrm{Re}(\lambda) < 0$ simplifies directly to:

$$\left.\frac{df}{dN}\right|_{N^*} < 0$$

Since we calculated $\frac{df}{dN}(K) = -r$ and $r \in \mathbb{R}_+^*$, the eigenvalue is strictly negative, proving asymptotic stability.

It means any trajectory starting close to K will converge towards K and near $N = 0$ the growth rate is positive making 0 an unstable equilibrium. Said differently the tumor will naturally converge towards its carrying capacity K matching clinical observations of untreated tumors.

Limitations : Structural Bias and Missing Dynamics

While the Gompertz model provides a robust baseline for natural growth, relying on it exclusively creates a significant structural bias. By definition the equation (Ψ) assumes that the derivative $\frac{dN}{dt}$ is fully explained by the current state N and two constant parameters r and K.

In a clinical context this assumption collapses. A patient undergoing treatment is subject to external perturbations that the original equation cannot see. Let us assume the true dynamics of the tumor follow a perturbed law:

$$\frac{dN(t)}{dt} = \underbrace{rN(t)\ln\left(\frac{K}{N(t)}\right)}_{\text{Known Physics }(f)} + \underbrace{U(t, N(t))}_{\text{Unknown Physics}} \tag{3}$$

Here $U(t, N)$ represents the missing dynamics. It could model:

- A chemotherapy term $-C(t)N$, where $C(t)$ is the drug concentration.
- An immune response term that saturates over time.
- An acquired resistance mechanism where the efficacy of the drug decays.

If we force a pure Gompertz model ($U = 0$) onto clinical data generated by a system where $U \neq 0$, the optimizer will attempt to compensate for the missing term by distorting the values of r and K. We would obtain "optimal" parameters that are mathematically minimizing the error but biologically false.

We are therefore left with an inverse problem where the functional form of U is unknown. We cannot simply guess it. This motivates the central idea replacing the specific unknown function $U(t, N)$ with a Universal Approximator, a Neural Network, capable of learning the shape of the perturbation directly from the trajectory.

1.2 From ODEs to Universal Differential Equations (UDEs)

The Gray-Box Philosophy

We have established that the classical Gompertz equation is insufficient to capture complex clinical scenarios due to the unknown term $U(t, N)$. This leaves us with a dilemma. We cannot stick to a pure physical model (White-Box) because it is too rigid but we do not want to switch to a pure Deep Learning model (Black-Box) because we would lose the biological interpretability of parameters r and K.

We need a mathematical object capable of respecting the known physics while mimicking any continuous function to fill the gap. This is the essence of Gray-Box modeling.

To implement this philosophy we rely on the framework of Universal Differential Equations (UDEs) popularized by Rackauckas et al. (2020). The core idea is to embed a neural network directly inside the differential equation structure.

We define our hybrid system as follows :

$$\frac{dN(t)}{dt} = \underbrace{f_{phys}(N(t), p)}_{\text{Known Physics}} + \underbrace{\mathcal{NN}_\theta(N(t))}_{\text{Unknown Physics}} \tag{4}$$

Where:

- $f_{phys}(N, p) = rN \ln(K/N)$ the trusted Gompertz growth with physical parameters $p = \{r, K\}$.
- $\mathcal{NN}_\theta : \mathbb{R} \to \mathbb{R}$ is a neural network parametrized by a vector of weights and biases $\theta \in \mathbb{R}^d$ $card(d) < \infty$

It is important to understand that the neural network isn't predicting the future state $N(t + \Delta t)$ directly as a standard Recurrent Neural Network would. Instead it acts as a constituent of the vector field.

At each time step t the numerical solver evaluates the physical term passes the current state $N(t)$ through the neural network to compute the correction term then sums them up to get the total derivative $\frac{dN}{dt}$ and only then integrates this sum to move to the next step. This ensures that the learning process is constrained by the differential structure of the problem.

The problem shifts from finding a function U to finding the optimal parameters θ that minimize the distance between the solution of equation (4) and the observed data.

A legitimate question arises. Why do we choose a Neural Network to model the unknown term $U(N)$? Why not a polynomial or a Fourier series like usual ? The choice is not arbitrary. It relies on a strong result from functional analysis : the Universal Approximation Theorem (UAT). While we have no idea what the mathematical formula for the drug resistance or immune response looks like, we can safely assume it is a continuous process. The UAT gives us the mathematical guarantee that a neural network can mimic any continuous function, no matter how complex, provided it has enough neurons.

(Theorem) Universal Approximation Theorem - Cybenko and Hornik, 1991
Let $\sigma(\cdot)$ be a non-constant, bounded, and continuous activation function (e.g., sigmoid or tanh). Let $K \subset \mathbb{R}^n$ be a compact set. The space of continuous functions implemented by a single-hidden-layer neural network:

$$\mathcal{F} = \left\{ F(x) = \sum_{i=1}^{m} w_i \sigma(a_i^T x + b_i) \mid m \in \mathbb{N}, w_i, b_i \in \mathbb{R} \right\}$$

is dense in $C(K, \mathbb{R})$ with respect to the uniform norm.

In simpler terms, for any continuous function $f : K \to \mathbb{R}$ and any error tolerance $\varepsilon > 0$, there exists a network parameters configuration θ (with sufficient width m) such that:

$$\sup_{x \in K} |f(x) - \mathcal{NN}_\theta(x)| < \varepsilon$$

This theorem is the mathematical foundation of our project. It validates the form of equation (4).

$$\frac{dN(t)}{dt} = \underbrace{rN(t) \ln\left(\frac{K}{N(t)}\right)}_{\text{Known Physics } (f)} + \underbrace{\mathcal{NN}_\theta(N(t))}_{\approx U(N(t))} = \frac{dN}{dt} = \text{Gompertz}(N) + \underbrace{\mathcal{NN}_\theta(N)}_{\approx U(N)} \tag{5}$$

By invoking this theorem, we operate a fundamental shift in our solving strategy. Instead of searching for an unknown function in an infinite-dimensional space, we reduce the problem to finding a vector of parameters θ in $\mathbb{R}^d$. Also we no longer need to guess the equation, just need to find the optimal θ that minimizes the error.

However this theorem is an existence result not a constructive one. It tells us that a good θ exists but it does not tell us how to find it. Finding this optimal θ requires training the network inside the differential equation but it creates a new danger. The network could potentially learn the wrong thing. Before finding the right parameters we must ensure that our hybrid system is well-posed before calculations.

Structural Identifiability

We have built a system that adds a physical model and a neural network.

$$\frac{dN}{dt} = \text{Gompertz}(N) + \mathcal{NN}_\theta(N)$$

The question is : how do we know who is doing the work ?

Imagine we overestimate the proliferation rate r. The Gompertz term will predict a tumor that grows way too fast. The neural network, whose only goal is to minimize the error, will naturally learn a negative correction just to slow down the equation and match the data. Visually the final curve $N(t)$ will look perfect but scientifically it's a disaster. We would have a false physical parameter compensated by a fake neural term.

If we try to learn parameters r, K and weights θ simultaneously the problem is illposed. There is an infinite number of ways to combine a wrong physical model with a compensating neural network to fit the same dataset:

$$\text{Observed Data} \approx \underbrace{f_{phys}(p_{wrong})}_{\text{Too fast}} + \underbrace{\mathcal{NN}_{compensating}}_{\text{Brakes}}$$

In this scenario the discovery is just a mathematical illusion. The network isn't learning biology it's fixing our mistakes. To counter this we need to discipline the model. We cannot let the neural network touch the parts of the growth that we already understand. We therefore adopt a strict protocol mimicking a clinical trial :

1. We train only the Gompertz parameters (r, K) on untreated tumors. Here the neural network is turned off $(U = 0)$. This anchors the white-box model on solid ground.
2. We introduce the data from treated patients. Freeze r and K. The physical model is no longer allowed to change. Consequently any deviation from the natural growth must be captured by the neural network.

By structurally separating these steps we force the network to learn exactly the missing biological mechanism and nothing else.

But to find the optimal parameters θ we need to minimize the error between our prediction and the data. This requires calculating the gradient of the loss function with respect to the neural network weights. But how do we calculate a derivative through a simulation without exploding our computer's memory ? This question leads us to the implementation of the Adjoint Sensitivity Method.

2 The Adjoint Sensitivity Method

Before detailing our specific implementation, it is necessary to establish the mathematical framework of the Adjoint Sensitivity Method. Although we will later demonstrate why a Forward-Mode approach is superior for our specific low-parameter constraint (see Section 3), the Adjoint method remains the theoretical cornerstone of the Neural ODE field and serves as the primary benchmark for our work.

2.1 Formulation of the Inverse Problem

The Loss Function

In a real clinical setting we don't have a continuous video of the tumor growing. We only have snapshots taken during medical exams. For this reason we define our dataset $\mathcal{A}$ as a collection of M discrete observations:

$$\mathcal{A} = \{(t_i, N_i^{obs})\}_{i=1}^{M}$$

Where:

- M is the total number of measurements available e.g. if the patient came for a scan 10 times, $M = 10$
- t_i represents the specific date of the i-th exam e.g. $t_1 = \text{Day } 0, t_2 = \text{Day } 7 \ldots$
- N_i^{obs} is the observed tumor size at that specific time

Our goal is to find the neural network parameters θ such that our model's simulated trajectory, denoted $N(t, \theta)$, passes as close as possible to these observed points. That's why we define the Loss Function $J(\theta)$ as the Mean Squared Error (MSE) :

$$J(\theta) = \frac{1}{M} \sum_{i=1}^{M} \left\| N(t_i, \theta) - N_i^{obs} \right\|^2 \tag{6}$$

We notice the term $N(t_i, \theta)$ is the result of an Cauchy Problem :

$$\begin{cases} \frac{dN}{dt} = f(N, \theta) \\ N(t_0) = N_0 \end{cases}$$

This implies that every time we want to evaluate the error for a specific θ we have to simulate the entire patient's history from t_0 to t_{end}.

Regularization

If we let the neural network minimize $J(\theta)$ without constraints it tends to hallucinate. It might use massive weights to fit the noise in the data creating a wiggly curve that makes no biological sense.

To prevent this we apply Weight Decay. This consists to add a penalty term to the loss function to force the weights θ to remain small. So the total loss becomes :

$$Loss(\theta) = J(\theta) + \alpha \|\theta\|_2^2 \tag{7}$$

Where α is a hyperparameter controlling the strength of the penalty, usually small, e.g. 10^{-4}.

2.2 Derivation of the Continuous Adjoint State

To perform gradient descent we need the sensitivity of the objective functional $J(\theta)$ with respect to the parameters. Since the state $N(t)$ is constrained by a differential equation we cannot differentiate J directly. Instead we employ the Lagrangian formalism adapted to Banach spaces treating the ODE as a constraint in the space of square-integrable functions.

The Lagrangian Formalism and Constrained Optimization

First we establish the functional framework. We define the state trajectory N in the space of continuously differentiable functions $\mathcal{C} = C^1([t_0, t_f], \mathbb{R})$ and the parameter vector θ in $\mathbb{R}^d$.

a) The Constrained Minimization Problem

Our optimization problem is formally stated as minimizing the functional J on the manifold defined by the differential equation. We define the running cost density $L : [t_0, t_f] \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ as the instantaneous measure of the discrepancy between the simulation and the reality.

In our clinical setting observations are sparse. We possess a finite set of measurements $\mathcal{A} = \{(t_i, N_i^{obs})\}_{i=1}^{M}$. To incorporate these discrete points into the continuous calculus of variations framework, we represent the loss density L using Dirac delta distributions :

$$L(t, N(t), \theta) = \sum_{i=1}^{M} \frac{1}{2\sigma^2} \|N(t) - N_i^{obs}\|^2 \delta(t - t_i) \tag{8}$$

Where σ^2 represents the measurement noise variance and $\delta(\cdot)$ is the Dirac distribution. It bridges the gap between statistics and calculus. When we integrate this density over time, the property of the Dirac distribution $\int f(t)\delta(t - t_i)dt = f(t_i)$ recovers exactly the Sum of Squared Errors (SSE) :

$$\int_{t_0}^{t_f} L(t, N, \theta)dt = \sum_{i=1}^{M} \frac{1}{2\sigma^2} \|N(t_i) - N_i^{obs}\|^2$$

But mathematically minimizing $\sum(x - y)^2$ or minimizing $\sum \frac{1}{\sigma^2}(x - y)^2$ gives exactly the same optimal parameters θ because σ is just a multiplicative constant. Therefore we set the constant $\sigma = 1$:

$$J(N, \theta) = \int_{t_0}^{t_f} L(t, N, \theta)\, dt = \sum_{i=1}^{M} \frac{1}{2} \|N(t_i) - N_i^{obs}\|^2$$

The optimization problem is then formally written as :

$$\min_{\theta \in \mathbb{R}^d} \quad J(N, \theta) = \int_{t_0}^{t_f} L(t, N(t), \theta)\, dt \tag{9}$$

$$\text{subject to} \quad \frac{dN}{dt} - f(t, N, \theta) = 0, \quad \forall t \in [t_0, t_f] \tag{10}$$

b) Geometry of the Constraint

How do we strictly enforce a constraint that must hold at every single instant $t \in [t_0, t_f]$? In finite-dimensional optimization ($\mathbb{R}^n$) the classic Lagrange Multiplier Theorem allows us to adjoin constraints using a vector $\lambda \in \mathbb{R}^k$. Here our constraint is not a set of equations but a differential operator acting on a function space.

However we have a theorem that allows us to use $\lambda \in \mathbb{R}^k$. It's the Lagrange Multiplier Theorem for Banach Spaces. This theorem states that if we minimize a functional subject to an equality constraint $G(N) = 0$ where G maps to a Banach space Z, there exists a continuous linear functional $\lambda^* \in Z^*$ (the topological dual space) such that the Lagrangian is stationary at the optimum.

Here our constraint is not a single point but a continuous trajectory. To handle this rigorously we must extend the concept of the dot product to function spaces.

Let us define the residual of the differential equation as a function $e(N, \theta)$ living in the Hilbert space $\mathcal{H}$ of square-integrable functions $L^2([t_0, t_f])$ and our goal is to force this entire function to be the zero function :

$$e(N, \theta)(t) = \frac{dN}{dt} - f(t, N, \theta) = 0$$

To incorporate this functional constraint into our scalar objective J, we introduce a Lagrange multiplier λ. Mathematically λ acts as a linear functional in the topological dual space of L^2. We can represent this action using the canonical L^2 inner product. For any two real-valued functions u, v defined on $[t_0, t_f]$:

$$\langle u, v \rangle_{L^2} = \int_{t_0}^{t_f} u(t)v(t)\,dt \tag{11}$$

Think of this as the continuous limit of the standard dot product. Just as $\lambda^T x = \sum \lambda_i x_i$ for vectors the pairing in function space becomes an integral:

$$\langle \lambda, e(N, \theta) \rangle_{L^2} = \int_{t_0}^{t_f} \lambda(t) \underbrace{\left(\frac{dN}{dt} - f(t, N, \theta) \right)}_{\text{Constraint error}} dt \tag{12}$$

This defines $\lambda(t)$ not just as a variable but as a time-dependent function : the Adjoint State. It weights the importance of the constraint violation at each moment in time.

c) The Lagrangian Functional

Having defined the scalar product structure we can now construct the central object of our optimization method. We define the Lagrangian $\mathcal{L} : \mathcal{C} \times \mathbb{R}^d \times \mathcal{C}^* \to \mathbb{R}$. It's constructed by subtracting the inner product of the multiplier and the constraint from the objective functional:

$$\mathcal{L}(N, \theta, \lambda) = J(N, \theta) - \left\langle \lambda, \frac{dN}{dt} - f(N, \theta, t) \right\rangle_{L^2} \tag{13}$$

It can be rewritten in its integral form as follows :

$$\mathcal{L}(N, \theta, \lambda) = \int_{t_0}^{t_f} L(N, \theta, t)\,dt - \int_{t_0}^{t_f} \lambda(t) \left(\frac{dN}{dt} - f(N, \theta, t) \right) dt \tag{14}$$

This functional combines the objective we want to minimize (J) and the physics we must respect.

(Proposition) Consistency on the Feasible Manifold

Let $\mathcal{M}$ be the feasible manifold defined by the set of pairs (N, θ) that strictly satisfy the ODE constraint. For any feasible pair $(N, \theta) \in \mathcal{M}$, the residual term vanishes identically. Consequently :

$$\mathcal{L}(N, \theta, \lambda) = J(N, \theta), \quad \forall \lambda \in L^2([t_0, t_f])$$

This identity justifies the method of Lagrange Multipliers. It implies that the constrained extrema of the original functional J correspond to the stationary points of the unconstrained Lagrangian $\mathcal{L}$.

Instead of minimizing J directly which is difficult because of the complex constraint, our strategy is to find the triplet (N, θ, λ) that makes the Lagrangian stationary. Mathematically we require the total variation of $\mathcal{L}$ to vanish:

$$\delta\mathcal{L} = 0 \implies \begin{cases} \delta_N\mathcal{L} = 0 & \text{(Adjoint Equation)} \\ \delta_\lambda\mathcal{L} = 0 & \text{(State Equation)} \\ \delta_\theta\mathcal{L} = 0 & \text{(Gradient Condition)} \end{cases}$$

2.3 Derivation of the Adjoint Equation

We now calculate the variation of the Lagrangian $\mathcal{L}$ with respect to the state variable N. We consider a perturbation $\delta N \in \mathcal{C}$ around the optimal trajectory. For $\mathcal{L}$ to be stationary with respect to state variations,we require the Fréchet derivative $\delta_N\mathcal{L}$ to vanish for all arbitrary perturbations δN.

a) Total Variation Calculation

Let us perturb the trajectory $N(t) \to N(t) + \delta N(t)$ while keeping θ and λ fixed. The linear variation of $\mathcal{L}$ is given by:

$$\delta_N\mathcal{L}(\delta N) = \int_{t_0}^{t_f} \frac{\partial L}{\partial N}\delta N\, dt - \int_{t_0}^{t_f} \lambda(t)\left(\frac{d}{dt}(\delta N) - \frac{\partial f}{\partial N}\delta N\right) dt \tag{15}$$

Rearranging the terms to group those multiplying δN and those involving the derivative $\frac{d}{dt}(\delta N)$:

$$\delta_N\mathcal{L} = \int_{t_0}^{t_f} \left(\frac{\partial L}{\partial N} + \lambda^T \frac{\partial f}{\partial N}\right)\delta N\, dt - \int_{t_0}^{t_f} \lambda^T \frac{d}{dt}(\delta N)\, dt \tag{16}$$

b) Integration by Parts

The term $\frac{d}{dt}(\delta N)$ is problematic because it prevents us from factoring out δN. To resolve this we apply Integration by Parts to transfer the time derivative from the perturbation δN to the adjoint state λ.

Since $(\lambda, N) \in \mathcal{C}^2$:

$$\int_{t_0}^{t_f} \lambda(t)\frac{d}{dt}(\delta N)\, dt = [\lambda(t)\delta N(t)]_{t_0}^{t_f} - \int_{t_0}^{t_f} \frac{d\lambda}{dt}\delta N(t)\, dt \tag{17}$$

Substituting this result back into (16) our variation becomes :

$$\delta_N\mathcal{L} = \int_{t_0}^{t_f} \underbrace{\left(\frac{\partial L}{\partial N} + \lambda^T \frac{\partial f}{\partial N} + \frac{d\lambda^T}{dt}\right)}_{\text{Must be zero}}\delta N\, dt - \left[\lambda^T\delta N\right]_{t_0}^{t_f} \tag{18}$$

c) The Adjoint Differential Equation

For $\mathcal{L}$ to be a stationary point, the integral term must vanish for any arbitrary perturbation δN. This implies :

$$\frac{d\lambda^T}{dt} + \lambda^T \frac{\partial f}{\partial N} + \frac{\partial L}{\partial N} = 0$$

By taking the transpose of this entire equation to return to the standard column-vector notation where $\nabla L = (\frac{\partial L}{\partial N})^T$ we obtain the final Adjoint Equation :

$$\frac{d\lambda}{dt} = -\left(\frac{\partial f}{\partial N}\right)^T \lambda - \nabla_N L \tag{19}$$

This result describes how the sensitivity vector $\lambda(t)$ evolves over time. Mathematically strictly note that even if the original physical system f is highly non-linear like Gompertz growth, the adjoint equation is always a Linear Time-Varying system with respect to λ.

The presence of the transposed Jacobian $\left(\frac{\partial f}{\partial N}\right)^T$ is the signature of a dual problem. While the physical state N flows forward along the vector field the adjoint state λ flows upstream against it gathering information about the gradient.

However a linear ODE requires a boundary condition to be solvable. Since the equation depends on the derivative of the loss at time t its anchor naturally lies in the future, at $t = t_f$. This leads us to the boundary conditions derived in the next section.

The Transversality Condition

We are left with the boundary term from the integration by parts:

$$B = -\left(\lambda(t_f)^T \delta N(t_f) - \lambda(t_0)^T \delta N(t_0)\right) \tag{20}$$

We analyze the physical constraints on the endpoints :

- **$t = t_0$:** The initial condition of the tumor is fixed $N(t_0) = N_0$. Therefore no variation is allowed $\delta N(t_0) = 0$ and the term $\lambda(t_0)^T \delta N(t_0)$ vanishes naturally.
- **$t = t_f$:** The final state do not impose a target tumor size. Thus $\delta N(t_f)$ is arbitrary. For the boundary term to vanish for any $\delta N(t_f)$ we must impose $\lambda(t_f) = 0$

To compute the gradien we must first solve the state equation forward from t_0 to t_f and then solve the adjoint equation backward in time from t_f to t_0 using the final condition $\lambda(t_f) = 0$.

2.4 Gradient Computation

We have derived the state dynamics (forward) and the adjoint dynamics (backward). The final piece of the puzzle is to compute the gradient of the objective function J with respect to the parameters θ. This gradient is what feeds the optimizer (e.g., ADAM or BFGS) to update the weights.

Integral Form of the Gradient

We return to the total variation of the Lagrangian. The condition for stationarity with respect to parameters is $\delta_\theta \mathcal{L} = 0$. Differentiation of the Lagrangian functional with respect to θ gives :

$$\nabla_\theta J = \nabla_\theta \mathcal{L} = \int_{t_0}^{t_f} \left(\frac{\partial L}{\partial \theta} + \lambda(t)^T \frac{\partial f}{\partial \theta}\right) dt \tag{21}$$

In our specific case the vector field is defined as $f(N, \theta) = \text{Gompertz}(N) + \mathcal{NN}_\theta(N)$. The mechanistic part (Gompertz) does not depend on the neural network weights θ. Therefore the Jacobian reduces to the neural network's contribution :

$$\frac{\partial f}{\partial \theta} = \frac{\partial \mathcal{NN}_\theta(N)}{\partial \theta}$$

Assuming the regularization is explicit outside the integral, e.g. $+\alpha\|\theta\|^2$, the final gradient formula is :

$$\frac{dJ}{d\theta} = \int_{t_0}^{t_f} \lambda(t)^T \frac{\partial \mathcal{NN}_\theta(N(t))}{\partial \theta}\, dt + 2\alpha\theta \tag{22}$$

This integral has a clear physical meaning. The term $\frac{\partial \mathcal{NN}}{\partial \theta}$ represents how the network's output changes locally when weights are tweaked. The adjoint state $\lambda(t)$ acts as a time-dependent weight, it amplifies the gradient at times where the model makes large errors i.e. where sensitivity is high and suppresses it where the model is already accurate.

3 Computational Implementation in Julia

3.1 Complexity Analysis

When training a neural network embedded in a differential equation, the choice of the gradient computation method is not merely a technical detail but a fundamental trade-off between memory efficiency and numerical stability. This decision depends on two competing dimensions:

1. **The Parameter Space :** P the number of trainable parameters in the network.
2. **Numerical Stability :** The requirement for exact gradient computation which is crucial for scientific discovery where we aim to identify physical laws rather than just fitting curves.

The following table summarizes why, contrary to standard Deep Learning trends, we diverged from the Adjoint method for this specific high-fidelity simulation:

Method	Complexity & Constraints	Suitability for this Project
1. Forward-Mode Sensitivity	**Complexity:** $O(P)$ Calculates exact derivatives by extending the state vector with the Jacobian. Highly stable and precise.	**Selected Strategy.** Since our optimized anti-stiffness architecture is compact ($P \approx 300$) this method offers the highest numerical precision without prohibitive computational cost.
2. Discrete Backprop (BPTT)	**Memory:** $O(T \times N_{state})$. Requires storing the state at *every* step of the solver (treating the solver as a giant RNN).	**Infeasible.** For high-fidelity explicit solvers like `Vern7` ($T > 10^4$) this approach hits the memory wall crashing GPU/CPU RAM.
3. Adjoint Method	**Memory:** $O(1)$ (Constant). Solves the ODE backward in time to reconstruct gradients.	**Rejected.** It's memory efficiency but reverse-mode integration often suffers from numerical drift masking the true physical dynamics of biological systems.

Table 1: Comparison of gradient computation strategies.

Multiple Dispatch and the Composability Advantage

A recurring hurdle in scientific computing is the Two-Language Problem, where models are prototyped in a high-level language but rewritten in C or Fortran for execution. Julia bypasses this by using Multiple Dispatch a paradigm that allows the compiler to generate specialized machine code based on the types of all function arguments.

In the context of our tumor growth model this results in unparalleled composability :

- **Type-Agnostic Solvers :** The ODE solvers in `DifferentialEquations.jl` do not know they are integrating a neural network. They are written to handle any callable object that satisfies the derivative interface.
- **Vertical Integration :** Because the neural networks via `Lux.jl` and the solvers are both written in native Julia automatic differentiation tools like `Zygote.jl` can differentiate through the entire solver's logic.

3.2 Sensitivity Analysis

The bridge between our numerical solver and the gradient-based optimizer is the sensitivity analysis. This choice determines how the gradients of the loss function with respect to the parameters θ are propagated through the temporal integration of the tumor dynamics.

a) Continuous Adjoint (The Standard Approach)

This framework treats the ODE as a continuous functional object, deriving an Adjoint ODE to propagate gradients backward in time. While highly memory efficient because it avoids storing intermediate solver stages the resulting gradient corresponds to the ideal mathematical ODE rather than the specific numerical approximation. In stiff biological systems this adjoint mismatch can destabilize the training process leading to drifting gradients.

b) Forward-Mode Sensitivity (Our Approach)

Unlike standard backpropagation which traverses a computational graph backward, this approach computes the sensitivities $\frac{du}{d\theta}$ simultaneously with the solution $u(t)$. It leverages dual numbers, an algebraic trick where every number carries its own derivative e.g. $x + \epsilon x'$. As the solver advances from t to $t + dt$ it automatically updates both the state of the tumor and the exact derivative of that state with respect to the neural weights.

3.3 Conclusion on Implementation Strategy

While the Adjoint method is the industry standard for large-scale Deep Learning e.g.Computer Vision with millions of parameters, our focus prioritizes numerical exactness.

Consequently we adopt Forward-Mode Sensitivity Analysis via `ForwardDiff.jl`. This approach allows us to compute exact gradients for our compact architecture ensuring that the optimization landscape is not corrupted by the approximation errors inherent to reverse time integration. This choice effectively transforms our UDE into a robust solver capable of uncovering the hidden trap dose dynamics.

This means that when we compute the gradient of our loss function we aren't just treating the ODE solver as a black box external library. The compiler optimizes the interaction between the neural network's weights θ and the solver's internal stages k_i as a single unified mathematical operation.

Having established the theoretical framework for our optimization strategy, we now transition from the mathematical "why" to the structural "how" detailing the implementation of these dynamics.

4 Software Architecture

Implementing a Universal Differential Equation requires a tight integration between deep learning primitives and numerical analysis tools. We rely on the `SciML` (Scientific Machine Learning) ecosystem in Julia specifically `DifferentialEquations.jl` for the solver engine and `Lux.jl` for the neural parameterization.

4.1 ODE Solvers

The numerical integrator is the beating heart of our training loop. To understand the sophisticated algorithms used in `DifferentialEquations.jl` we must first revisit the mathematical foundation of ODE integration the Runge-Kutta family.

Explicit Runge-Kutta Methods

Numerical integration aims to solve $\frac{dy}{dt} = f(t, y)$ by stepping forward in time: $y_{n+1} = y_n + \Delta y$. The classical pedagogical approach is the Runge-Kutta 4 (RK4) method. It consists of taking four samples of the derivative to construct a weighted average. For a step size h:

$$k_1 = f(t_n, y_n) \qquad \text{(Derivative at the beginning)} \tag{23}$$

$$k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \qquad \text{(First estimate at midpoint)} \tag{24}$$

$$k_3 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \qquad \text{(Second estimate at midpoint)} \tag{25}$$

$$k_4 = f(t_n + h, y_n + hk_3) \qquad \text{(Estimate at the end)} \tag{26}$$

The next state is computed by a specific weighted sum known as Simpson's quadrature:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(h^5) \tag{27}$$

Although RK4 is robust for simple problems, it fails when applied to stiff equations systems where variables evolve at drastically different speeds. In a UDE the neural network $\mathcal{NN}_\theta(x)$ can initially produce high frequency oscillations. If we applied a fixed step solver like RK4 we would be forced to reduce h to infinitesimal values ($h < 10^{-6}$) to prevent the solution from exploding making training computationally intractable.

Adaptive Step Sizing and Embedded Methods

To introduce numerical safety and efficiency, we transition to Adaptive Step Sizing. Modern solvers do not use a single method but a pair of methods embedded in one another.

Formalized by the Butcher Tableau (Eq. 28) these methods compute two approximations of the next state using the same intermediate slopes k_i but with different weighting vectors $\mathbf{b}$ (high order) and $\hat{\mathbf{b}}$ (lower order) :

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \quad \text{(Solution)} \\ & \hat{\mathbf{b}}^T \quad \text{(Error Estimator)} \end{array} \tag{28}$$

This allows the solver to estimate the local error $\epsilon = \|y_{n+1} - \hat{y}_{n+1}\|$ at virtually zero cost. The step size h is then dynamically adjusted :

$$h_{new} = h \cdot \left(\frac{\text{Tolerance}}{\epsilon}\right)^{\frac{1}{\text{Order}+1}}$$

Verner's Method

For our specific goal of scientific discovery requires higher precision. We need to distinguish subtle biological signals like the drug interaction from numerical noise. We adopted the Verner's 7/6 Runge-Kutta method denoted `Vern7`.

Property	**Specification**
Algorithm	Verner's 7/6 (`Vern7`)
Numerical Order	7^{th} Order (Solution)
Error Estimator	6^{th} Order (Adaptive Step Control)
Target Tolerance	High Fidelity ($\approx 10^{-10}$)
Classification	**Explicit** Runge-Kutta

Table 2: Technical specifications of the chosen numerical integrator.

The choice of an explicit solver for a biological problem might seem counter intuitive. Typically biological models are stiff which forces explicit methods to take infinitesimally small steps to avoid numerical explosion. This necessitates slow implicit solvers like Rosenbrock methods. We bypass this by enforcing a `Tanh` activation on the network's output.

$$\frac{dN}{dt} = \text{Physics}(N) + \alpha \cdot \tanh(\mathcal{NN}_\theta(N, t)) \tag{29}$$

It bounds the derivative of the learned term within $[-\alpha, +\alpha]$. This simple structural constraint limits the Lipschitz constant of the differential equation. This combination of `Vern7` for precision and architecture stabilization for speed allows us to reduce the simulation time by orders of magnitude compared to traditional stiff solvers.

4.2 Neural Network Framework

While the choice of the ODE solver determines the stability of the integration, the choice of the neural network framework determines the flexibility of the optimization. For this study we depart from standard implicit frameworks like PyTorch and Flux.jl and adopt `Lux.jl` a library designed around the paradigm of Explicit Parameter Handling.

The Paradigm Shift : Implicit vs. Explicit State

Traditionaly layers are typically stateful objects. A layer internally stores its weights W and biases b. While convenient for standard supervised learning this object oriented approach creates friction in SML.

Differential equation solvers mathematically treat parameters as a single contiguous vector θ. Traditional frameworks encapsulate these weights inside mutable layer objects necessitating costly destructuring operations at every integration step. `Lux.jl` resolves this friction by strictly decoupling the function topology from the data. This design transforms the neural network into a pure immutable function where the parameters are passed explicitly as arguments.

Mathematical Formulation

Mathematically a Lux neural network is defined as a function f that maps an input x, a parameter set θ denoted as ps and an internal state S denoted as st to an output :

$$(y, S_{new}) = f(x, \theta, S) \tag{30}$$

This aligns perfectly with the definition of a UDE. When defining the vector field for the ODE solver we simply inject the network as a function call inside the derivative calculation :

$$\frac{dN}{dt} = \text{Growth}(N) - \text{Kill}(N, t) + \underbrace{\mathcal{NN}_\theta(N, t, S)}_{\text{Neural Augmentation}} \tag{31}$$

Implementation Advantages

The adoption of this functional approach provides three decisive architectural advantages for our project.

Feature	**Impact**
1. Zero-Overhead Abstraction	**Speed :** Parameters are passed as explicit vectors. This allows the solver to compute exact gradients directly via `ForwardDiff` bypassing the overhead of object-oriented tracking.
2. Determinism	**Reliability :** Separating weights (ps) from state (st) makes the network a pure mathematical function. Identical inputs yield identical outputs, eliminating bugs in numerical debugging.
3. Thread Safety	**Scalability :** The model structure is immutable and stateless. It can be evaluated in parallel across multiple CPUs without race conditions, enabling fast Monte Carlo simulations.

Table 3: Architectural advantages of Explicit Parameter Handling.

4.3 Network Topology and Initialization

```
# The Anti-Stiffness Architecture
nn = Lux.Chain(
    Lux.Dense(2, 16, swish),  # Non-linear feature extraction
    Lux.Dense(16, 16, swish), # Deep representation
    Lux.Dense(16, 1, tanh)    # Bounded Output [-1, 1])
```

Listing 1: Implementation of the Anti-Stiffness Architecture using Lux.jl

- **Swish Activation ($x \cdot \sigma(x)$) :** Chosen over ReLU to ensure smoothness (C^∞) and non-zero gradients preventing dead neuron stagnation during the initial optimization phase.
- **Tanh Output :** This layer allows the use of explicit solvers by bounding the vector field.

4.4 Automatic Differentiation Mechanics [SOON]

References

[1] C. Rackauckas, Y. Ma, J. Martensen, et al., *Universal Differential Equations for Scientific Machine Learning*, arXiv preprint arXiv:2001.04385, 2020.

[2] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, D. Duvenaud, *Neural Ordinary Differential Equations*, Proceedings of NeurIPS, 2018.

[3] K. Hornik, *Approximation capabilities of multilayer feedforward networks*, Neural Networks, 4(2), 251-257, 1991.

[4] E. Hairer, G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, Springer Series in Computational Mathematics, 2nd Edition, 1996.

[5] G. Allaire, *Analyse numérique et optimisation*, Éditions de l'École Polytechnique, 2005.

[6] A. K. Laird, *Dynamics of tumor growth*, British Journal of Cancer, 18(3), 490–502, 1964.

[7] S. Benzekry, et al., *Classical Mathematical Models for Description and Prediction of Experimental Tumor Growth*, PLoS Computational Biology, 10(8), 2014.

[8] C. Rackauckas, Q. Nie, *DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem*, The Journal of Open Research Software, 5(1), 2017.

[9] J. Revels, M. Lubin, T. Papamarkou, *Forward-Mode Automatic Differentiation in Julia*, arXiv preprint arXiv:1607.07892, 2016.

[10] The SciML Open Source Software Organization, *SciML: Scientific Machine Learning*, https://sciml.ai (Consulté en 2024).

Appendix

Appendix: Legal Notices and Credits

Use of AI

For this project we utilized artificial intelligence to assist with orthographic corrections and the drafting of specific sections to ensure fluid transitions and enhance readability. We also leveraged these tools to generate or optimize segments of the simulation code particularly for the visualization and graphical components. We have taken great care to rigorously verify all generated content and have clearly identified any section entirely produced by AI to maintain total transparency in our professional practice as developing mathematicians.