TEAM 5

Tuberculosis with Differential Infectivity

RESEARCH PROPOSAL

NUMERICAL METHODS COURSE SBEG108

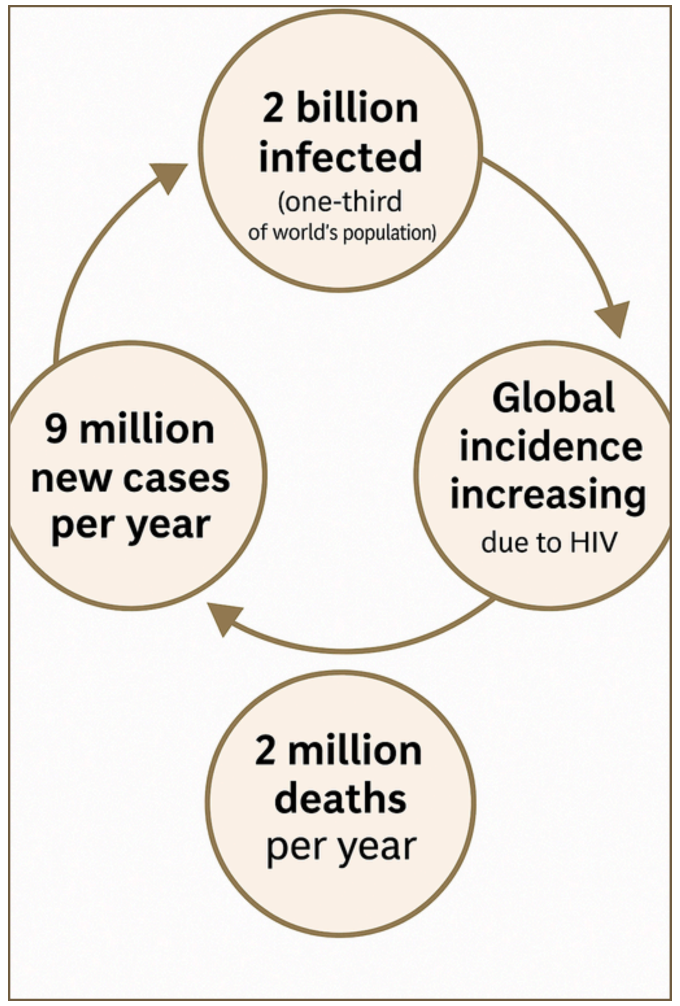
TABLE OF CONTENTS

SECTION 01

Introduction

Problem Definition

- Tuberculosis (TB) is a contagious disease caused by Mycobacterium tuberculosis, primarily affecting the lungs.
- It remains one of the world's most widespread infectious diseases, with nearly 2 billion people infected.
- Despite ongoing control efforts, global TB cases are rising, largely due to its strong link with HIV infection.

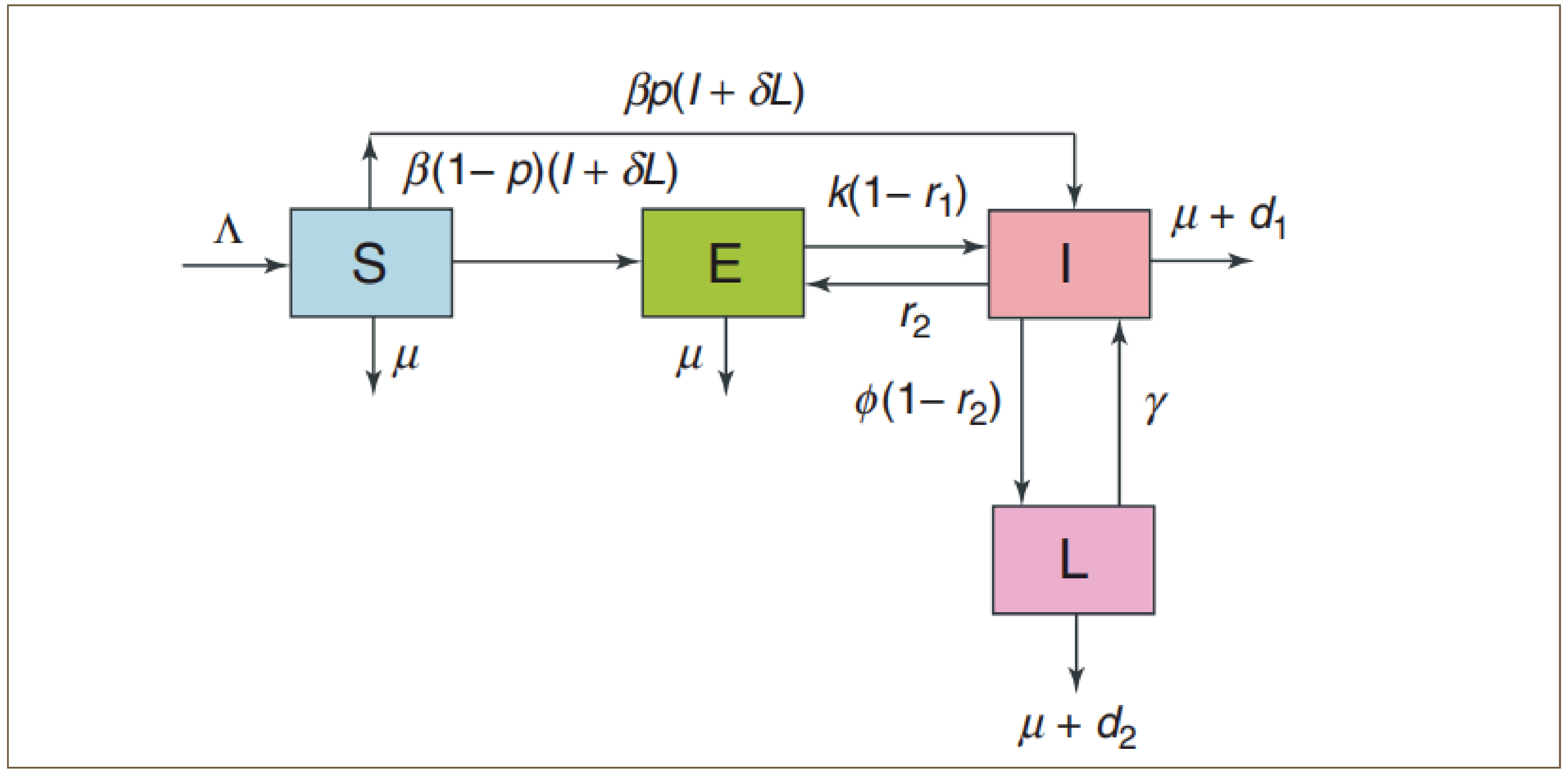

2 billion infected (one-third of world's population)
Global incidence increasing due to HIV
2 million deaths per year
9 million new cases per year

SECTION 02

SEIL Model & ODE’s

S E I L MODEL

S	Susceptible (healthy people who can potentially catch TB)
E	Exposed (but not yet infectious)
I	Infectious
L	Latent (post-infection or partially treated)



Y is the vector of all compartments (Y=[S,E,I,L])

Modeling

INPUTS

1) Model Parameters:

Parameter	Description	Estimated Value
Λ	Recruitment rate	2
β	Transmission rate	0.025
δ	Infectivity of L compared to I	1
p	Proportion rapidly progressing	0.3
μ	Natural death rate	0.0101
k	Progression rate from E to I	0.005
r_1	Effectiveness of chemoprophylaxis	0
r_2	Rate of recovery from I	0.8182
φ	Rate from I to L	0.02
γ	Reactivation rate from L to I	0.01
d_1	Death rate from I	0.0227
d_2	Death rate from L	0.2

INPUTS

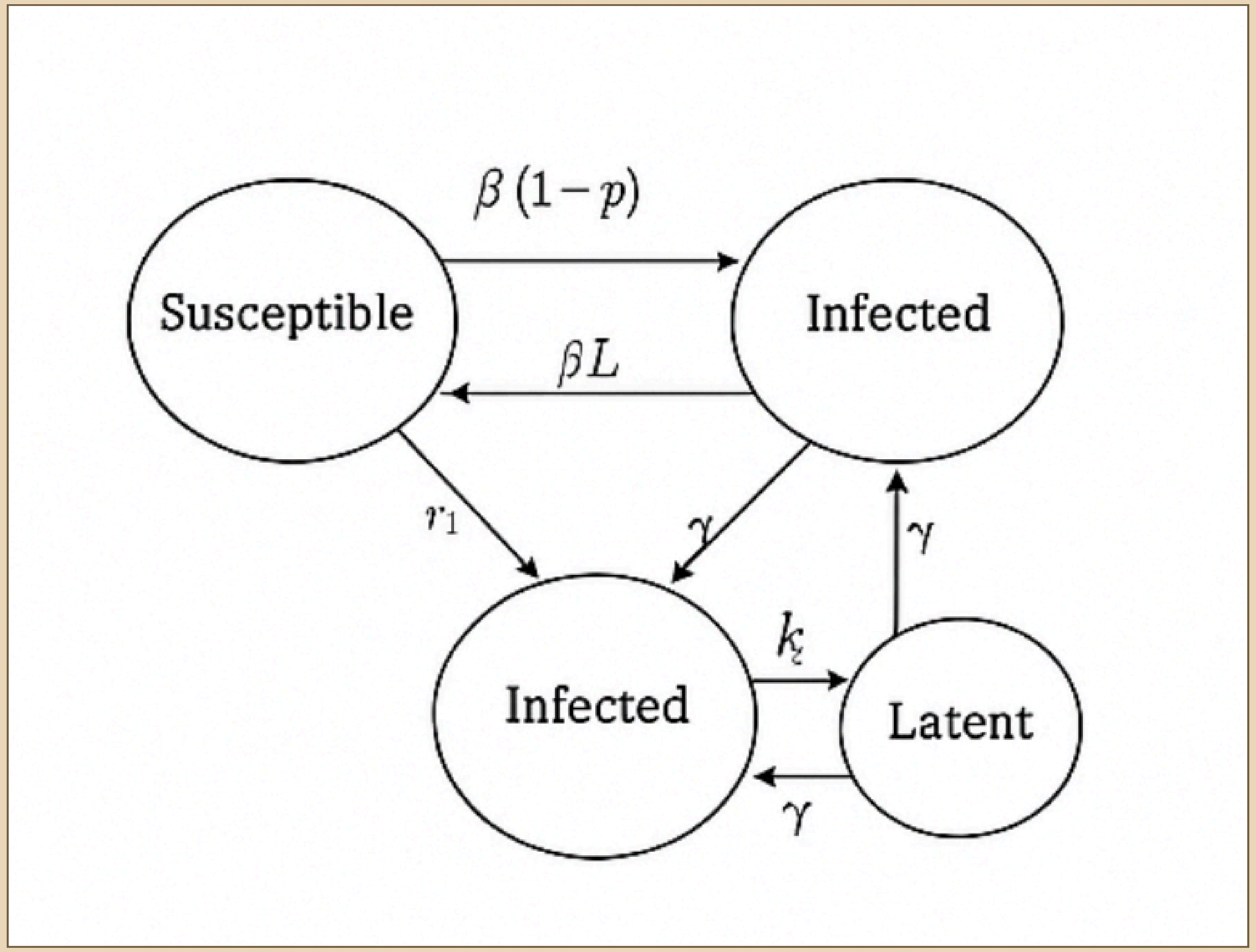
2) Time (t) :
Independent variable over the simulation period (e.g., 0 to 20 years).

3) Initial Conditions (Y_0):

S(0): Susceptible population = Λ / μ (198.01982)
E(0): Exposed individuals = 1.0
I(0): Infectious individuals = 0.0
L(0): Lost-to-follow-up individuals = 0.0

OUTPUTS

S	Susceptible
E	Exposed (but not yet infectious)
I	Infectious
L	Latent (post-infection or partially treated)



ODE'S

Equation 1: Governs the change in the number of **susceptible** individuals

$$\frac{dS}{dt} = \Lambda - \beta S(I + \delta L) - \mu S$$

- Λ is the recruitment rate (e.g., births or entries into the population) and has a positive effect

- β S (I + δL) is the rate at which susceptible people become exposed , it's a non linear term and has the strongest negative effect when L and I are high .

- They get infected through contact with infectious (I) or latent (L) individuals.

- δ is a scaling factor for how infectious L is compared to I.

- μS is the natural death rate of the susceptible group.

ODE'S

Equation 2: Governs the change in the number of **exposed** individuals

$$\frac{dE}{dt} = \beta(1-p)S(I+\delta L)+r_2 I-[\mu + k(1 - r_1)]\,E$$

Inflow (positive terms):

- $\beta(1 - p)\ S\ (I + \delta\ L)$: Individuals who get infected but don't immediately become infectious is the main source of new exposed individuals
- $r_2\ I$: Previously infectious individuals who relapse into exposed

Outflow (negative terms):

- Natural death occurs at a rate μ
- $k(1 - r_1)\ E$: Progression to the infectious stage happens at a rate k, adjusted by the factor (1–r1), where:
  - r1 is the fraction diverted away from I (into latent or lost compartments).
  - this term has the strongest negative effect assuming r1 is zero, which means the **chemoprophylaxis** is completely ineffective, i.e., no one is prevented from progressing from E to I.

****Chemoprophylaxis** is the use of medication to prevent disease like a vaccine or pills

ODE'S

Equation 3: Governs the change in the number of **infectious** individuals

$$\frac{dI}{dt} = \beta p S(I+\delta L)+k(1-r_1)E+\gamma L-[\mu + d_1 + \phi(1 - r_2) + r_2]\, I$$

Inflow (positive terms) :

- S - directly via the fraction p of new infections has a fast progression - immediate infection becomes infectious
- E - progressing via $k(1 - r_1)$ from E to infectious this and the previous point has the major inflows to I pool
- L - reactivating with γ (reactivation from latent)

Outflow:

- Natural death μ
- d1 I , Death due to TB
- Progression to L with $\varphi(1 - r_2)$ It is Partial treatment leading to latent TB
- successful Treatment with r_2

ODE'S

Equation 4: Models the **latent** compartment

$$\frac{dL}{dt} = \phi(1 - r_2)I - (\mu + d_2 + \gamma)L$$

Inflow:

- From I via $\varphi(1 - r_2)$ → those not successfully treated but not fully infectious anymore

Outflow:

- Natural death μ
- d_2L, Death in latent
- Reactivation to I with rate γ (Go Back to infectious) it has the strongest negative effect especially if reactivation is common

****Reactivation** refers to when a person who was previously infected with tuberculosis (but not actively sick) develops active, contagious TB later, usually after a long period of latency.

SECTION 03

TB Modeling Approach

TB Modeling Approaches

We surveyed several recent works tackling tuberculosis (TB) modeling through both traditional **numerical** and modern **machine learning** techniques:

1. Kanwal et al. developed a 5-compartment TB model with treatment and recovery, but fixed-step methods like RK4 led to numerical instability due to system stiffness.
2. Tadesse et al. (2024) incorporated vaccination and reinfection into the TB model, offering realistic dynamics but focusing only on theoretical analysis.
3. Syafruddin et al. applied RK4 to a basic SIR model using real TB data, showing the method’s reliability and accuracy even in simple models.
4. Raissi et al. introduced PINNs, a method that embeds differential equations into neural network training, achieving highly accurate solutions without requiring small time steps.
5. Pal et al. applied PINNs to TB modeling in diabetic patients using DeepXDE, combining physics-informed and data-driven loss to handle complex, uncertain dynamics effectively.

SECTION 04

METHOD 00

LSODE-Reference Method

LSODES (Livermore Solver for ODEs)

Definition:

- A powerful numerical solver for both stiff and non-stiff ODE systems.
- Automatically detects stiffness

Switches between two methods:

- Adams-Moulton (predictor-corrector) – for non-stiff regions
- BDF (Backward Differentiation Formula) – for stiff regions

Controls:

- Step size
- Method order
- Local truncation error

METHODOLOGY

Numerical Implementation(in R):

- Implemented in R using **deSolve** (R package) and **lsodes()** (ODE solver)
- RHS equations coded in **tb_1.R** (function that calculates the RHS of the TB ODE system at any given time t and for any given state vector y=[S,E,I,L])

Two initial cases:

- Case 1 (ncase = 1): Equilibrium
- Case 2 (ncase = 2): Epidemic trigger with E(0)=1

- Time range: 0 to 20 years, 41 points
- Tolerances: rtol = 1e-8, atol = 1e-8

METHODOLOGY

LSODES Behind the Logic:

- Starts with Adams-Moulton (non-stiff)
- Monitors integration behavior:
  If step size is limited by stability → switches to BDF
- In BDF mode:
  Uses Newton-Raphson iteration
  Computes/approximates Jacobian
- Always seeks the largest safe step size

RESULTS:

Case 1: Equilibrium

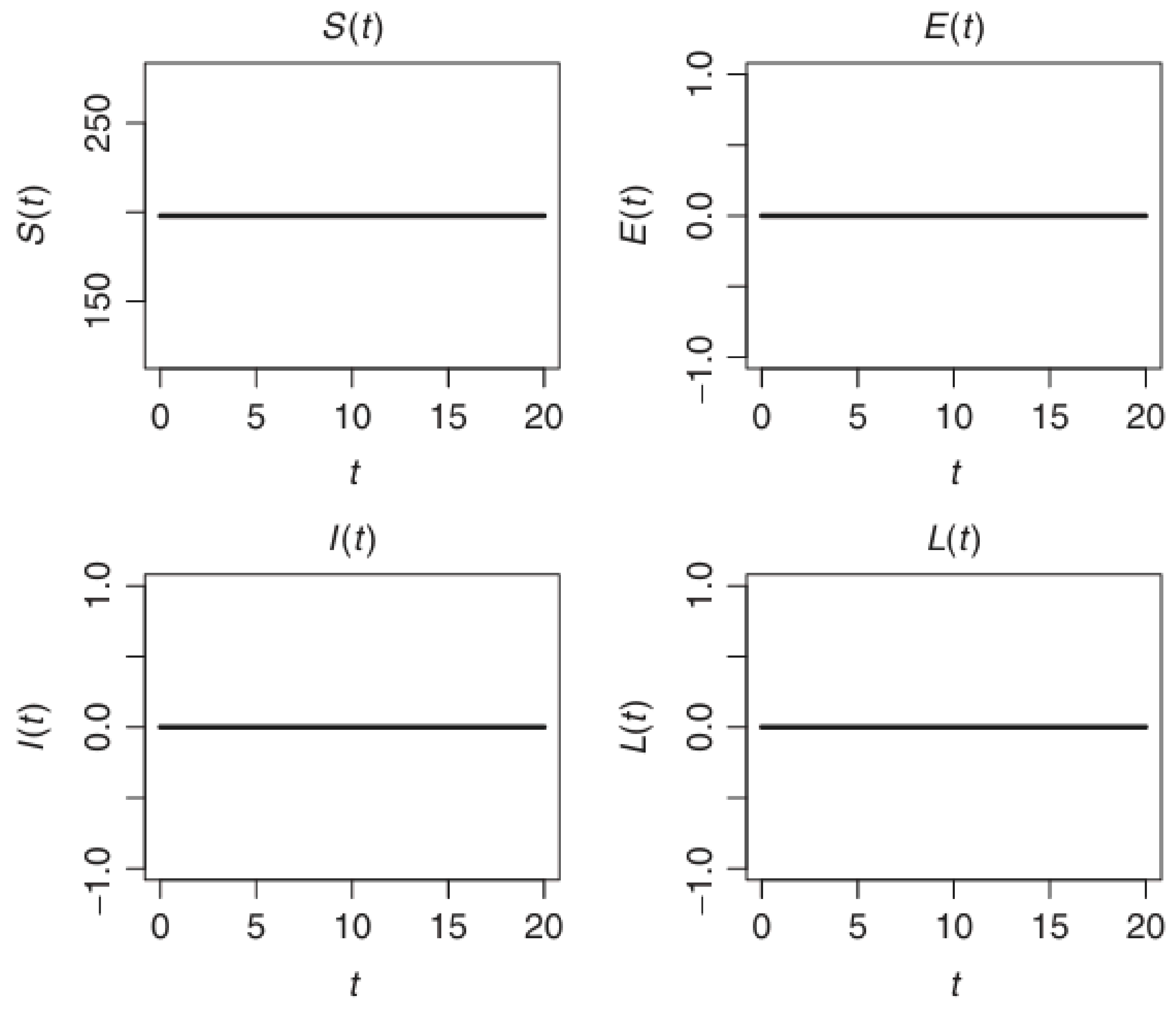
- Initial Conditions: E=I=L=0, S=Λ/μ

Result:

- No dynamics, system remains at equilibrium

LSODES did **minimal work**:

- Number of ODE calls = **53**



RESULTS:

Case 2: Epidemic

- Initial Conditions: E(0)=1, all others at equilibrium

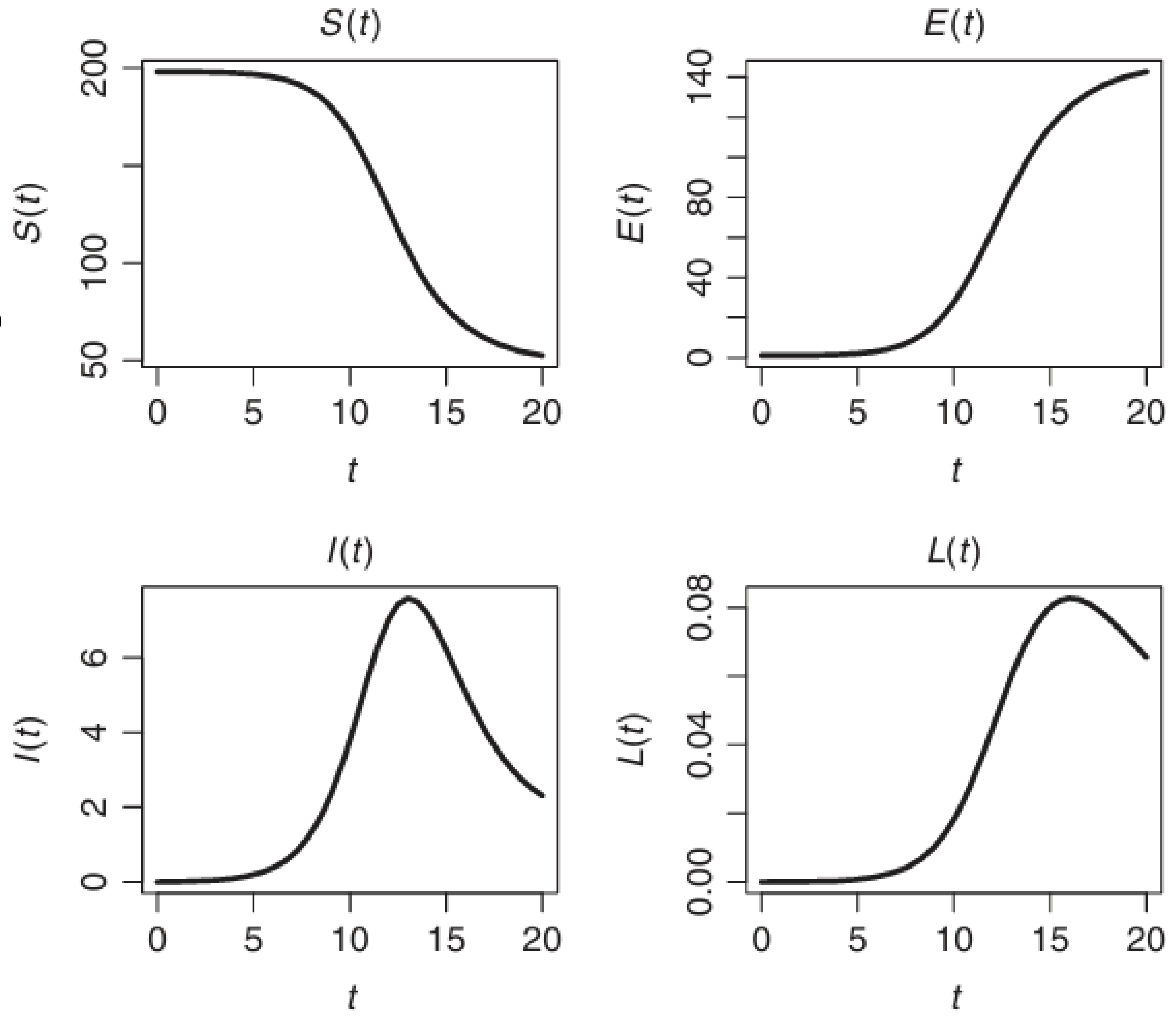
LSODES detected stiffness and switched to **BDF** when needed

System **shows epidemic behavior:**

- Decreasing S(t), Sharp rise in E(t), Slower, delayed rise in I(t), L(t)

LSODES effort:

- Number of ODE calls = **133**



SECTION 04

METHOD 01

Non-Self-Starting Heun’s Method

NON-SELF-STARTING HEUN’S METHOD

Definition:

- A two-step predictor-corrector method used to numerically solve systems of first-order ordinary differential equations (ODEs).
- Called “non-self-starting” because it requires a special step (typically Euler’s method) to compute the first value before applying the Heun updates.
- Based on an improved version of Euler’s method that reduces local truncation error and increases accuracy.

METHODOLOGY

Predictor step (Euler Estimate): :

- Compute a preliminary estimate of the next state Y_{n+1} using the current slope:

$$[\tilde{Y}_{n+1} = Y_n + h \cdot f(Y_n)]$$

Corrector Step (Slope Averaging):

- Recalculate the slope at the predicted point Y_{n+1}
- Average it with the original slope at Y_n, then compute the corrected next value:

$$[Y_{n+1} = Y_n + \frac{h}{2}\left[f(Y_n) + f(\tilde{Y}_{n+1})\right]]$$

ADVANTAGES

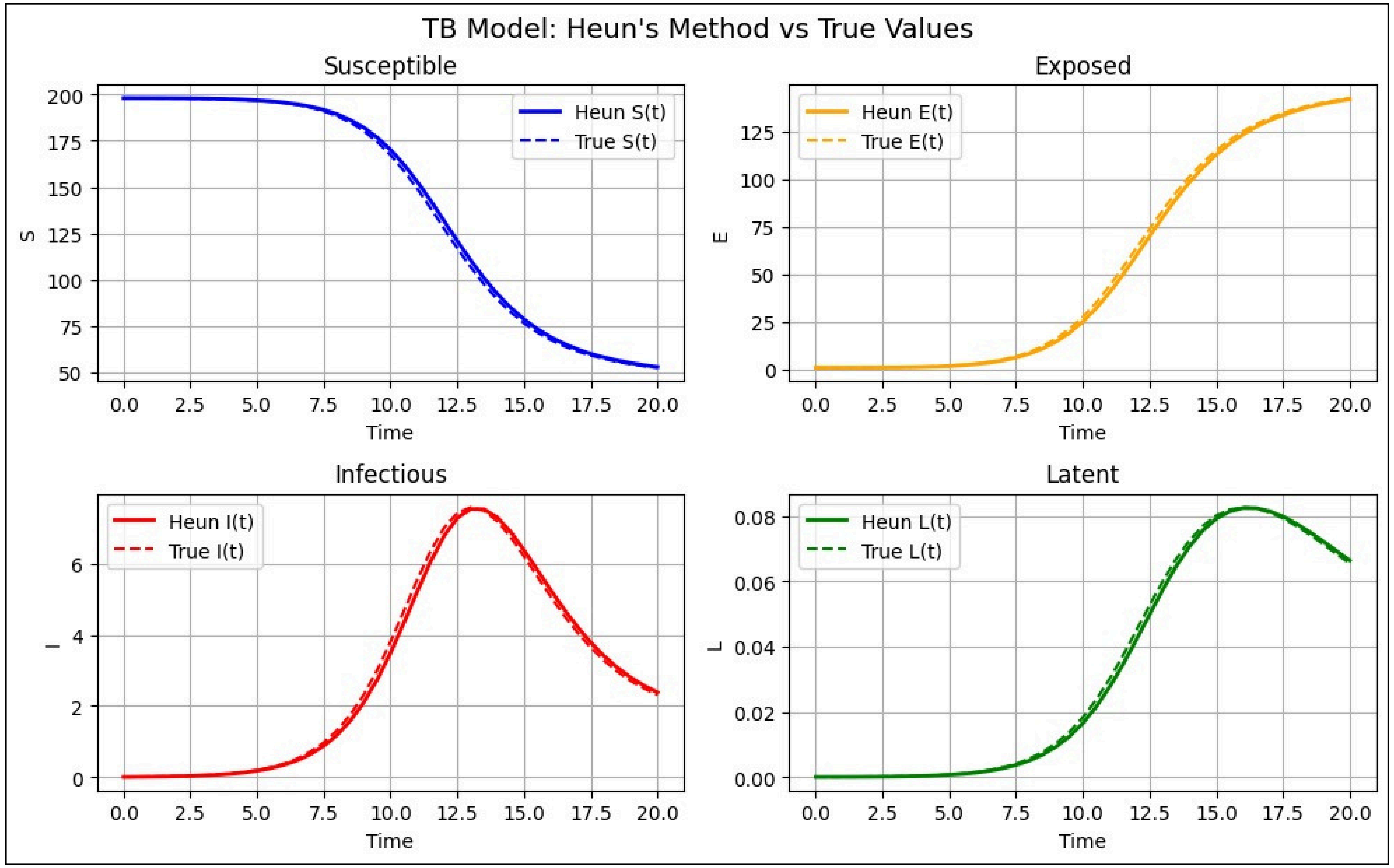
- Accuracy: More accurate than Euler’s method by using slope at both ends of interval.
- Simplicity: Easy to implement and understand; requires only two function evaluations per step.
- Versatile: Works well with single ODEs and systems (e.g., epidemiology, physics, chemistry).
- Stable for moderately stiff problems compared to Euler.

LIMITATIONS

- Cannot be used alone for the first step (requires Euler or another method).
- Not as powerful for highly stiff systems compared to implicit methods.

RESULTS:

Heun’s method performs very well overall, particularly for the Infectious and Latent compartments, where the mean square errors are shown to be extremely low.



Compartment	MSE
S	3.687817
E	3.394948
I	0.022697
L	0.000002

SECTION 04
METHOD 02

Runge-Kutta Fehlberg

RUNGE-KUTTA-FEHLBERG METHOD:

Definition:

- The Runge-Kutta methods are a family of iterative techniques used to solve ordinary differential equations (ODEs) numerically.
- These methods approximate the solution by evaluating the derivative (slope) at multiple points within each time step and then combining these to get an accurate estimate of the next value.
- Among them, the 4th-order Runge-Kutta method (RK4) is the most widely used due to its balance between accuracy and computational cost.

METHODOLOGY:

- It calculates 4 slopes at each step, one at the beginning and two at the midpoint using the previous slopes and one at the end of the interval.
  These slopes represent estimates of the rate of change, and are calculated using a chosen step size h.

- The mid point slopes are given more weight, to better capture the system’s behavior.

- After computing the four slopes, we take a weighted average of them to estimate the next value of the variable. Then apply this to each equation in the TB model (S,E,I,L) to update all values at each time step.

$$k1 = h * f(t_n, y_n)$$

$$k2 = h * f\left(t_n + \frac{h}{2}, y_n + \frac{k1}{2}\right)$$

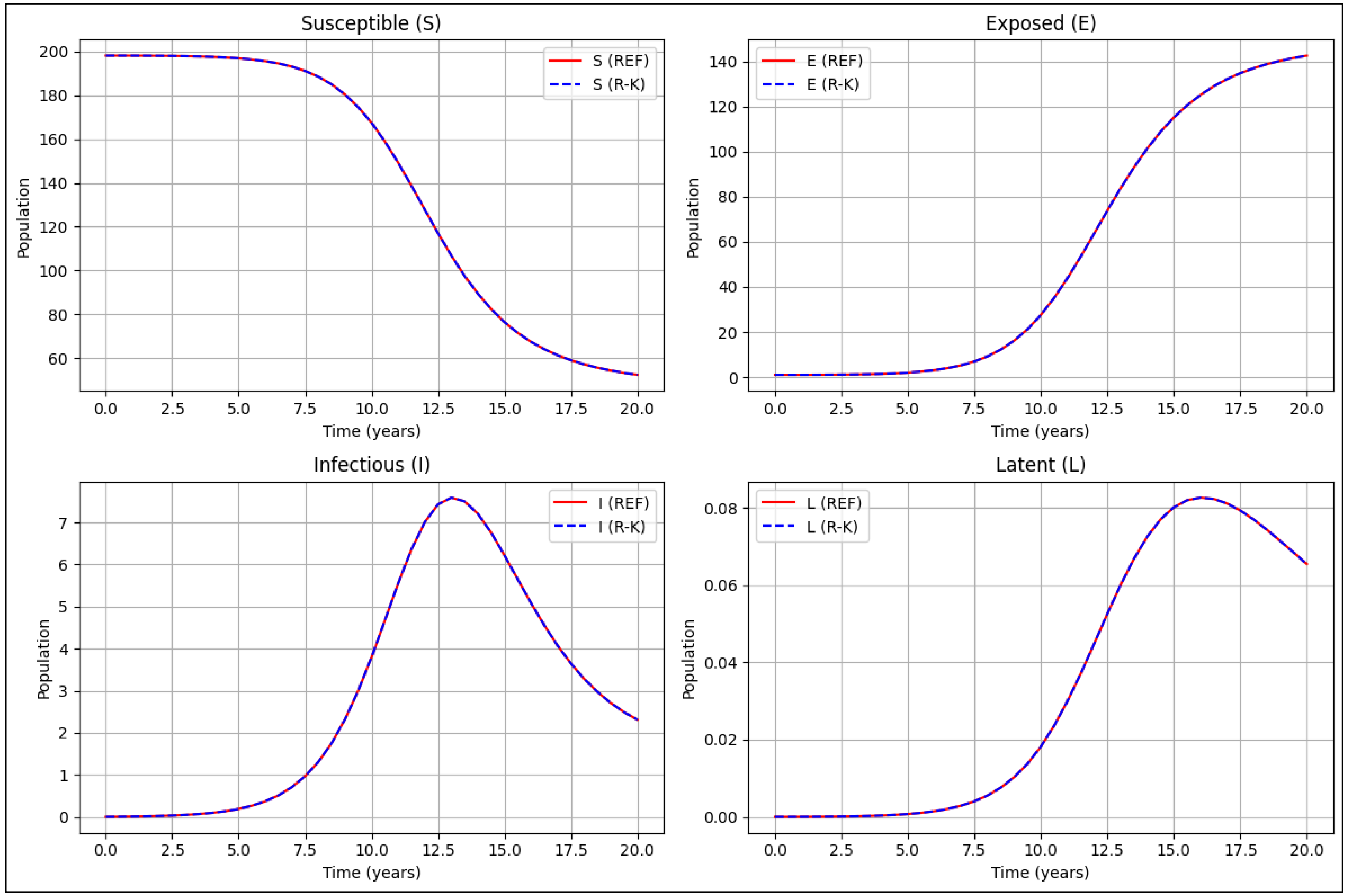
$$k3 = h * f\left(t_n + \frac{h}{2}, y_n + \frac{k2}{2}\right)$$

$$K4 = h * f(t_n + h, y_n + k3)$$

$$y_{\{n+1\}} = y_n + \left(\frac{1}{6}\right) * (k1 + 2k2 + 2k3 + k4)$$

RESULTS :

- RK4 offers high accuracy because it's a fourth-order method, the error decreases rapidly with smaller step sizes, and even with moderate step sizes, it remains very precise with extremely low (almost zero) mean squared errrors.



Compartment	MSE
S	0.000128928
E	0.000119767
I	$8.57439\text{x}10^{-8}$
L	$6.70993\text{x}10^{-11}$

SECTION 04

METHOD 03

Adaptive Quadrature

ADAPTIVE QUADRATURE

Definition:

- Numerical integration methods that dynamically adjust step size based on estimated local error to maintain a specified level of accuracy.
- It doesn’t represent a certain method. It represents an integration technique.
- It can be used to solve ODEs by transforming them into a form that contains an integral. That integral can be approximated each step to get the solution.

Key Properties:

- Accuracy control.
- Computational effeciency

METHODOLOGY

- Converting the ODEs to Integral form

$$[S(t) = S(0) + \int_0^t [\Lambda - \beta S(\tau)(I(\tau) + \delta L(\tau)) - \mu S(\tau)]\, d\tau]$$

$$[E(t) = E(0) + \int_0^t [\beta(1-p)S(\tau)(I(\tau) + \delta L(\tau)) + r_2 I(\tau) - (\mu + k(1-r_1))E(\tau)]\, d\tau]$$

$$[I(t) = I(0) + \int_0^t [\beta p S(\tau)(I(\tau) + \delta L(\tau)) + k(1-r_1)E(\tau) + \gamma L(\tau) - (\mu + d_1 + \phi(1-r_2) + r_2)I(\tau)]\, d\tau]$$

$$[L(t) = L(0) + \int_0^t [\phi(1-r_2)I(\tau) - (\mu + d_2 + \gamma)L(\tau)]\, d\tau]$$

METHODOLOGY

- An ODE solver generally consists of two numerical methods with different orders of accuracy to estimate the integral, a local truncation error calculation method, and a step adjustment method that uses the calculated error.

- Our solver is based on the Dormand-Prince from the Runge-Kutta family. It uses RK methods of orders 4 and 5. However, they aren't calculated separately. They mostly use the same function evaluations.

- The local truncation error is simply the difference between the two estimated values. Finally, the step size can be adjusted using that error, the desired tolerance, and a safety factor to avoid oscillations.

$$y(t_1) = y(t_0) + h \sum_{i+1}^{s} b_i k_i$$

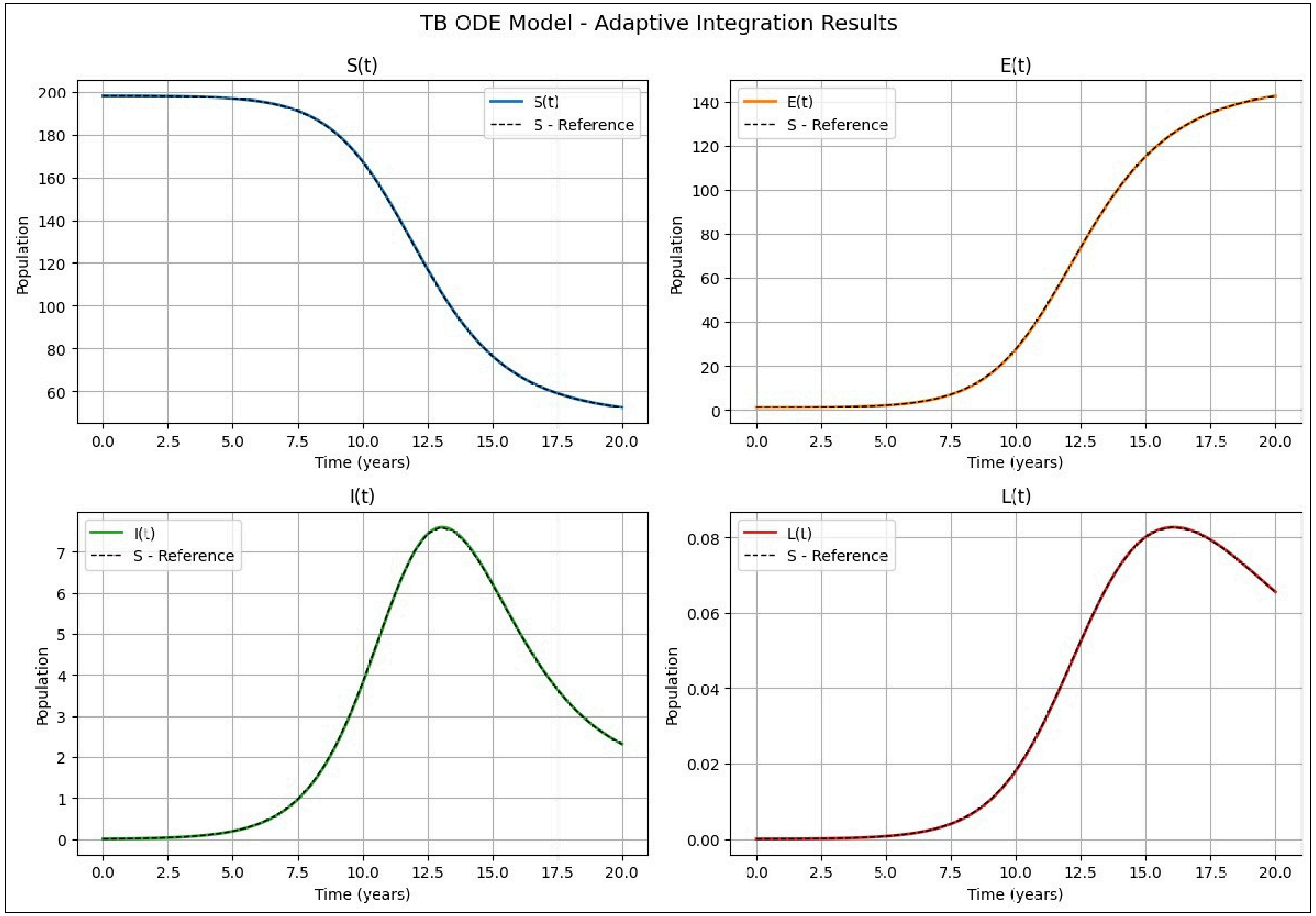
$$k_i = f(t_n + c_s h, y_n + \sum_{j=1}^{s-1} a_{sj} k_j)$$

$$LTE \approx ||y_{n+1}^{(5)} - y_{n+1}^{(4)}||$$

$$h_{new} = h_{old} . F . (\frac{T}{LTE})^{\frac{1}{(p+1)}}$$

RESULTS:

Dormand-Prince is the fastest method due to its reuse of computations. The relatively high accuracy is achieved by its dynamic step adjustment.



Compartment	MSE
S	0.016306
E	0.015111
I	0.000112
L	0.000000

SECTION 05

Physics-Informed Neural Networks (PINNs)

PINNs Method

- A deep learning Approach for solving Differential Equations
  PINNs are a class of neural networks that incorporate physical laws (usually in the form of differential equations) into the training process..

- It originates from the principle of universal approximation of neural networks: given sufficient data and the proper configuration, neural networks can learn to replicate the relationship between virtually any set of inputs and outputs.

- It was first proposed in 2017 by Raissi et al. to solve partial differential equations (PDEs) with a neural network [1]

- Mean Square Error (MSE) was selected as a criterion for calculating the loss between a set of predicted points and a set of actual true points

[1] Rom, M. (2022). Physics-informed neural networks for the Reynolds equation with cavitation modeling. Tribology International, 179, 108141. https://doi.org/10.1016/j.triboint.2022.108141

WHY PINNs Over a Numerical Solution ?

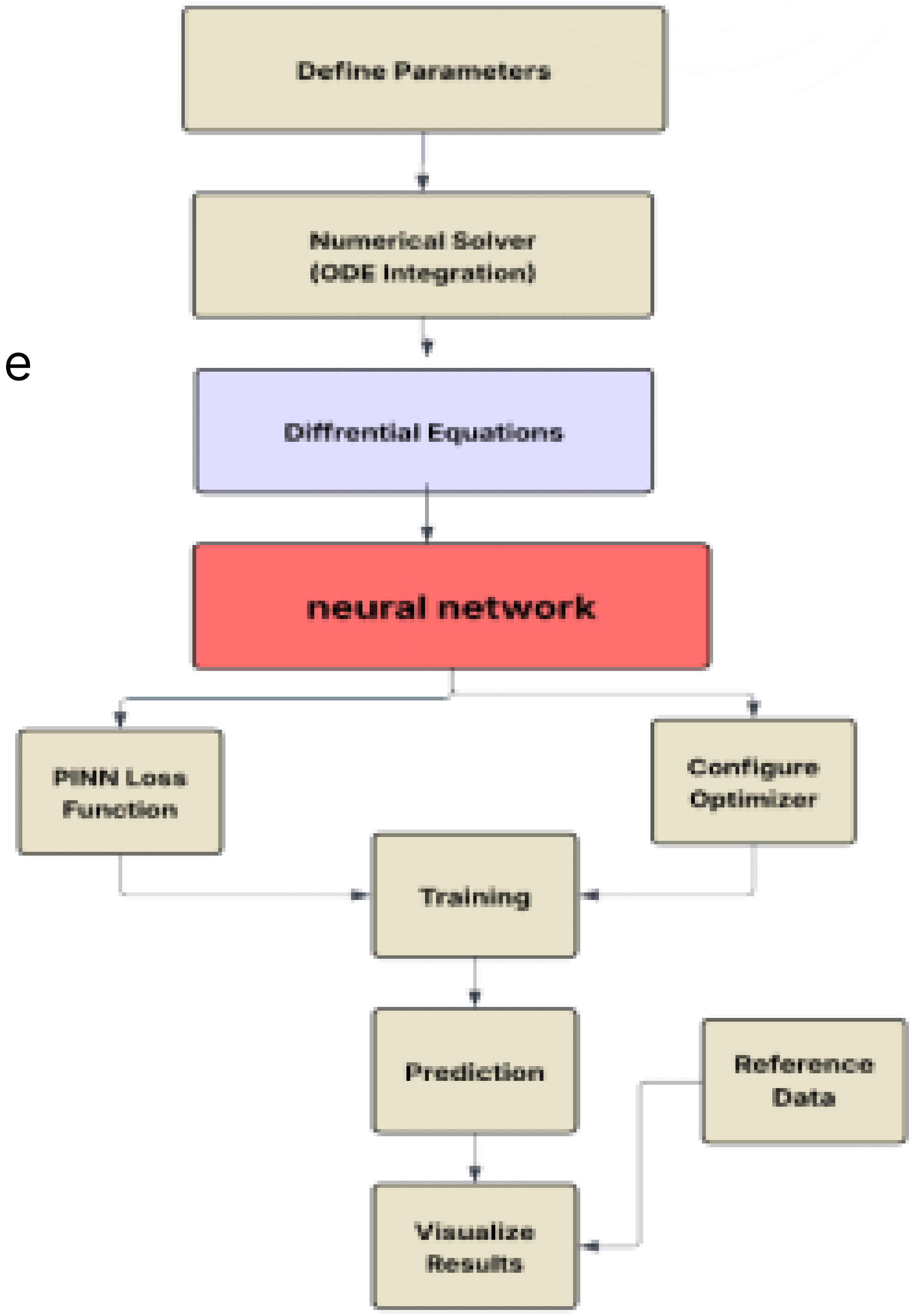
- We can evaluate the function at any given time t after training, unlike numerical methods, due to the continuity of the function modelled by the PINN

- The function was proven to have a high extrapolation accuracy for data points further than those used in training

- When the training is done, the model can be saved and evaluated later in a constant time complexity at any point t while yielding the same accuracy
  In terms of speed:
  - Gauss-Legendre quadrature took 0.30 seconds,
  - PINN took only 0.0038 seconds — 7900% faster.

Approach

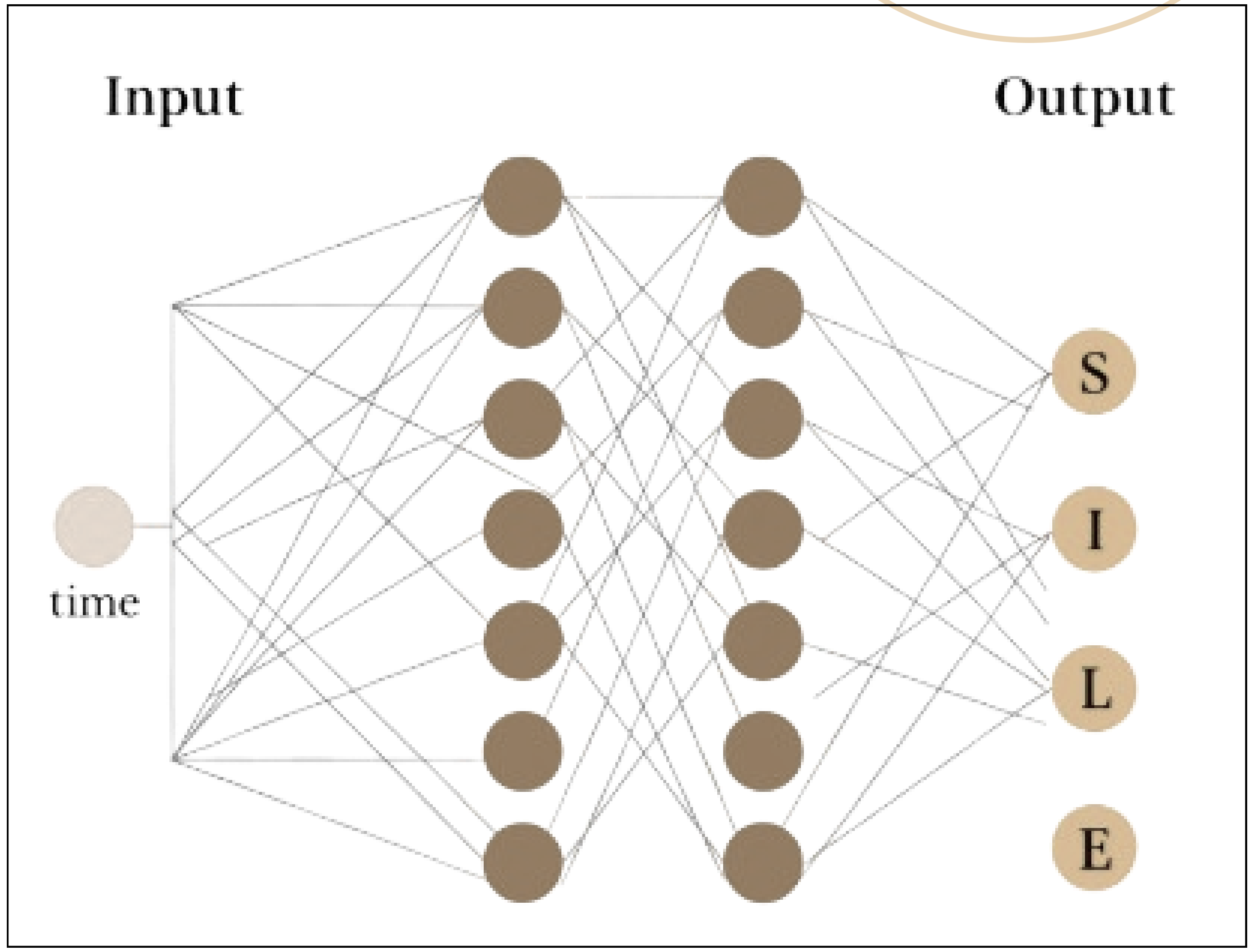
We start by defining the model parameters and solving
the ODEs numerically to generate reference data.
A neural network is then trained to predict the TB compartments over time
by minimizing a loss based on
the differential equations, initial conditions, and known data points.
Once trained, the PINN provides accurate, continuous predictions
which we compare and visualize against traditional methods.



NN Class

- Initialize the NN input and output sizes and defines the architecture using fully connected layers.

- Activation Function : uses *tanh* by default, which helps in normalizing the outputs.

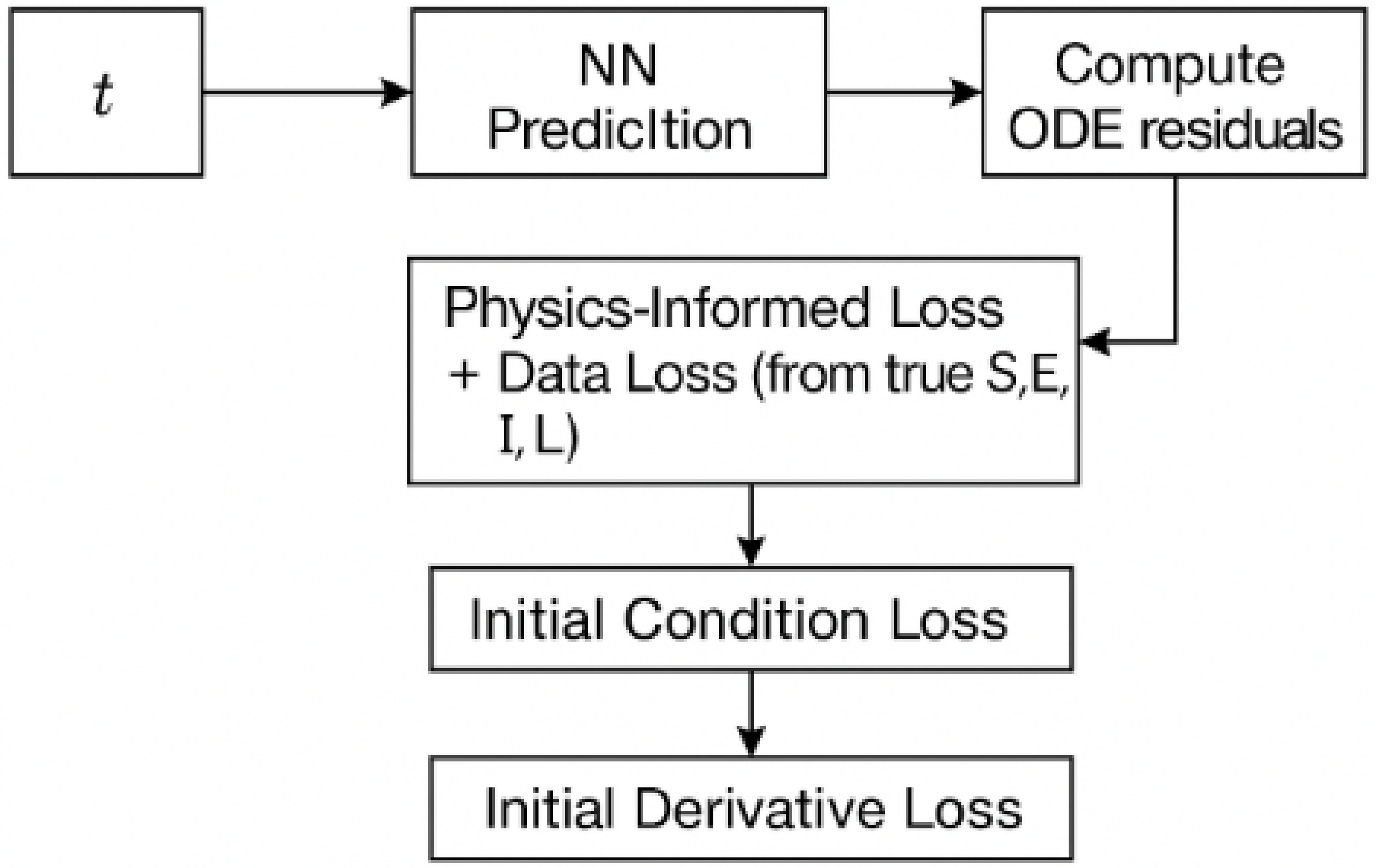


How Neural Network Learns

The neural network learns the structure of the ODE system through the *loss function* defined to train the network.

The **loss function** is composed of 4 components that capture the entirety of the ODE system

1. Physics Loss
2. Initial Condition Loss
3. Initial Derivatives Loss
4. Data Points Loss



Losses in the PINNs Method

- The physics loss represents how well the model follows the actual ODE system that we attempt to solve.

- The outputs S, E, I, and L were estimated. The derivative (gradient) at each of those outputs at the corresponding time was also estimated. The values were then used to compute the difference between the L.H.S and the R.H.S in the ODE equations shown on the left. The difference is the physics residual, also known as the *physics loss*.

- The data points loss is the difference between the model’s predictions at various time points with known value and the true values at those known points.

- The initial condition loss is the difference between the actual value of each of the outputs at t=0 and the model’s estimated values at t=0.

Losses in the PINNs Method

- The initial derivative loss is the difference between the value of the first derivative (from the ODE equations) at the initial conditions and the model's prediction for those derivatives.

- To find those data points, an integration solution (Gauss-Legendre) was utilized to find the values of the ODE system at **40** equally spaced points from **t=0** to **t=20**

The physics loss was computed as:

$$L_{physics} = 0.1 * S_{ode\ loss} + 10 * L_{ode\ loss} + 10 * I_{ode\ loss} + 1000 * L_{ode\ loss}$$

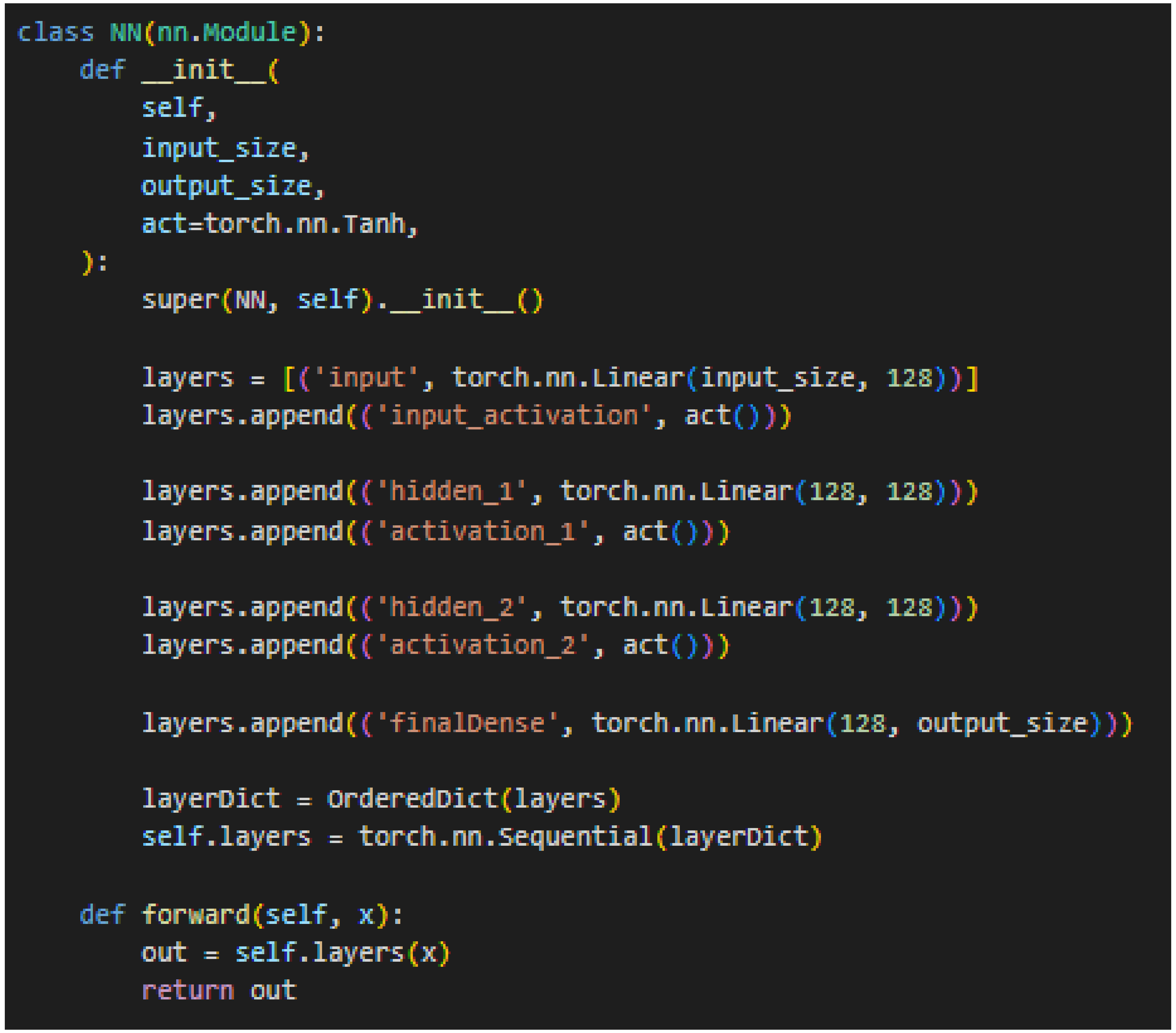
where each ode loss is the physics residual of a particular ODE in the ODE system.

Losses in the PINNs Method

- $L = \lambda_{physics} * L_{phyiscs} + \lambda_{IC} * L_{IC} + \lambda_{IC-derivative} * L_{IC-derivative} + \lambda_{data} * L_{data}$
- The parameters $\lambda_{physics}$ and $\lambda_{initial}$ are initialized to be 1 and 100, respectively.

Epoch	$\lambda_{physics}$	λ_{data}
0 < epoch< 1000	*1*	*1000*
1000 < epoch < 3000	*10*	*1000*
3000 < epoch < 7000	*100*	*1000*
7000 ≤ epoch < 12000	*100*	*100*
12000 ≤ epoch < 20000	*100*	*10*
epoch ≥ 20000	*100*	*1*

The Model’s Architecture



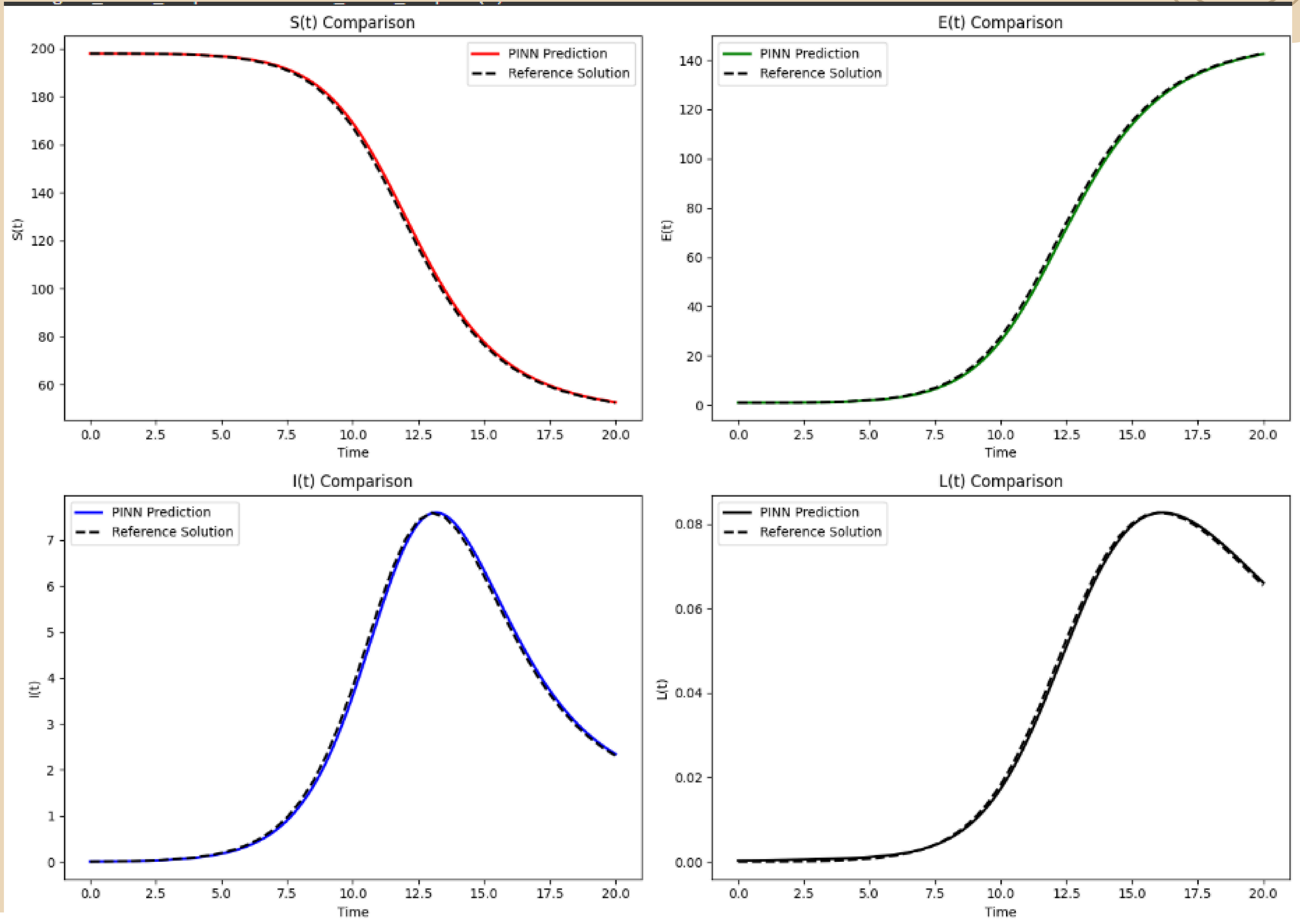
```
class NN(nn.Module):
    def __init__(
        self,
        input_size,
        output_size,
        act=torch.nn.Tanh,
    ):
        super(NN, self).__init__()

        layers = [('input', torch.nn.Linear(input_size, 128))]
        layers.append(('input_activation', act()))

        layers.append(('hidden_1', torch.nn.Linear(128, 128)))
        layers.append(('activation_1', act()))

        layers.append(('hidden_2', torch.nn.Linear(128, 128)))
        layers.append(('activation_2', act()))

        layers.append(('finalDense', torch.nn.Linear(128, output_size)))

        layerDict = OrderedDict(layers)
        self.layers = torch.nn.Sequential(layerDict)

    def forward(self, x):
        out = self.layers(x)
        return out
```

The Model’s Parameters

- The main optimizer is adam with a **learning rate** of **0.001** and an exponential scheduler with a **gamma** of **0.99**
- The model runs for **30000 epochs**, after which it’s optimized again by LBFGS optimizer L-BFGS, a quasi-Newton optimizer, a well-known second-order optimizer.it is used to fine-tune the network parameters and achieve very low training error once the solution was already close to the optimum. The optimizer ran until convergence
- The parameters of the LBFGS optimizer as the following:

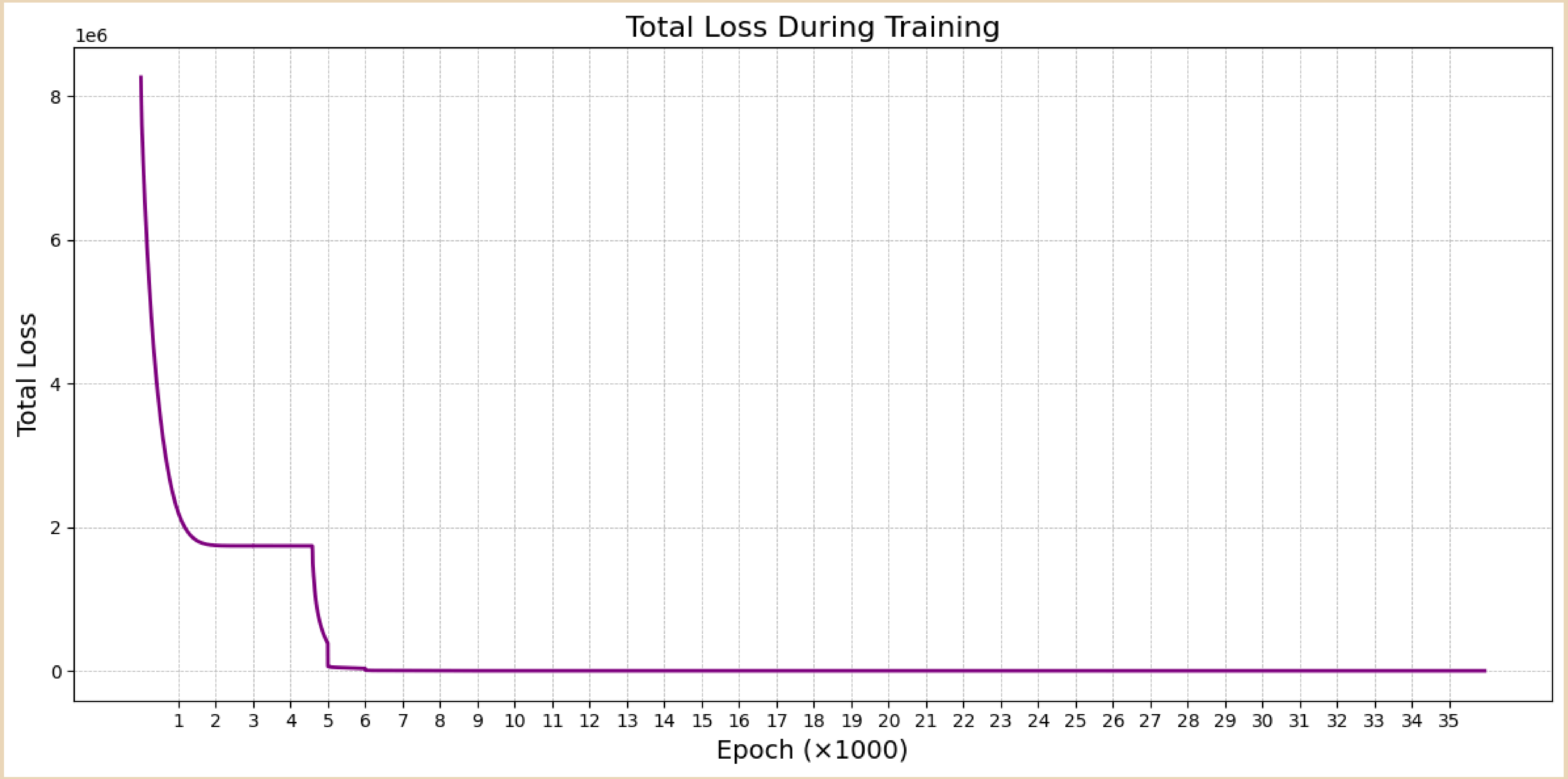
```
lr=1.0,
max_iter=100000,
max_eval=100000,
history_size=50,
tolerance_grad=1e-7,
tolerance_change=1.0 * np.finfo(float).eps
line_search_fn="strong_wolfe"
```

Models' Results



Pinn Model

Model Convergence ?

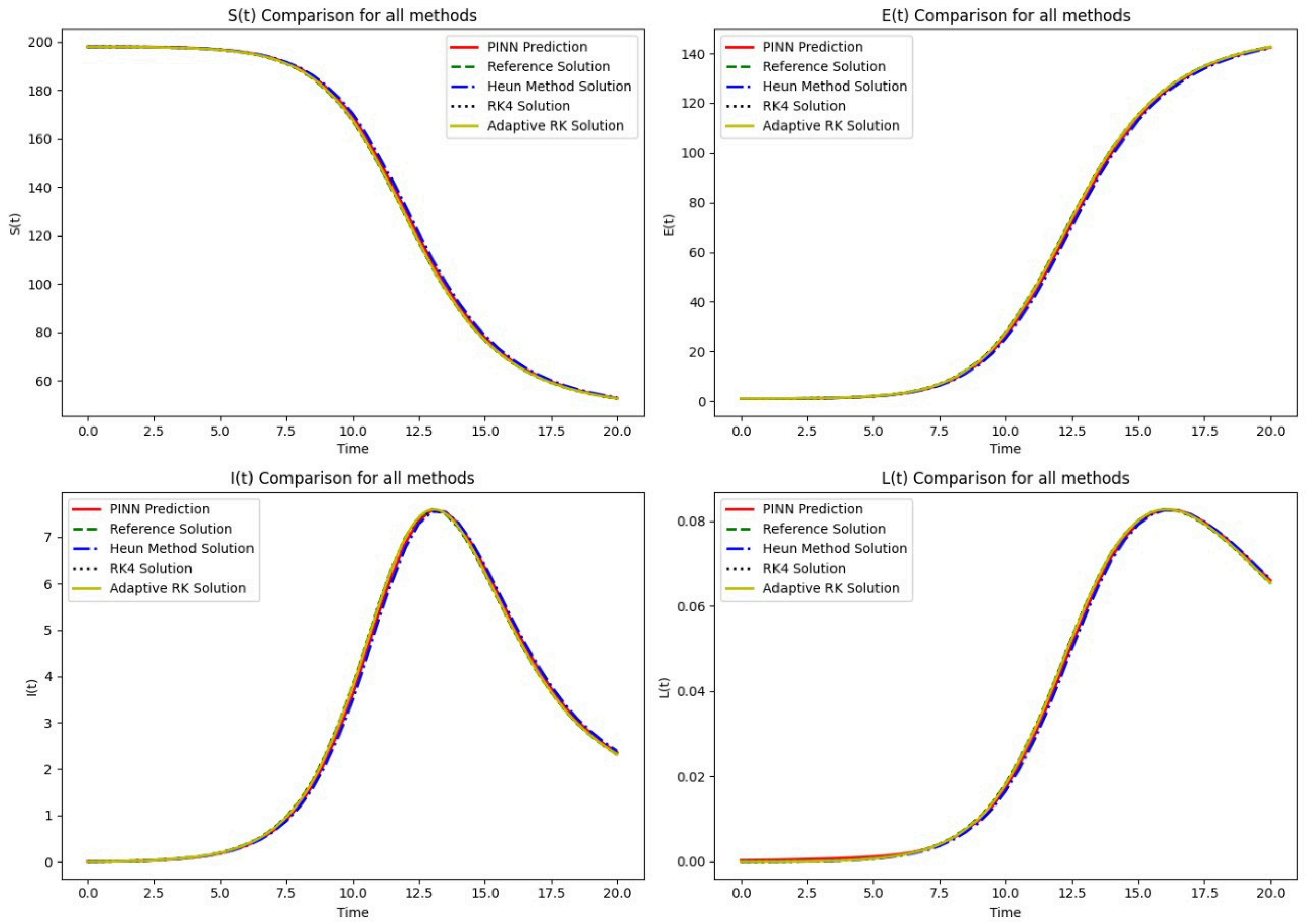


Epochs VS Total Loss

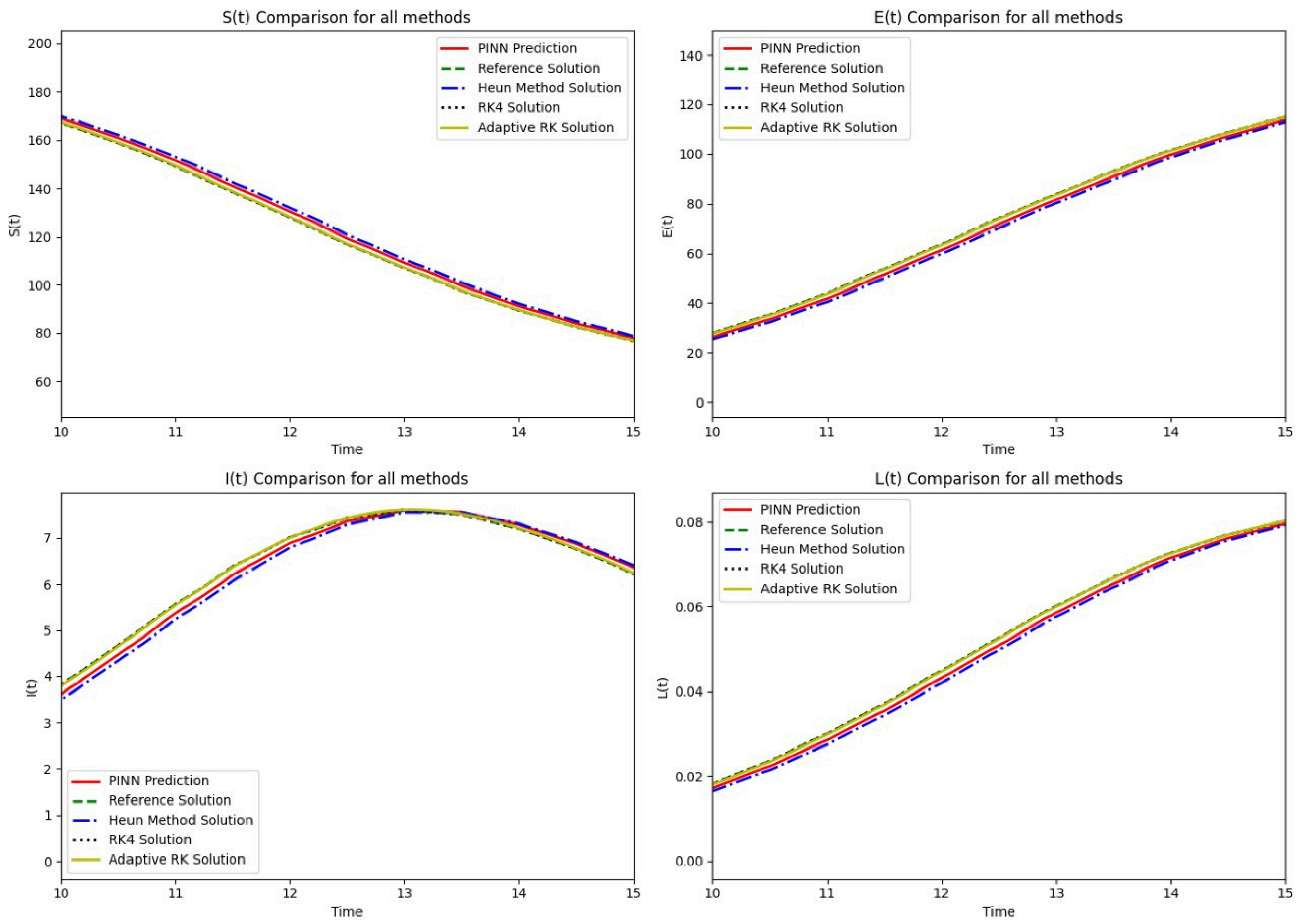
SECTION 06

Comparative Analysis

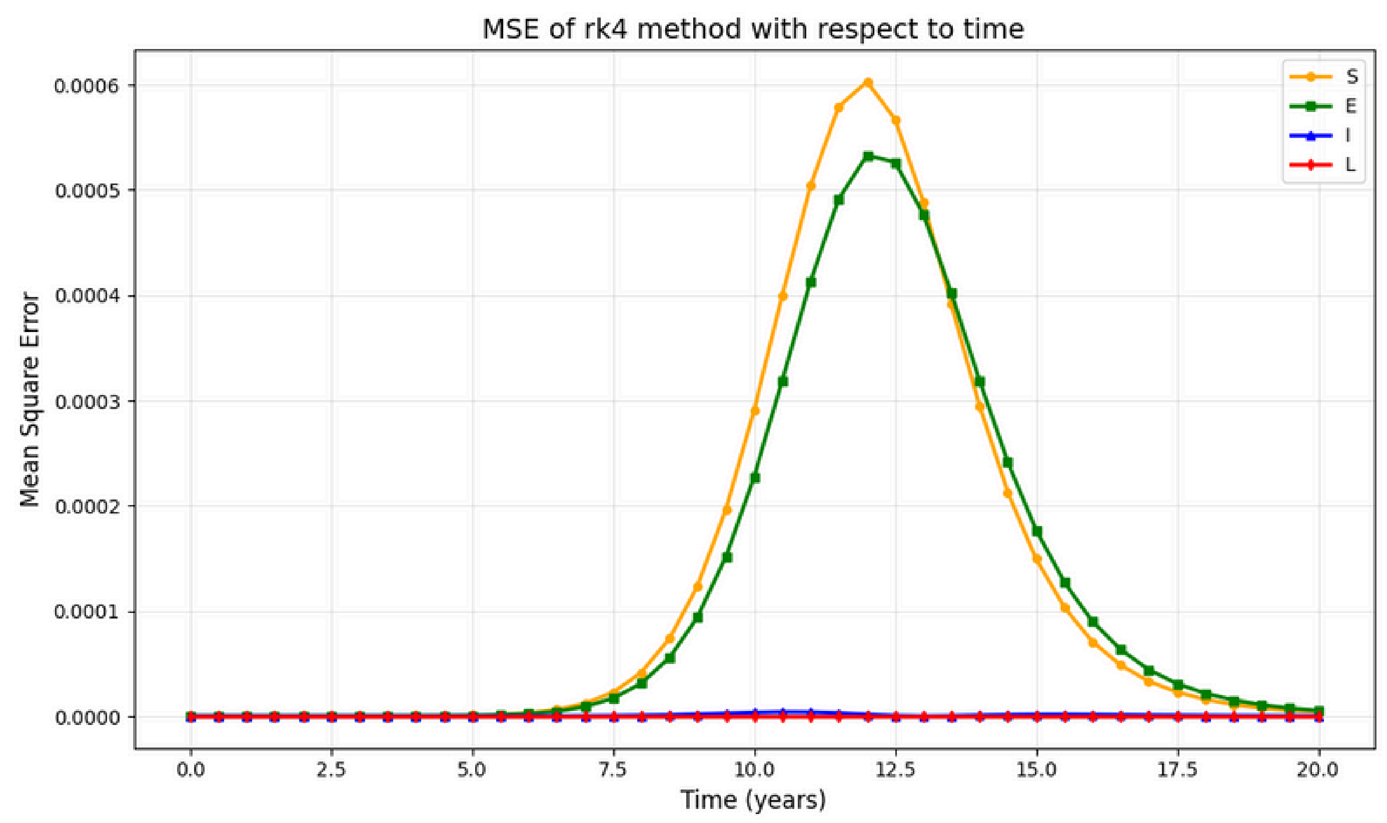
Graphical Comparison of All Methods:

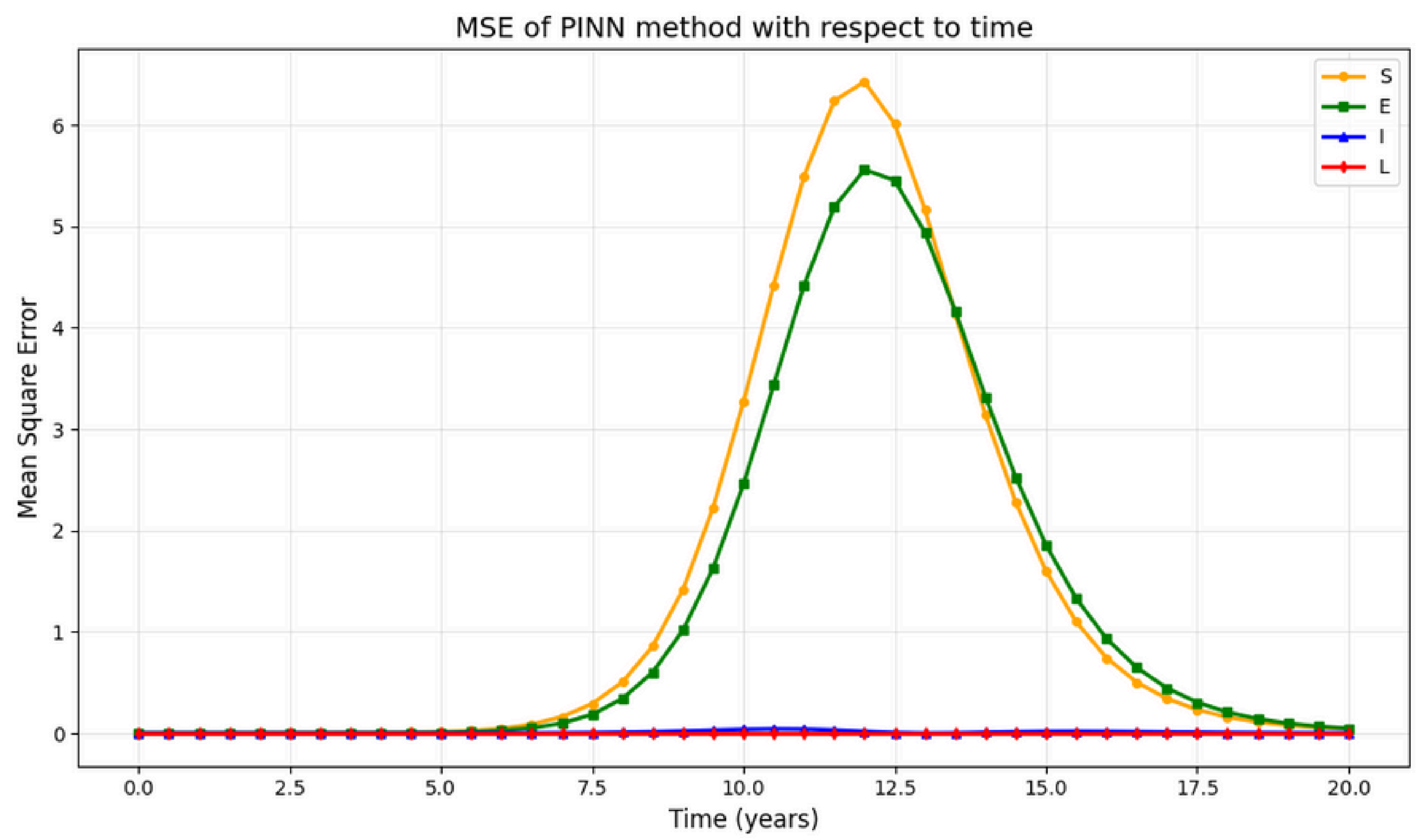


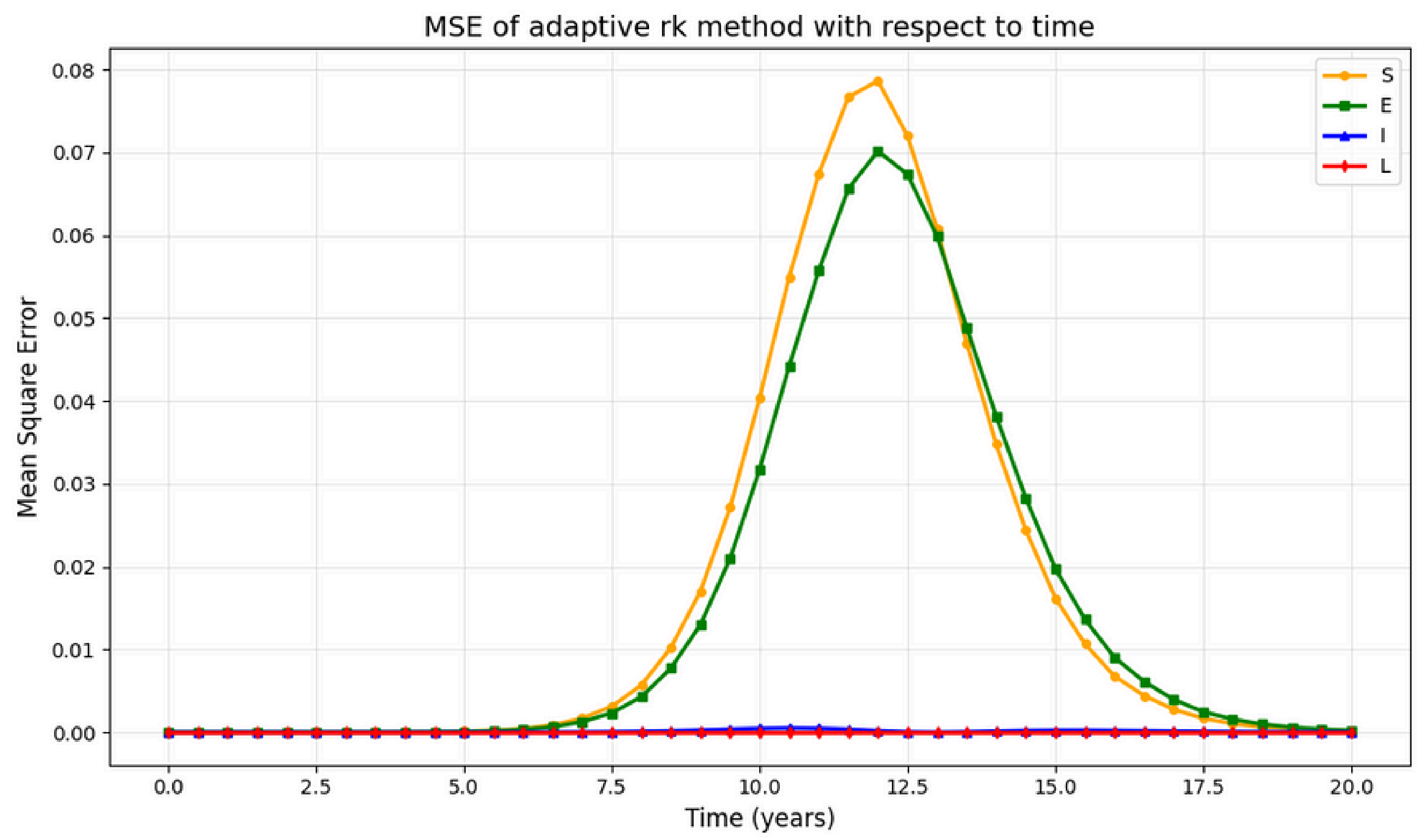
Enlarged Comparison of All Methods:

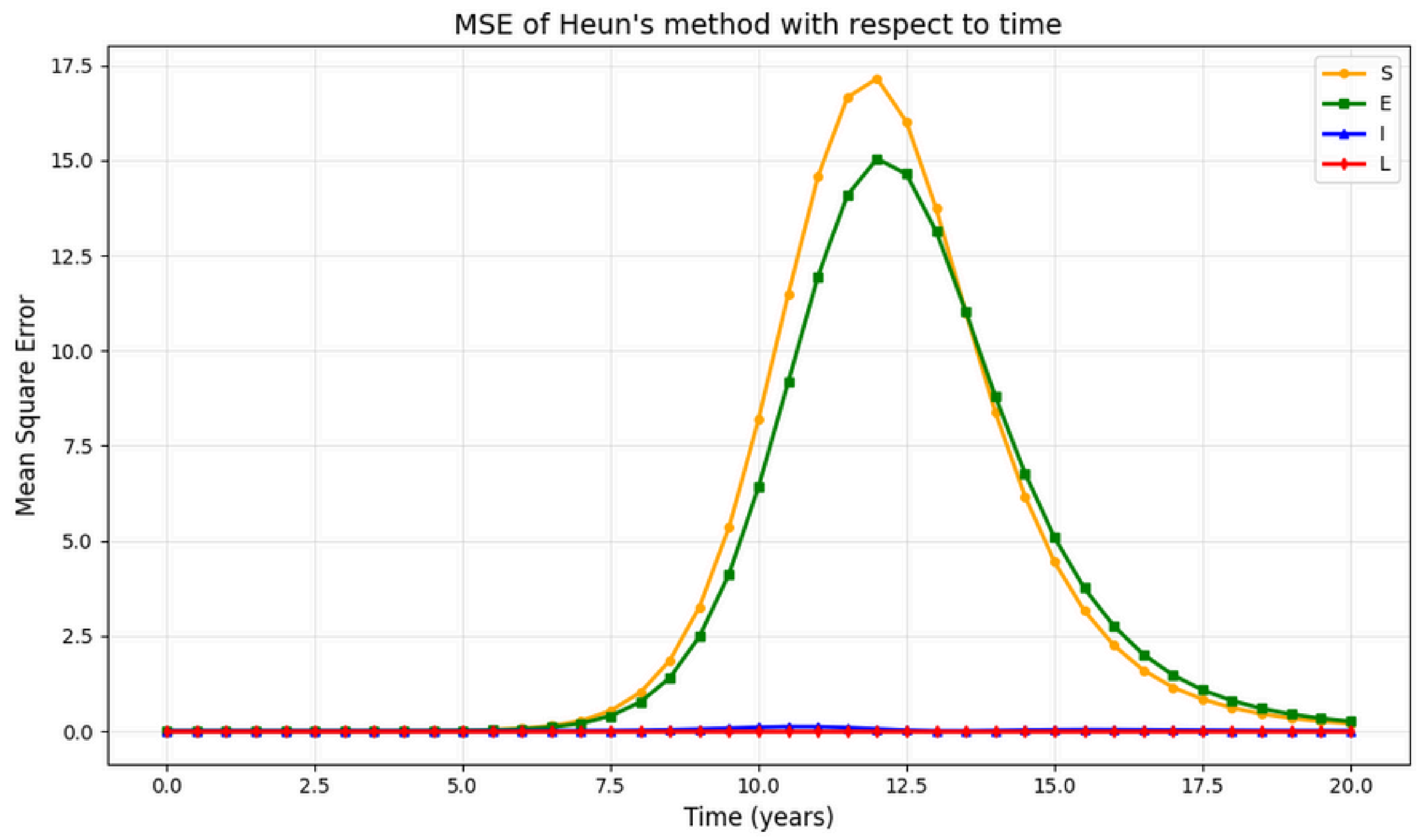


Graphical Comparison of MSE:









Results

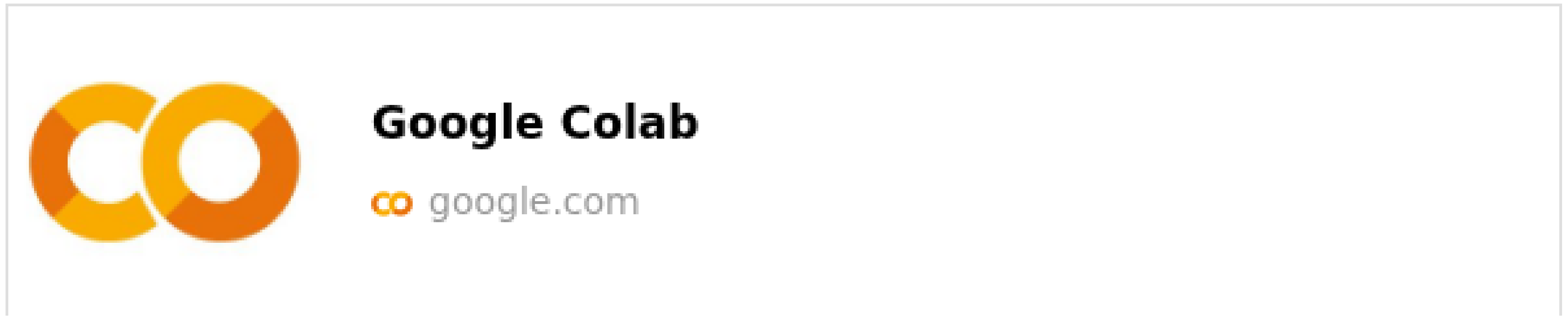
Method	Most Accurate	Fastest Execution	Extrapolation Support
Heun	×	Fast	×
RK4	✓ Very Accurate	Moderate	×
Adaptive RK	✓ Accurate	Fastest	×
PINN	✓ Accurate	Fast (post-training)	✓

Average Mean Squared Error (MSE) per Method:

Compartment	Heun	RK4	Adaptive RK	PINN
S	3.6878	0.000128928	0.016306	1.37
E	3.3949	0.000119767	0.015111	1.27
I	0.0227	8.57439x10-8	0.000112	0.009
L	0.000002	6.70993x10-11	0	0.000001

CODES:

Click on the link below to find the codes and plots of all methods and our AI model:



Google Colab

google.com

SECTION 07

Conclusion

Conclusion

- We modeled TB dynamics using a SEIL system of nonlinear ODEs and solved it with several numerical and AI-based methods.

- Classical Methods (Heun, RK4, Adaptive Quadrature):
  1. Provided high accuracy, especially RK4 and Adaptive Quadrature.
  2. Heun was simple but less effective for stiff problems.

- LSODES:
  - Handled stiffness automatically and served as a strong benchmark.

- PINNs (Physics-Informed Neural Networks):
  - Required training but delivered fast, flexible, and highly generalizable results.
  - Showed potential for data-limited scenarios and extrapolation.
  -
- Comparative Insight:
  - Each method has trade-offs in speed, accuracy, and scalability.
  - Classical solvers are reliable; PINNs are promising for future AI-driven modeling.

SECTION 08

Future Improvements

Future Improvements

1. PINN models that can solve this ODE without data points should be researched, a good start might be taking the log scale of S.
    - Reduces large value range in S
    - Makes training more stable and smoother
    - Helps the network focus on small changes

2. The model's architecture can be further optimized for extrapolation by tweaking the parameters contributing to the loss values of L.
    - L has small values and is hard to learn
    - Increasing its loss weight improves accuracy
    - Forces model to better capture latent dynamics

Any Questions?

Thank you!