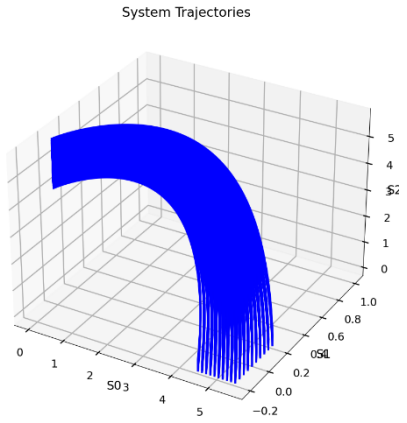
Training Neural Networks on the data from: "On the parameter combinations…"

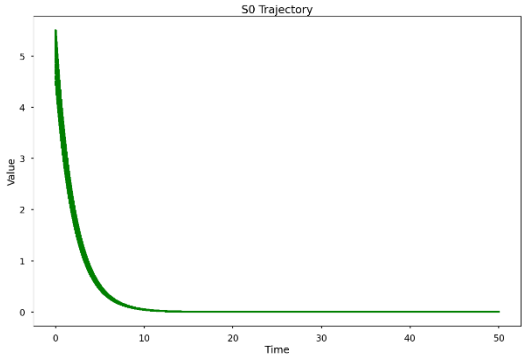
I examined the reduced-MSP model from the above paper and gathered data from Runge-Kutta integrations of the equations, first varying the starting values of the system and then varying the parameters that are incorporated into the equations of the systems. Then I trained two types of Neural Networks based on these data and evaluated their accuracy.

Experiment 1: Varying the starting values of the system

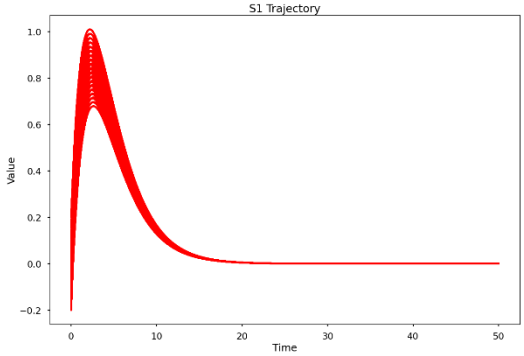
- First, I imported the necessary libraries, determined the parameters of the system based on the ones the paper uses ($k_{f,1} = 0.71$, $k_{f,2} = 0.97$, $k_{r,1} = 19$, $k_{r,2} = 7000$, $k_{cat,1} = 6700$, $k_{cat,2} = 10000$) made the function for the derivatives of the system, the functions for the Runge-Kutta and Euler integration maps and the one that computes the trajectory using these.
- Next, I took a sample of 10 values for each starting value, with deviation from the recommended from the paper values (S0=5, S1=S2=0) in the range [-0.5, 0.5] for S0, and [-0.2, 0.2] for S1 and S2, and plotted the trajectories (from Runge-Kutta) using these starting values. The relevant plots are presented below:



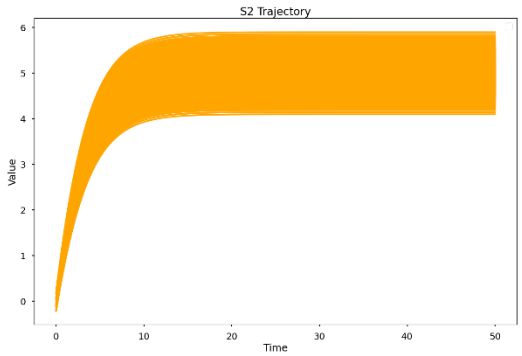
Set of Trajectories in 3D



Set of S0 Trajectories

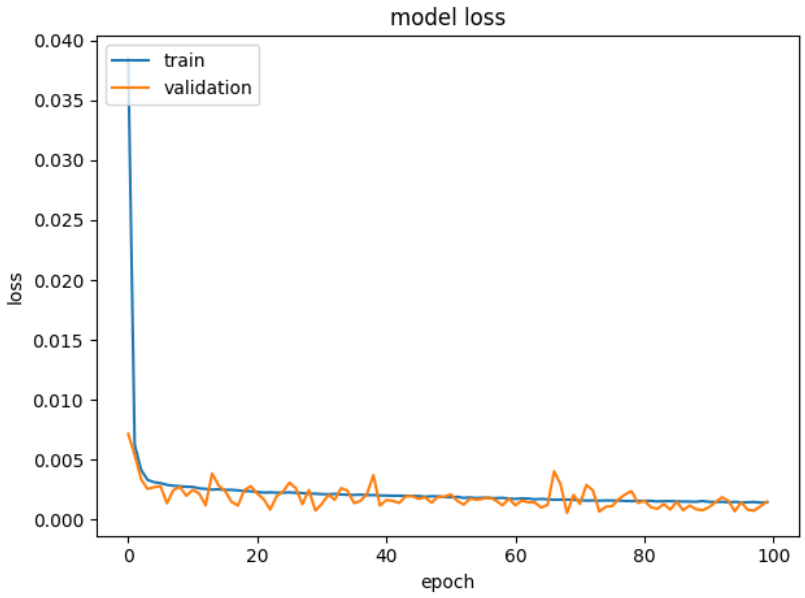


Set of S1 Trajectories



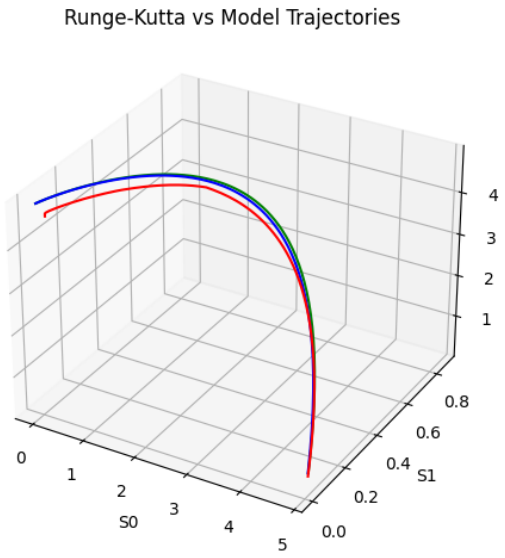
Set of S2 Trajectories

- From these trajectories I also made the training and test data, taking 80% of the 500000 points (500 steps with 10 choices for each starting value) as training and 20% as testing. 20% of the testing points are used as validation dataset for the training.
- Then, I trained a Neural Network to learn the time-1 map of the system, and consists of the input and output layers (input is a point in the S0, S1, S2 3D space and output is the point after time dt) and 2 hidden fully connected layers with 15 nodes each (after a layer that normalizes the input values to the same magnitude)
- The results of the training are shown below:

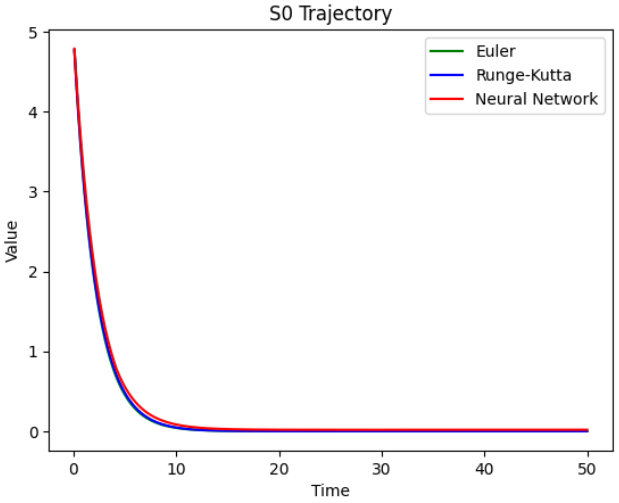



```
Evaluate on training data
3125/3125 [==============================] - 4s 1ms/step - loss: 0.0017
training loss: 0.001657055807299912
Evaluate on test data
782/782 [==============================] - 1s 1ms/step - loss: 0.0016
test loss: 0.0016471188282594085
```

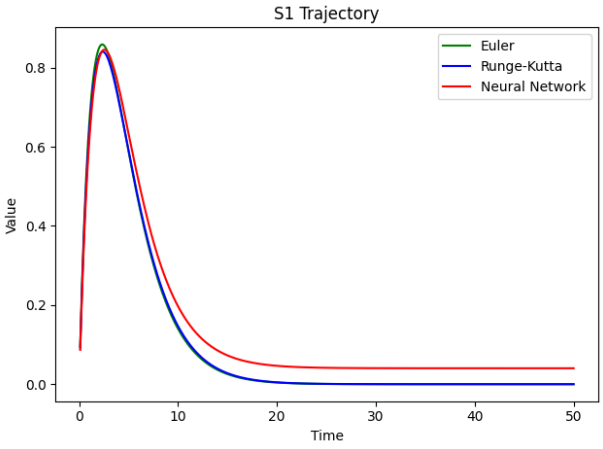
- Then, I plotted the trajectories of the Runge-Kutta method, the Euler method, and the ones predicted from the time-1 map model, to evaluate its accuracy. Green is the Euler trajectory, blue the Runge-Kutta, and red the Neural Network (the Euler method trajectory isn't visible as it is too close to the Runge-Kutta one):



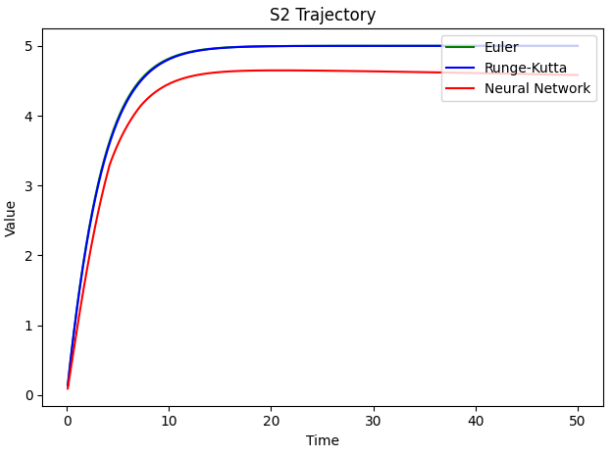
Trajectories in 3D



S0 Trajectories

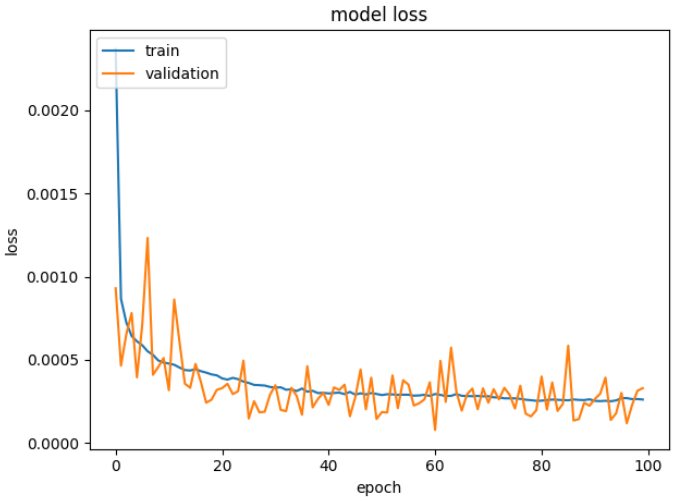


S1 Trajectories



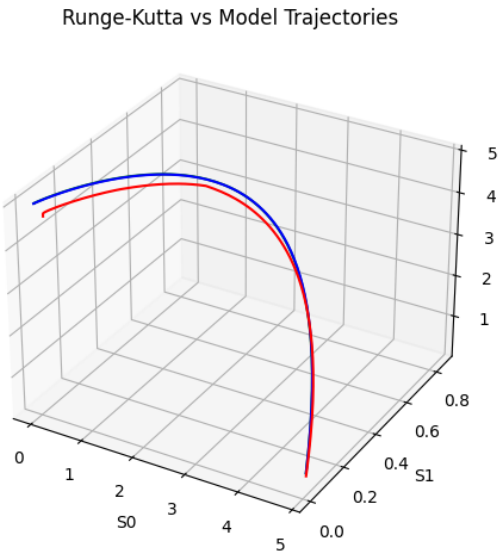
S2 Trajectories

- Then I trained a model that predicts the derivatives of the values at a certain point in the S0, S1, S2 space (the RHS of the differential equation of the system). Its structure is the same as the previous model, but the output contains the derivatives instead of the values of the map. The training data are also based on the trajectories plotted before. The training results are shown below:

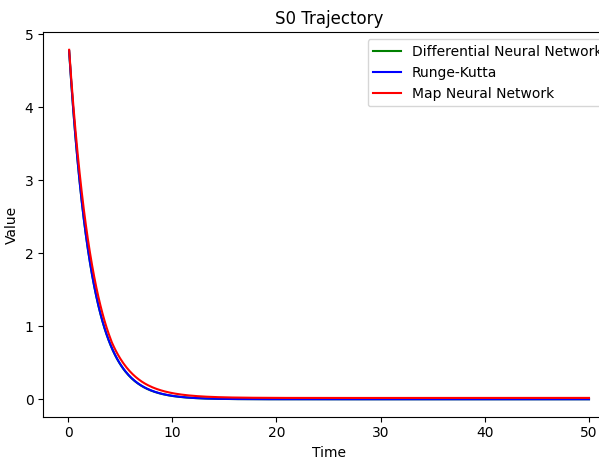



```
Evaluate on training data
3125/3125 [==============================] - 4s 1ms/step - loss: 3.2989e-04
training loss: 0.0003298925294075161
Evaluate on test data
782/782 [==============================] - 1s 1ms/step - loss: 3.2944e-04
test loss: 0.0003294444177299738
```

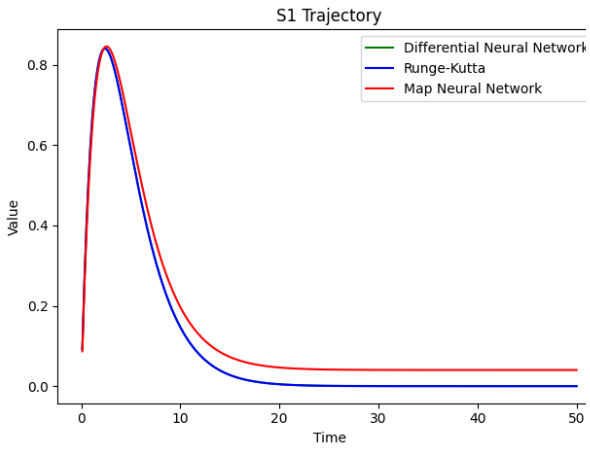
- Then I plotted the trajectories of the two different models, along with the ones with the Runge-Kutta method, in order to evaluate their accuracy. Green is the derivative Neural Network trajectory, blue the Runge-Kutta, and red the time-1 map Neural Network (the trajectory from the derivative model isn't even visible because it is almost the same as the Runge-Kutta one!):



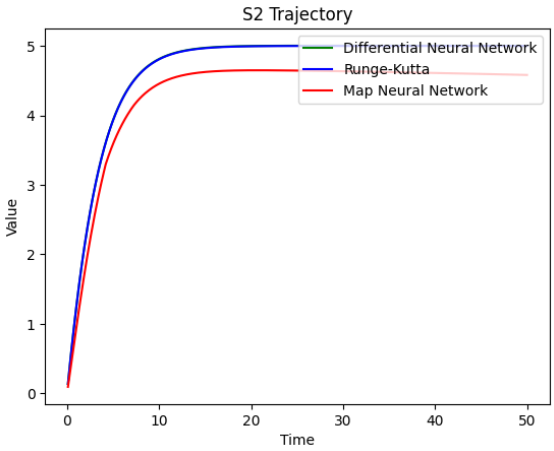
Trajectories in 3D



S0 Trajectories

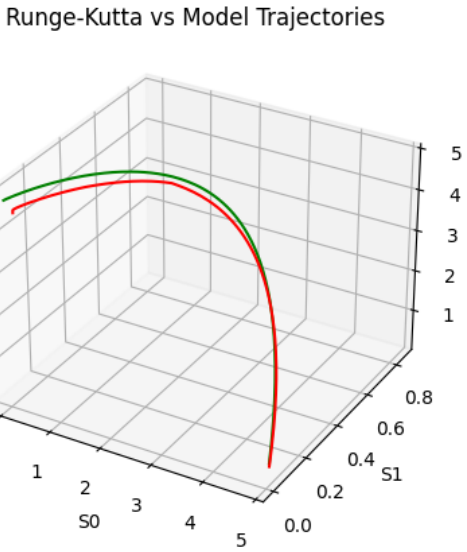


S1 Trajectories

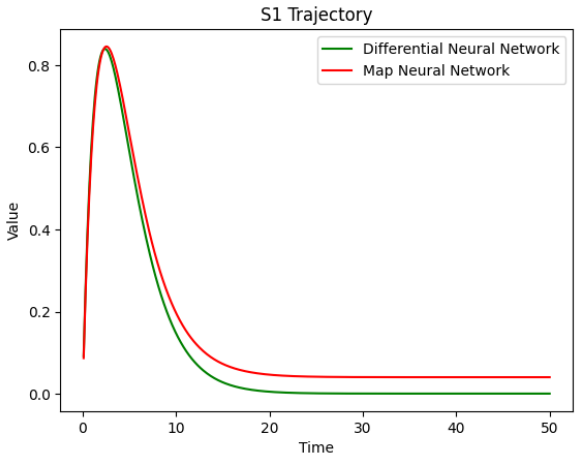


S2 Trajectories

- To show that the derivative NN trajectory exists in the graphs but is the same as the Runge-Kutta one, I compared the trajectory of that model (green) and the one from the time-1 map model (red):



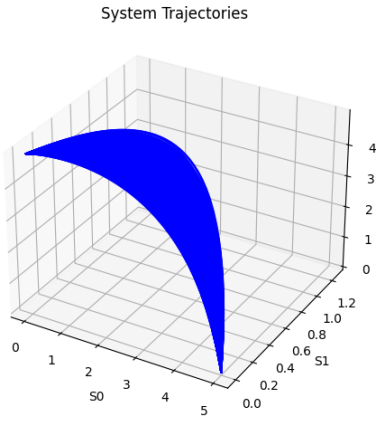
Trajectories in 3D



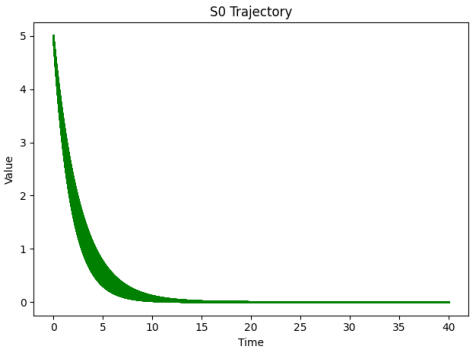
S1 Trajectories

Experiment 2: Varying the parameters of the system

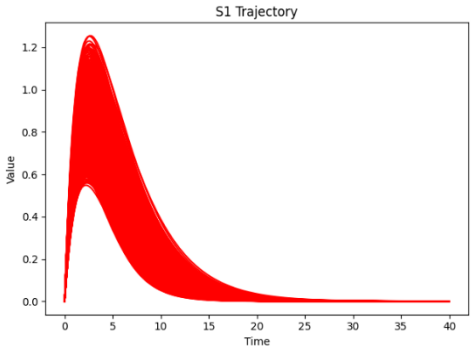
- I did the same preparation as the previous experiment, for the functions necessary for time integration, but this time including the parameters of the system as an input to the functions. I also determined the base parameters of the systems and the starting values as described in the paper.
- Next, I took a sample of 2000 random values for the parameters, with deviation of ±20% from the ones given by the paper, and plotted the trajectories (from Runge-Kutta) using these starting values. The relevant plots are presented below:



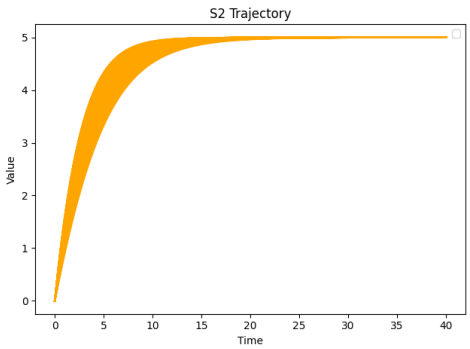
Set of Trajectories in 3D



Set of S0 Trajectories



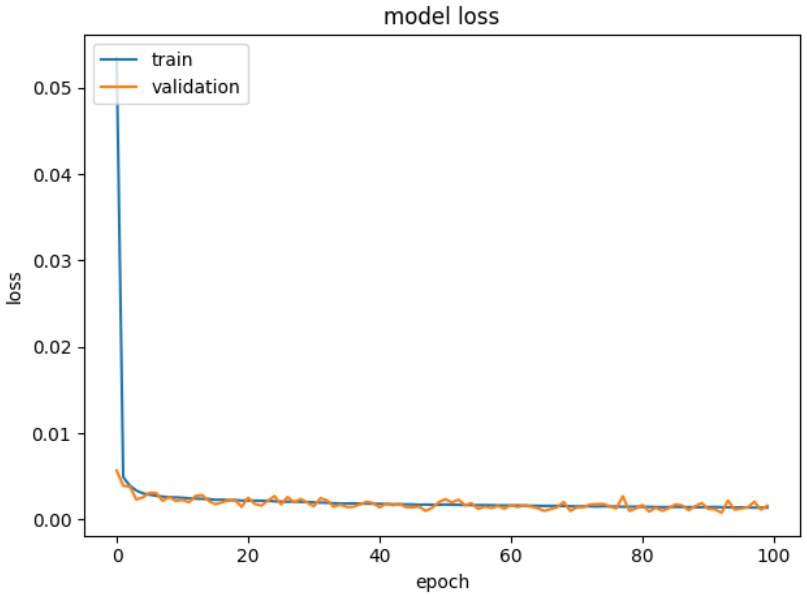
Set of S1 Trajectories



Set of S2 Trajectories

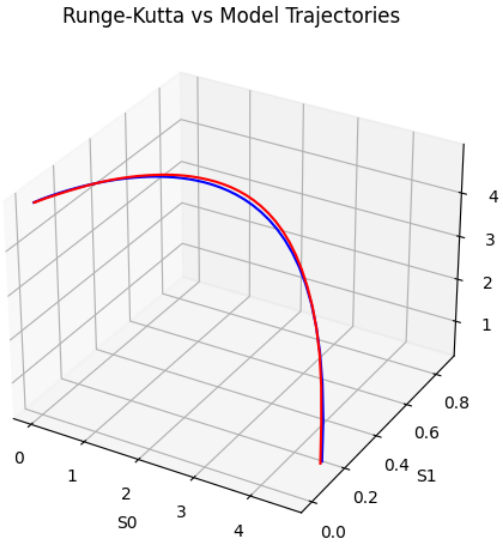
- From these trajectories I also made the training and test data, taking 80% of the 400000 points (200 steps with 2000 choices for the parameters) as training and 20% as testing. 20% of the testing points are used as validation dataset for the training.
- Then, I trained a Neural Network to learn the time-1 map of the system as before, but this time including the parameters of the system as input to the model, keeping the same output.

- The results of the training are shown below:

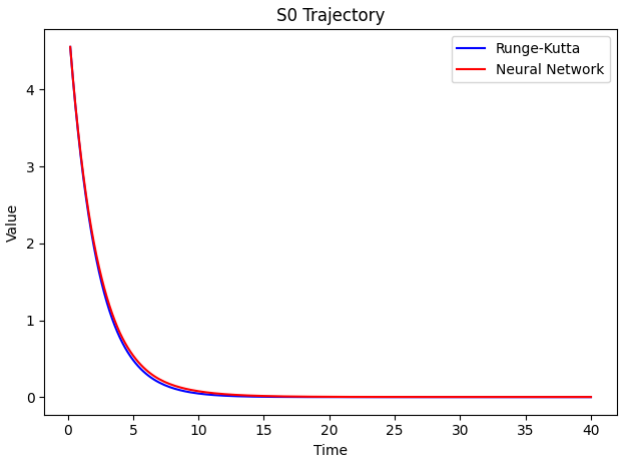



```
Evaluate on training data
2500/2500 [==============================] - 4s 1ms/step - loss: 0.0016
training loss: 0.001609036815352738
Evaluate on test data
625/625 [==============================] - 1s 1ms/step - loss: 0.0016
test loss: 0.001591112813912332
```

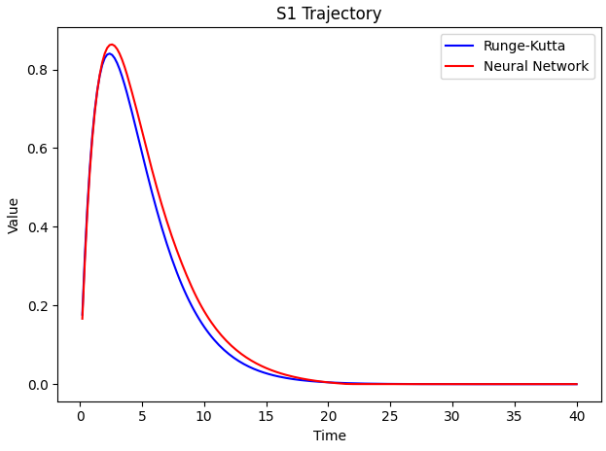
- Then, I plotted the trajectory of the Runge-Kutta method and the one predicted from the time-1 map model, with the base parameters and starting values, to evaluate its accuracy. Blue is the Runge-Kutta, and red the Neural Network:



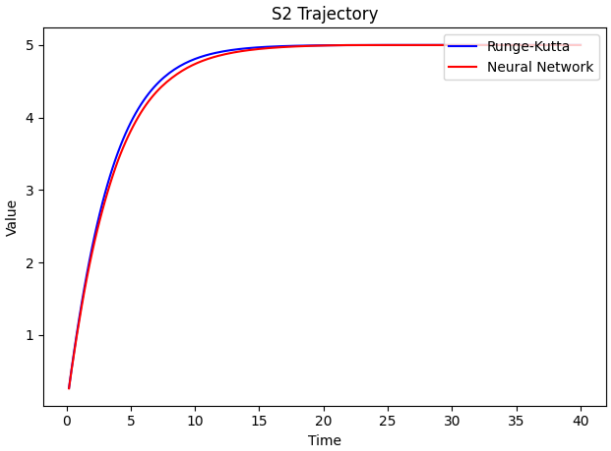
Trajectories in 3D



S0 Trajectories

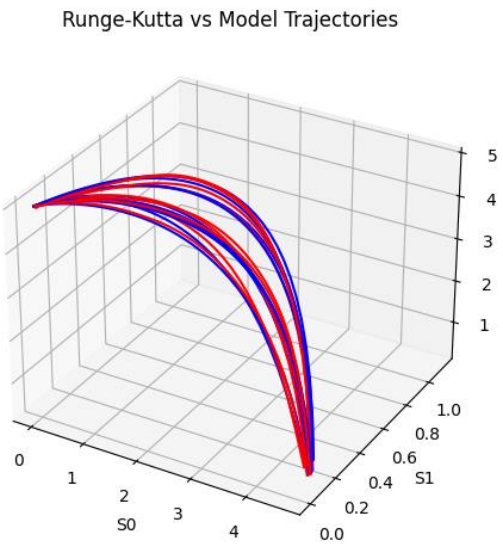


S1 Trajectories

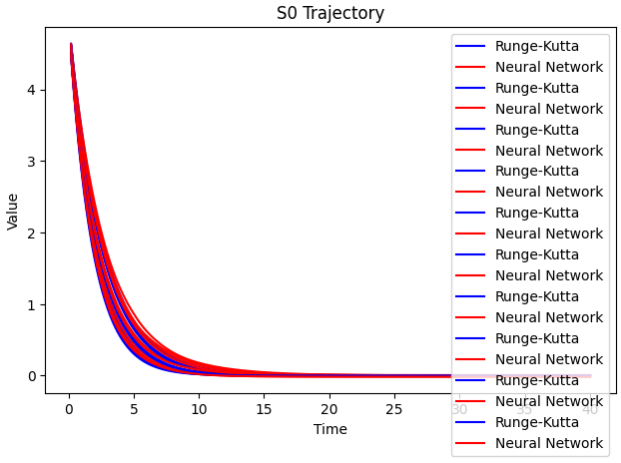


S2 Trajectories

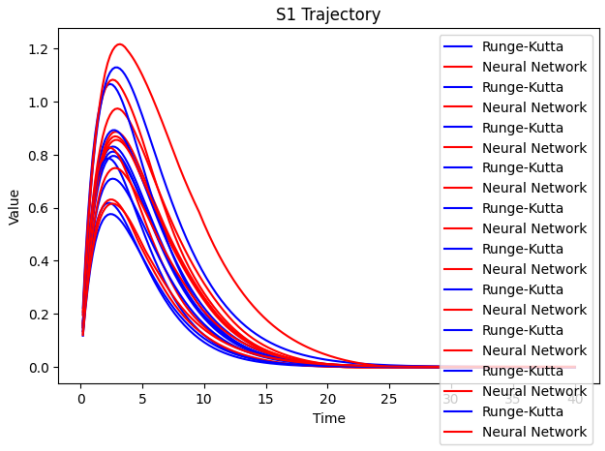
- I also plotted the above trajectories with a random sample of 10 parameters with ±20% deviation from the base parameters:



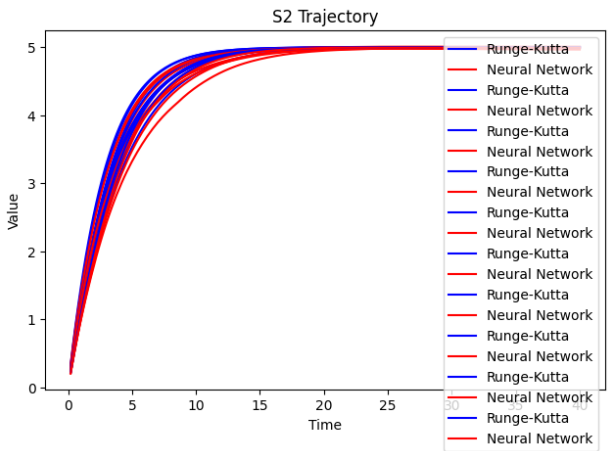
Set of trajectories in 3D



Set of S0 Trajectories

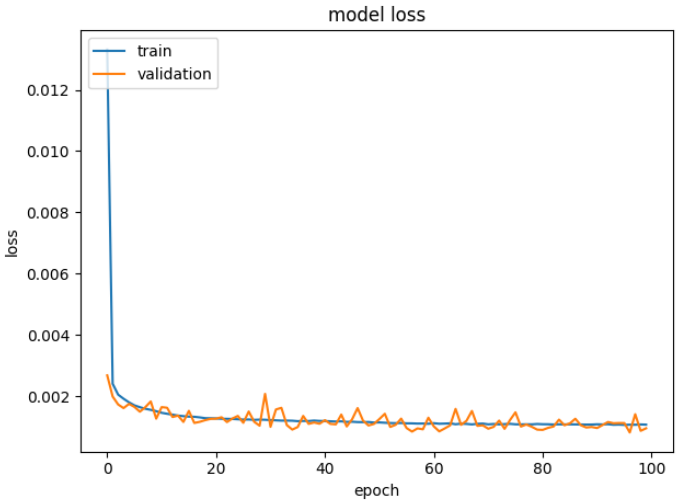


Set of S1 Trajectories



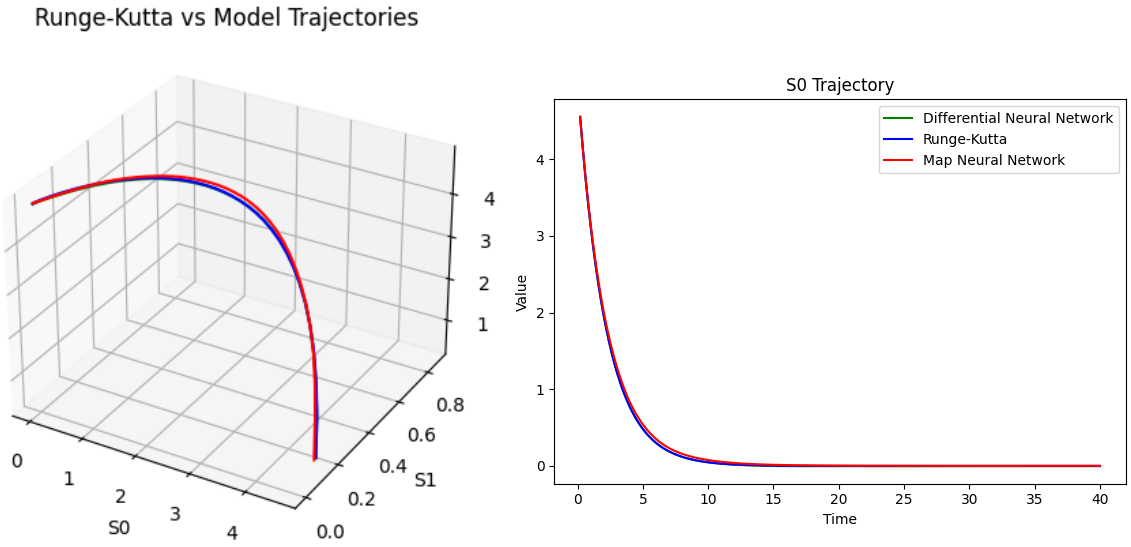
Set of S2 Trajectories

- Then I trained a model that predicts the derivatives of the values at a certain point in the S0, S1, S2 space, with the same structure as the previous experiment, but also including the parameters as input. The training results are shown below:



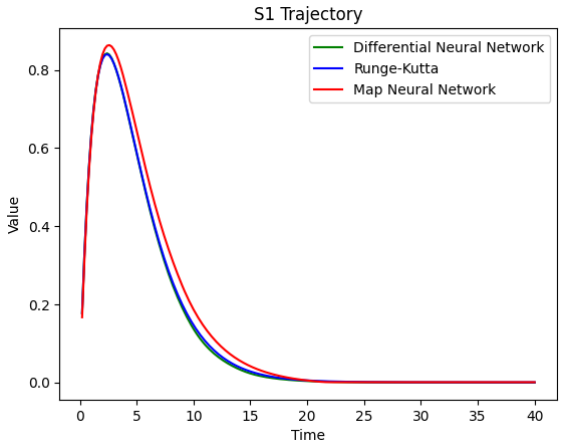

```
Evaluate on training data
   1/2500 [..............................] - ETA: 6:40 - loss: 7.5262e-04
2023-08-03 10:09:16.440562: W tensorflow/tsl/framework/cpu_allocator_impl.cc
2500/2500 [==============================] - 4s 1ms/step - loss: 9.5253e-04
training loss: 0.000952528091147542
Evaluate on test data
625/625 [==============================] - 1s 1ms/step - loss: 9.4190e-04
test loss: 0.000941900652833283
```

- Then I plotted the trajectories of the two different models, along with the ones with the Runge-Kutta method, in order to evaluate their accuracy. Green is the derivative Neural Network trajectory, blue the Runge-Kutta, and red the time-1 map Neural Network (the trajectory from the derivative model isn't visible because it is almost the same as the Runge-Kutta one, as in the previous experiment):

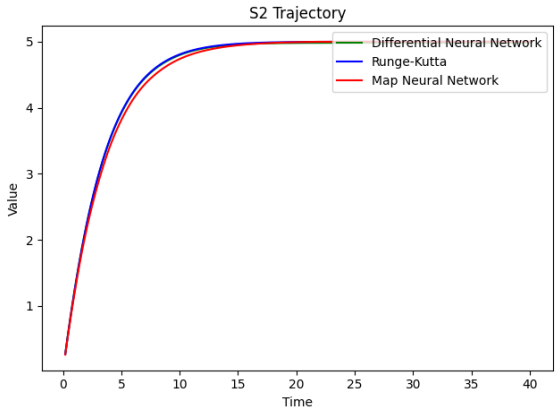


Trajectories in 3D

S0 Trajectories



S1 Trajectories



S2 Trajectories

Resources

- The python notebooks with which I made the plots and trained the models are:
  - https://1drv.ms/u/s!Apr4fLuQXyD6hf4AlqGMt8-LSqHI6w?e=lq30cl the one with the varying starting values
  - https://1drv.ms/u/s!Apr4fLuQXyD6hf4EaijJAbG9FdK2BQ?e=sbrji8 the one with the varying parameters
- The models (in .tf format) are:
  - https://1drv.ms/u/s!Apr4fLuQXyD6hf5UfAazqN_1AGLFEw?e=jQdJK0