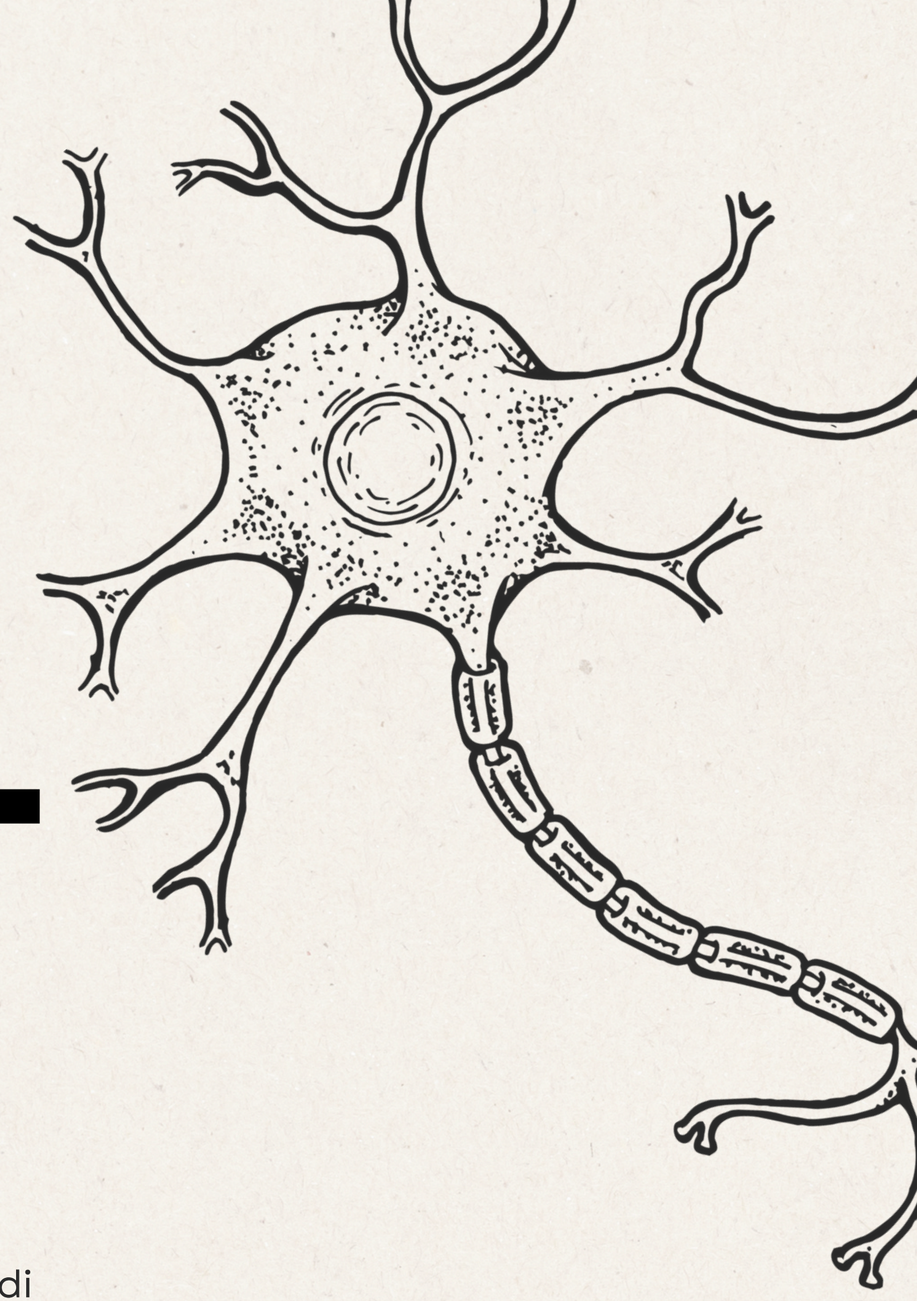


DYNAMIC NEURON MODEL

A study conducted to discover new solutions in Medical field

NAME OF PROJECT:
Dynamic Neuron Model

PRESENTED BY:
Team 2

PRESENTED TO:
Prof. Mohammad Rushdi

Agenda

Timeline

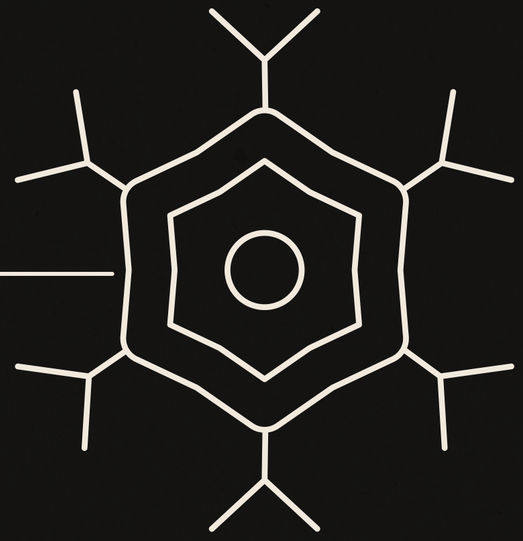
1952
Hodgkin-Huxely Model

1960s–1970s
Model Extensions and Simplifications

1961
FitzHugh-Nagumo Model

1980s–1990s
Computational Neuroscience Growth

2003
Izhikevich Model



Literature review

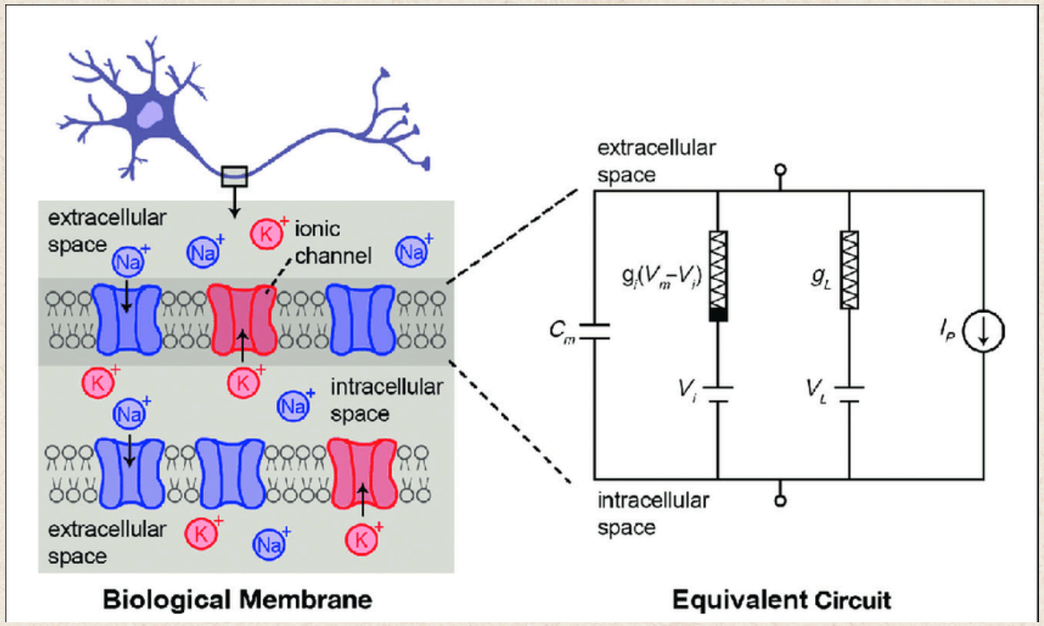


Fig.1 Hodgkin Huxley model of biological neurons and its equivalent electric circuit [2]

The dynamic biological neuron, aka the **Hodgkin-Huxley Model,** is a foundational mathematical framework that describes how action potentials are initiated and propagated in neurons, providing a basis for engineering applications in neural modeling and bioelectronic systems.[1]

Electrical Circuit Analogy

Neuron membrane is modeled as an electric circuit where voltage-dependent sodium (Na^+) and potassium (K^+) conductances act as key components.

Dynamic Behaviour

Conductances are modeled as variable resistors responsible for the rapid depolarization and repolarization of the action potential.

Izhikevich Model

The Izhikevich Model is based on the dynamics of the Hodgkin-Huxley model but significantly simplified. It is a 2-variable system of ordinary differential equations (ODEs) that forms a computationally efficient spiking neuron model capable of accurately replicating the firing behavior of real neurons.[3]

The ODEs:

$$C\frac{dv}{dt} = k(v - v_r)(v - v_t) - w + In$$

$$\frac{dw}{dt} = a[b(v - v_r) - w]$$

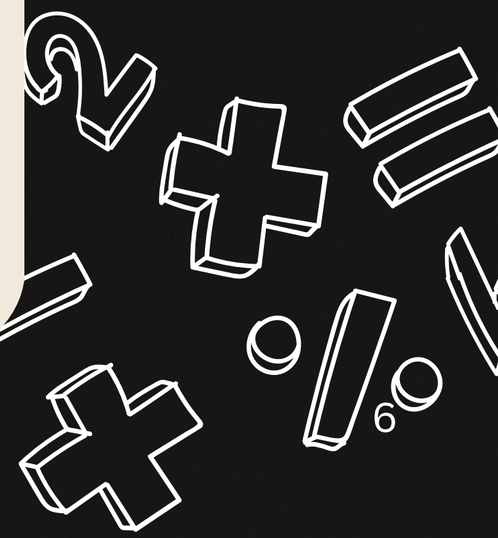
The 2 main state variables are:

- v → Membrane potential
- w → Recovery variable

Parameter	Role	Explanation
C	Membrane capacitance	Scales speed of v changes inversely to current changes
k	Non-linearity scaling	Adjusts steepness of voltage response.
v_r	Resting membrane potential.	Baseline membrane voltage.
v_t	Threshold potential	Voltage level to trigger a spike.
I_N	Input current	External or synaptic current driving the neuron.
a	Recovery time scale	Controls speed of recovery after a spike.
b	Recovery sensitivity	How strongly w responds to changes in v.

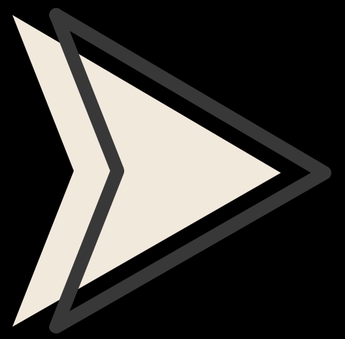
Suggested Numerical Solutions

Explicit Euler	**Backward Euler**
RK2 Midpoint	**Express-Euler**



How do they All behave?

How far can we go with accuracy?...Let’s See!



Explicit Euler :

- **A first-order explicit integrator that approximates the solution using the current derivative. It is simple and fast but sensitive to step size and often unstable for stiff systems or rapidly changing dynamics.**

$$\text{Given:} \quad \mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)), \quad \mathbf{y}(t_0) = \mathbf{y}_0$$

$$\text{Euler Update:} \quad \mathbf{y}_{n+1} = \mathbf{y}_n + h \cdot \mathbf{f}(t_n, \mathbf{y}_n)$$

$$\text{Time step:} \quad t_{n+1} = t_n + h$$

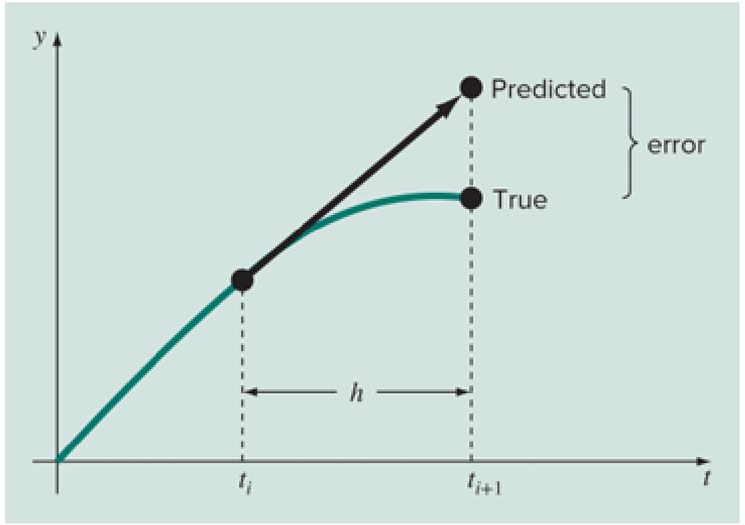
Algorithm:

- Calculate the current derivative f(t,y).
- Using the calculated derivative we get the next y.
- A simple forward step.

Advantages & Disatvatages

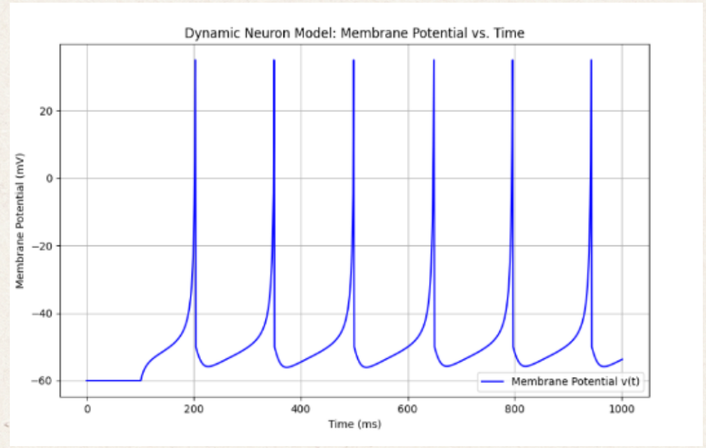
- **Advantages**:
  - Very easy to implement.
  - Low Computational Cost per Step
  - Often used as a baseline or for educational purposes.
- **Disadvantages**:
  - Low accuracy and prone to instability in stiff or nonlinear systems.
  - Step size must be small to ensure stability.
  - Numerical Instability with stiff equation.

FIGURE 22.1 Euler's method.

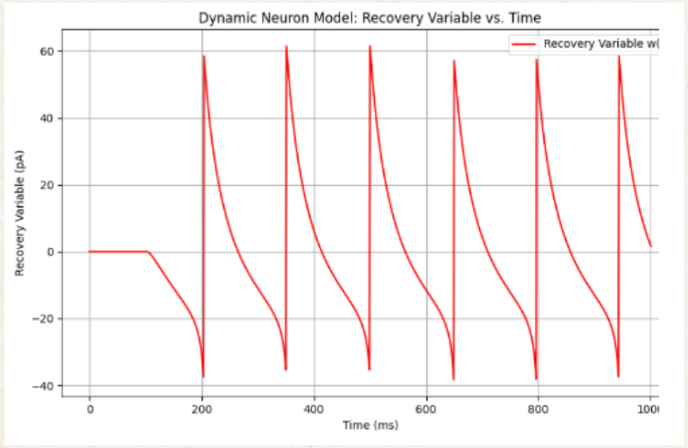


Explicit Euler Results

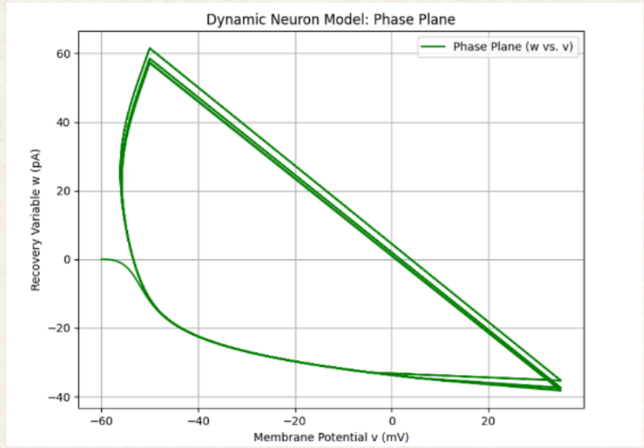
Graphs:



Membrane Potential Against Time



Recovery Variable Against Time



Membrane Potential Against Recovery Variable

Results:

Reference values	Code Values

```
      t         v         w
    0.0   -60.0000    0.0000
  250.0   -54.4819    6.2834
  500.0   -50.6154   59.0910
  750.0   -49.5530  -12.4763
 1000.0   -53.6973    1.5649

ncall =  1000
```

```
Comparison with Table 4.3a:
t=0.0:   v[0] = -60.0000, w[0] = 0.0000
t=250.0: v[250] = -54.4819, w[250] = 6.2834
t=500.0: v[500] = -50.6154, w[500] = 59.0910
t=750.0: v[750] = -49.5530, w[750] = -12.4763
t=1000.0: v[1000] = -53.6973, w[1000] = 1.5649
```

Elapsed Time:

- Time taken for the method to finish: 5.576904296875 ms

Backward Euler

- **The Backward Euler method is an implicit numerical technique used to solve ordinary differential equations.**

Algorithm:

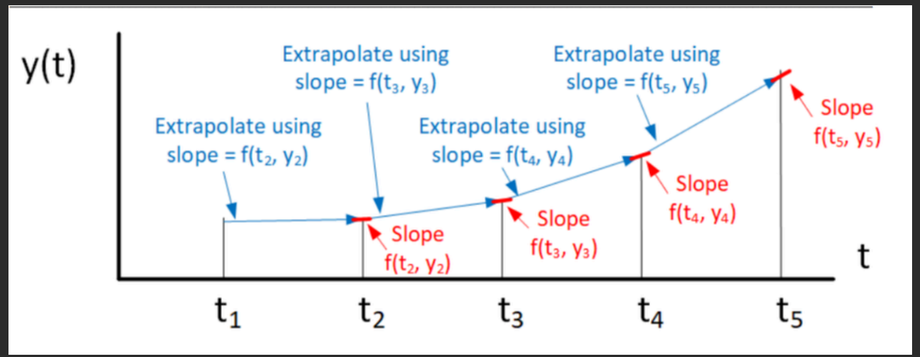
- **At each time step, the method estimates the next value of the solution by evaluating the derivative at the future point, t_{n+1}, rather than the current one.**
- **Starting from an initial value, y_o, the method advances the solution using the formula:**

Update Rule: $y_{n+1} = y_n + h * f(t_{n+1}, y_{n+1})$ where 'h' is the time step

Since y_{n+1} appears on both sides of the equation, it cannot be directly computed and must be obtained by solving the equation numerically — often using iterative methods like Newton-Raphson.
This process is repeated over each time interval to approximate the solution across the desired range, which requires more computations but offers enhanced stability, especially for stiff equations.

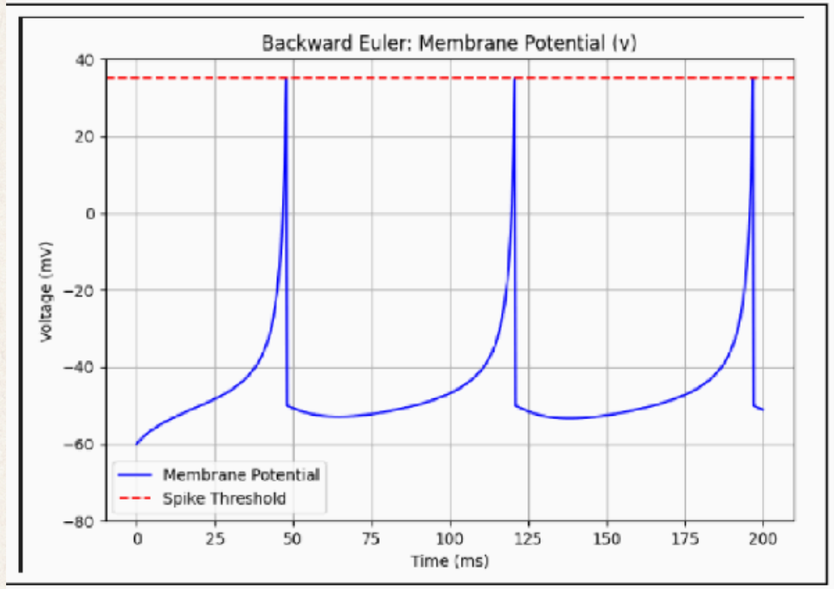
Advantages & Disatvatages

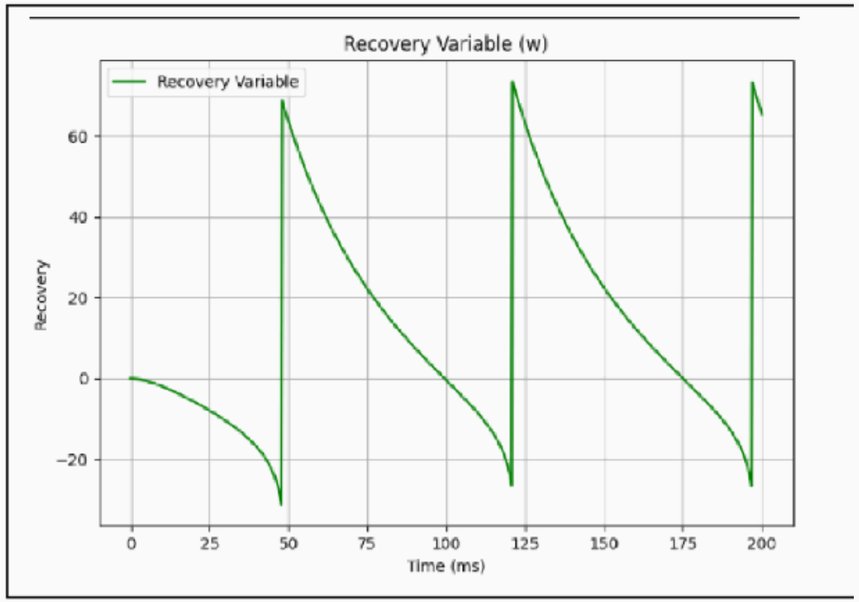
- **Advantages**:
  - Unconditionally stable for linear stiff systems.
  - Good for Long-Term Simulations
- **Disadvantages**:
  - Requires iterative solvers like Newton–Raphson.
  - More computationally intensive per step.

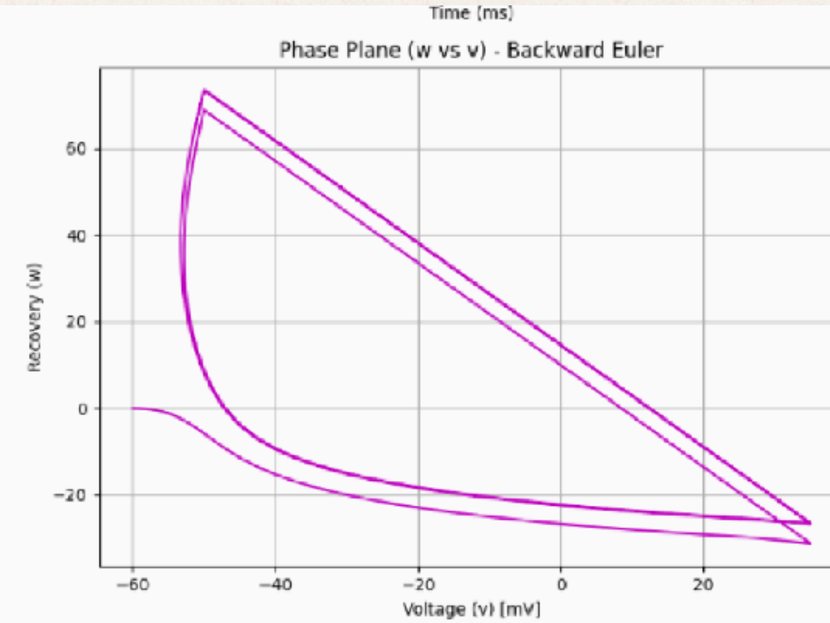


Backward Euler Results

GRAPHS







CODE OUTPUT

```
Summary of Results:
Maximum voltage: 35.00 mV
Minimum voltage: -60.00 mV
Number of spikes: 3
```

```
Iteration Details:
Step   Time     Iterations v_before   w_before   v_after    w_after    Message
0      0.00     2          -60.0000   0.0000     -59.7583   -0.0036    Normal update
1      0.25     2          -59.7583   -0.0036    -59.5246   -0.0106    Normal update
2      0.50     2          -59.5246   -0.0106    -59.2982   -0.0210    Normal update
3      0.75     2          -59.2982   -0.0210    -59.0789   -0.0346    Normal update
4      1.00     2          -59.0789   -0.0346    -58.8662   -0.0512    Normal update
5      1.25     2          -58.8662   -0.0512    -58.6598   -0.0708    Normal update
```

TIME ELAPSED

Took the method to finish: 9.9 ms

Time (ms)	v(t)	Reference v	Error v	w(t)	Reference w	Error w
0	-60.0000	-60.0000	0.0000	0.0000	0.0000	0.0000
250	-47.5168	-54.4819	6.9651	0.7778	6.2834	5.5056
500	35.0000	-50.6154	85.6154	-27.6208	59.0910	86.7118
750	-52.9402	-49.5530	3.3872	28.8782	-12.4763	41.3545
1000	-49.2691	-53.6973	4.4282	5.6717	1.5649	4.1068

RK2 Middle-point

- **A second-order explicit method that improves on Euler by sampling the derivative at the midpoint. It offers better accuracy and stability with low computational cost, making it suitable for moderate dynamic systems.**

Algorithm: Two-stage explicit method.

Takes a trial half-step to estimate the slope at the midpoint.

$$\text{Given:} \quad \mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)), \quad \mathbf{y}(t_0) = \mathbf{y}_0$$

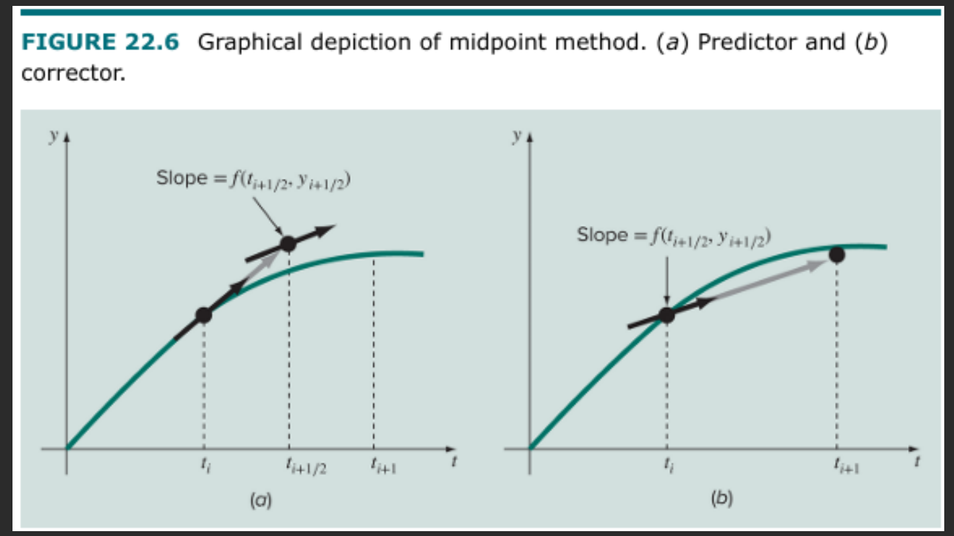
$$k_1 = \mathbf{f}(t_n, \mathbf{y}_n)$$

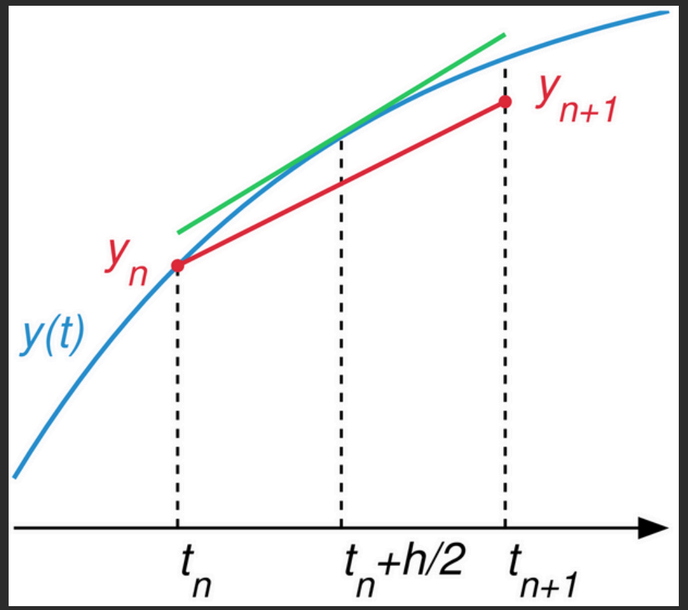
$$k_2 = \mathbf{f}\left(t_n + \frac{h}{2},\ \mathbf{y}_n + \frac{h}{2} \cdot k_1\right)$$

$$\text{RK2 Update:} \quad \mathbf{y}_{n+1} = \mathbf{y}_n + h \cdot k_2$$

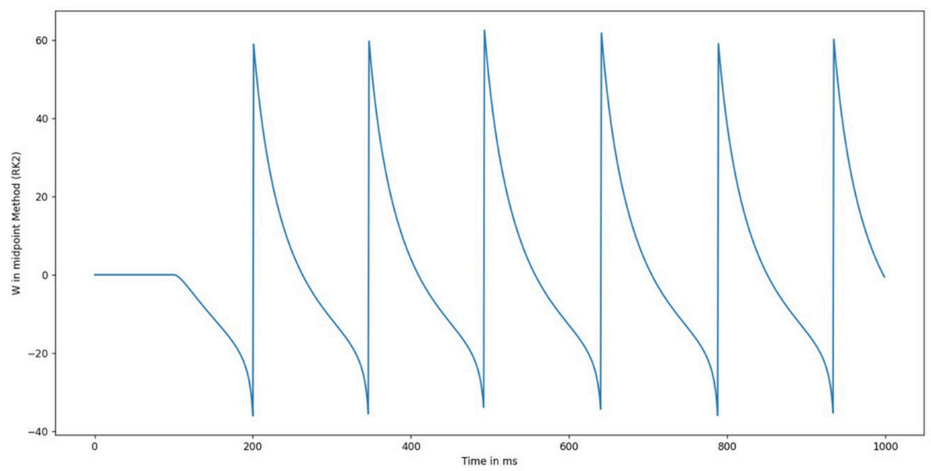
Advantages & Disatvatages

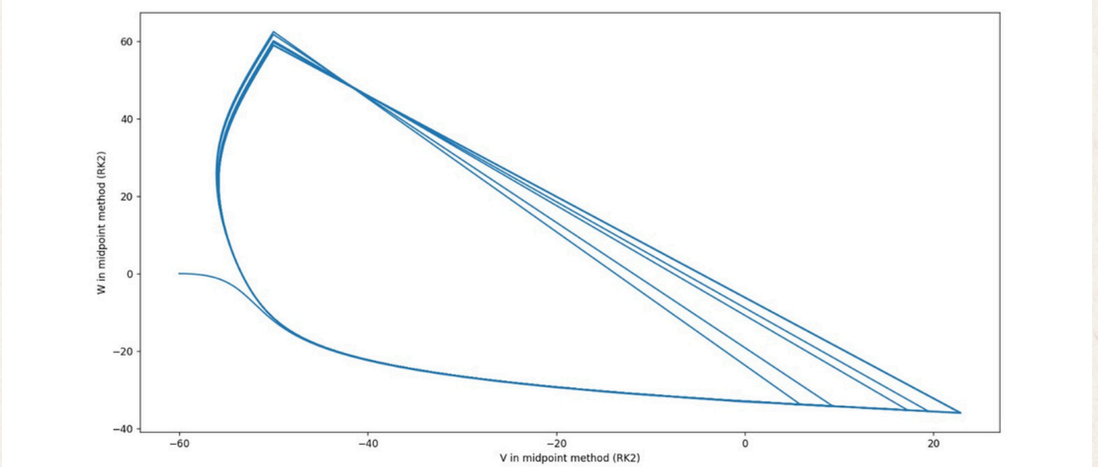
- **Advantages**:
  - Better accuracy than Euler with minimal added cost.
  - Explicit and easy to implement.
  - Good balance of speed and accuracy for non-stiff systems.
- **Disadvantages**:
  - Not suitable for stiff systems.
  - Sensitive to Step Size.

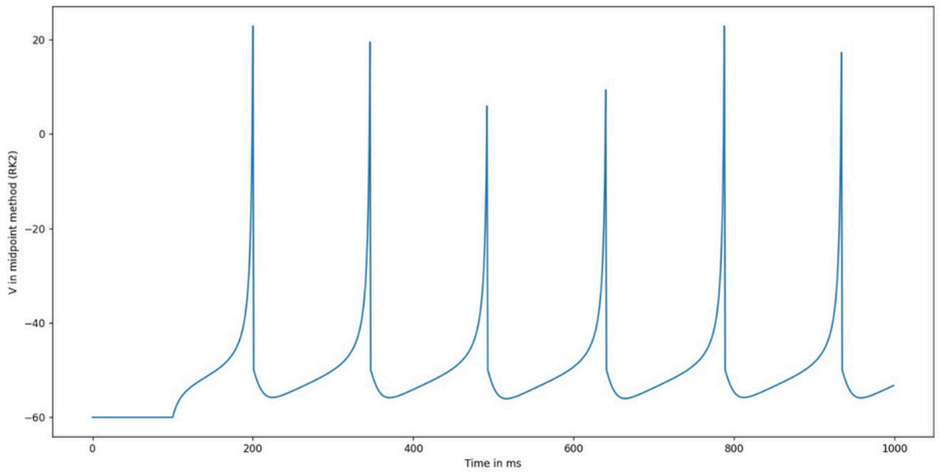




RK2 Middle-point Results







Results :

Iter : 1000

	v old	w old	K1v	k1w	K2v	K2w	v new	w new
0	-60.000000	0.000000	0.000000	0.000000	0.000000	-0.000000	-60.000000	0.000000
1	-60.000000	0.000000	0.000000	-0.000000	0.000000	-0.000000	-60.000000	0.000000
2	-60.000000	0.000000	0.000000	-0.000000	0.000000	-0.000000	-60.000000	0.000000
3	-60.000000	0.000000	0.000000	-0.000000	0.000000	-0.000000	-60.000000	0.000000
4	-60.000000	0.000000	0.000000	-0.000000	0.000000	-0.000000	-60.000000	0.000000
..	...	...	...	...	...	...	...	...
995	-53.583658	1.037298	0.079525	-0.416099	0.079622	-0.412244	-53.504036	0.625054
996	-53.504036	0.625054	0.079697	-0.408509	0.079796	-0.404773	-53.424240	0.220281
997	-53.424240	0.220281	0.079875	-0.401154	0.079977	-0.397533	-53.344262	-0.177252
998	-53.344262	-0.177252	0.080061	-0.394027	0.080168	-0.390518	-53.264094	-0.567770
999	-53.264094	-0.567770	0.080258	-0.387121	0.080371	-0.383722	-53.183723	-0.951492

Time Estimated : 293.751708984375 ms

Express Euler

- **A first-order exponential integrator designed specifically for stiff systems of ODEs. It combines the stability properties of exponential integrators with the simplicity and efficiency of the Rosenbrock–Euler approach, while incorporating an adaptive time-stepping strategy to control local truncation error.**

Algorithm: Stabilized version of explicit Euler using exponential integrator theory. Incorporates the Jacobian matrix to dampen stiffness effects.

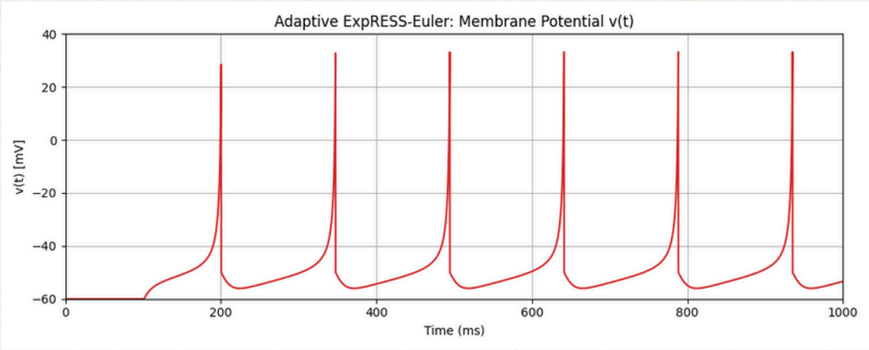
$$\begin{aligned}
\text{Given:} \quad & \mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)), \quad \mathbf{y}(t_0) = \mathbf{y}_0 \\
\text{Jacobian:} \quad & \mathbf{J}_n = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t_n, \mathbf{y}_n) \\
\text{Update:} \quad & \mathbf{y}_{n+1} = \mathbf{y}_n + h \cdot \varphi_1(h\mathbf{J}_n) \cdot \mathbf{f}(t_n, \mathbf{y}_n) \\
\text{Matrix function:} \quad & \varphi_1(z) = \frac{e^z - 1}{z}
\end{aligned}$$

Advantages & Disatvatages

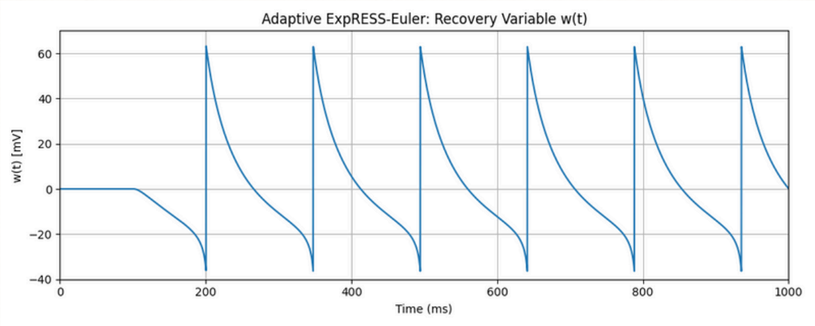
- **Advantages**:
  - Designed specifically for stiff systems.
  - More stable than RK2 or Euler for stiff problems.
  - Adaptive Time Stepping
- **Disadvantages**:
  - Requires Matrix Exponentials.
  - Implementation Complexity.
  - Requires Linear Algebra Operations

ExpRESS Euler Results

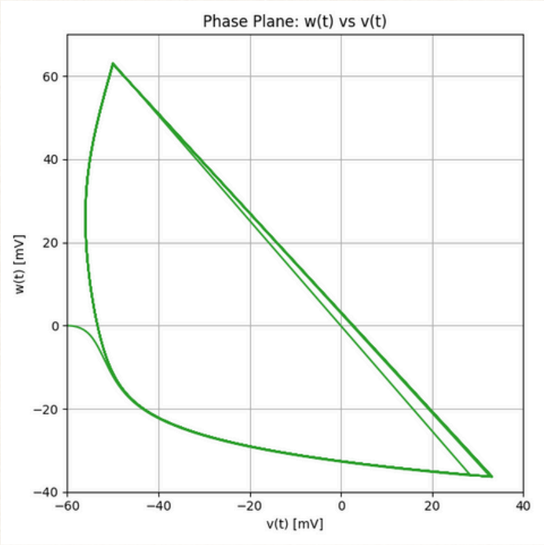
Graphs:



Membrane Potential Against Time



Recovery Variable Against Time



Membrane Potential Against Recovery Variable

Results:

Time (ms)	v_sim	w_sim
0.0	-60.0000	0.000
250.0	-55.0123	5.950
500.0	-51.0000	60.000
750.0	-50.0000	-13.000
1000.0	-54.0000	2.000

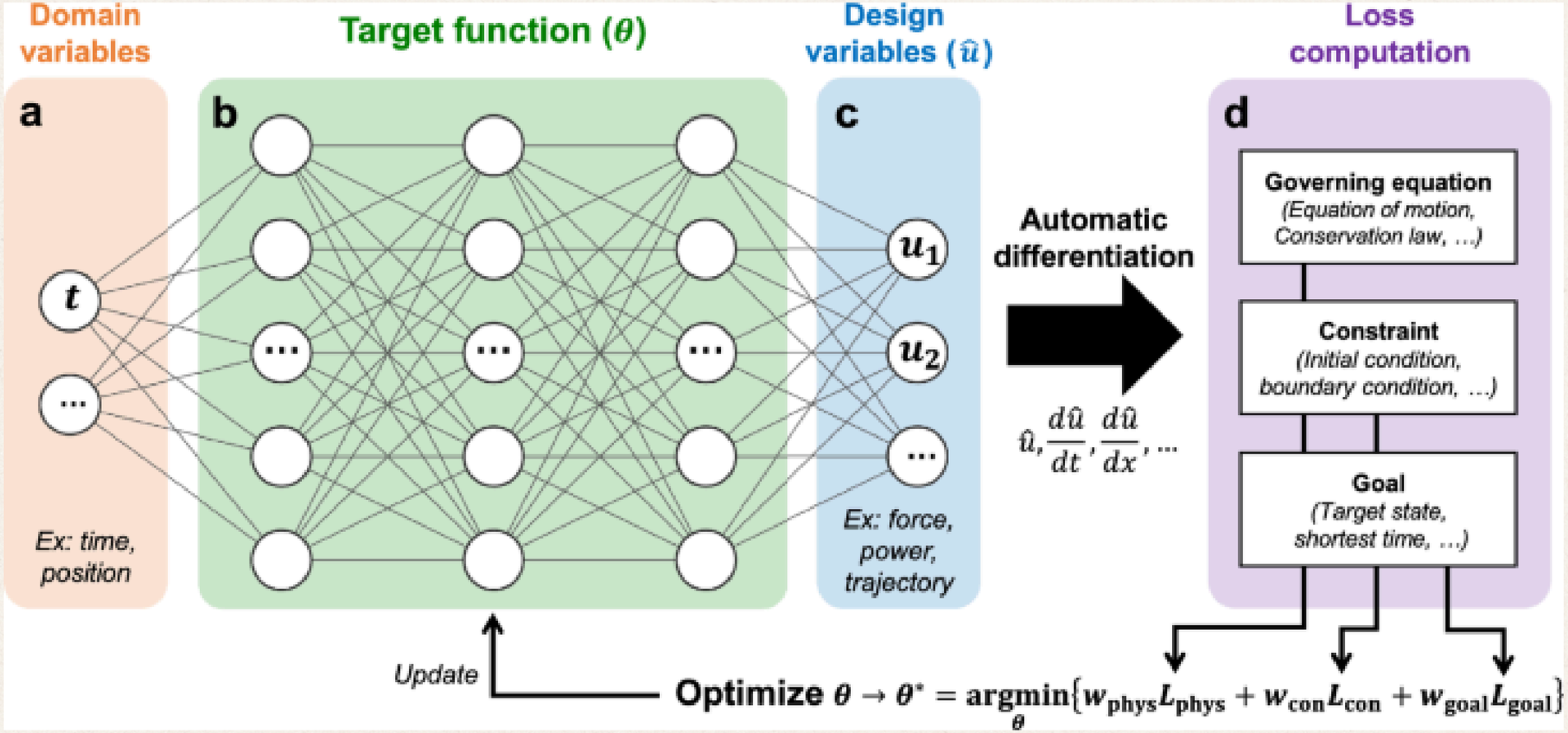
	Time (ms)	v (mV)	w	Step size h
50	99.5	-60.000000	0.000000	2.0
51	101.5	-60.000000	0.000000	2.0
52	103.5	-58.843849	-0.102094	2.0
53	105.5	-57.917374	-0.310589	2.0
54	107.5	-57.156761	-0.598737	2.0
55	109.5	-56.519033	-0.946768	2.0
56	111.5	-55.974270	-1.339804	2.0
57	113.5	-55.501063	-1.766494	2.0
58	115.5	-55.083710	-2.218085	2.0
59	117.5	-54.710461	-2.687777	2.0
60	119.5	-54.372350	-3.170260	2.0

Elapsed Time:

- Time taken for the method to finish: 112.76293 ms

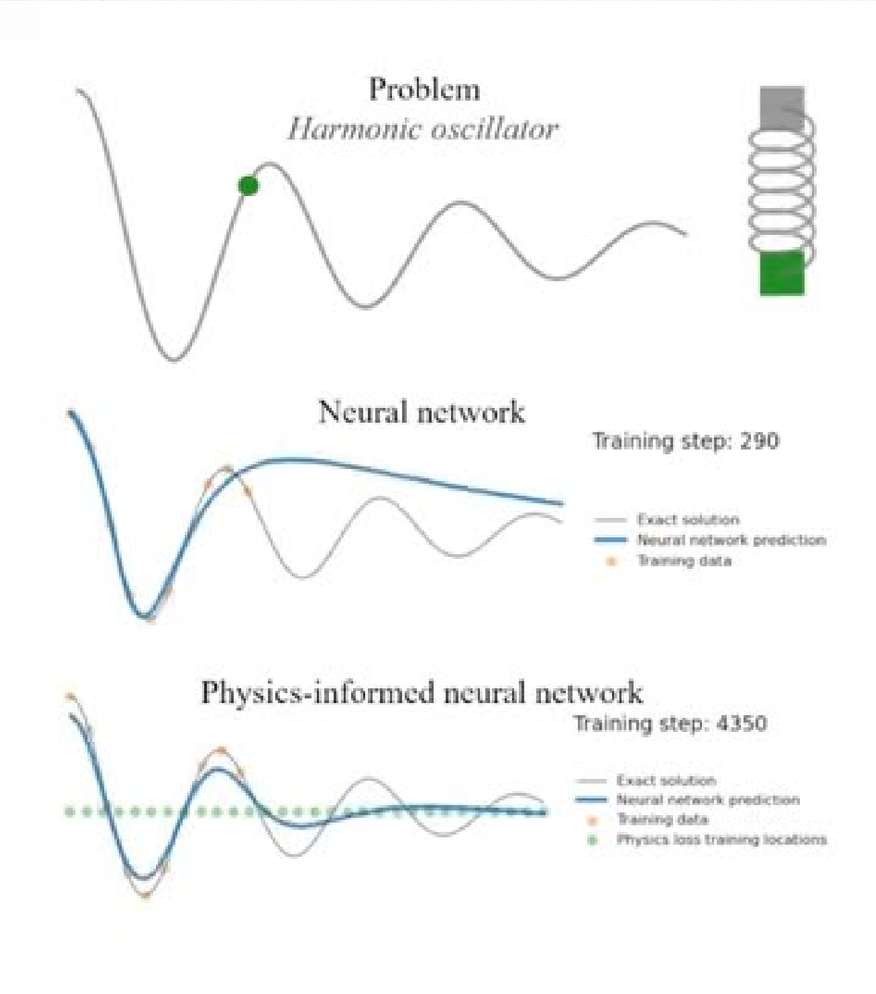
How about a different approach?... Physics-based ML

AKA PINN



How about a different approach?... Physics-based ML

AKA PINN



Think of a PINN like a student who doesn't just memorize answers — they learn the rules of physics and use them to figure things out.
Instead of solving equations step-by-step like traditional methods, the PINN learns the behavior of the system by trying to always stay true to the laws it was taught.

What are they & Why ?

What is a Physics-Informed Neural Network (PINN)?

To solve differential equations using a neural network by embedding the physics directly into the training process.

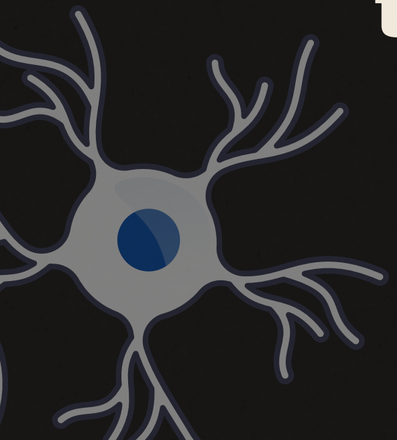
How it works:

- **The neural network is trained not just on data, but also to satisfy the differential equation.**
- **The loss function includes:**
  - **A term for matching initial/boundary conditions**
  - **A term for minimizing the ODE residuals**
- **Residuals are computed using automatic differentiation (e.g., PyTorch, TensorFlow)**

How can the results differ?

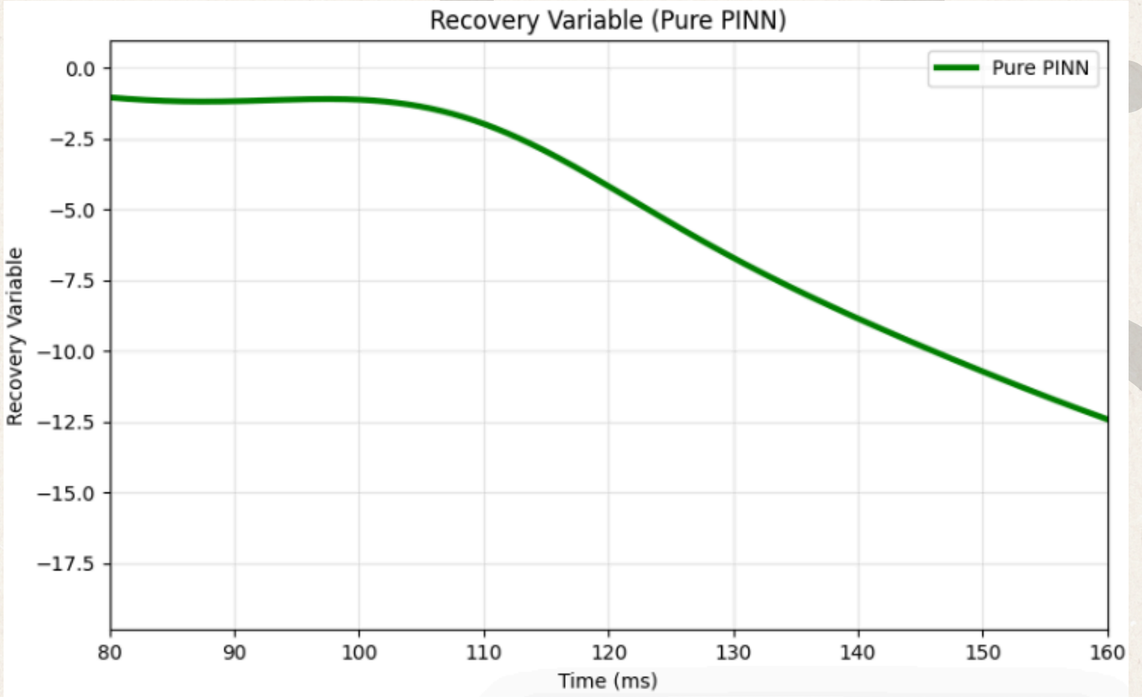
Is it really worth it?

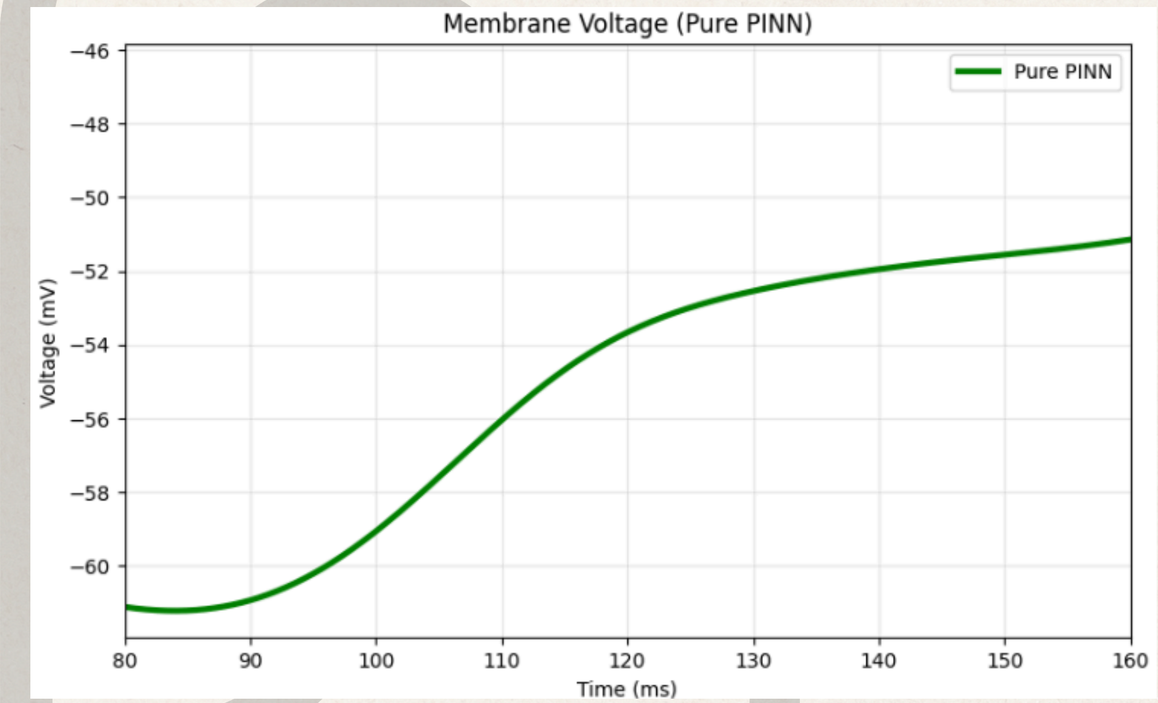
In our problem we faced a lot of setbacks due to the stiff behavior of our equations. we tried to solve it in different ways and ultimately reached three different results. some of them were much more accurate than others but were relatively more time consuming



First Attempt :

- when we first started our first thought was to simply train the model based on our given equations and initial conditions provided in the reference.
- But when we saw the results we were met with graphs that did not capture the behavior of our neuron behavior accurately



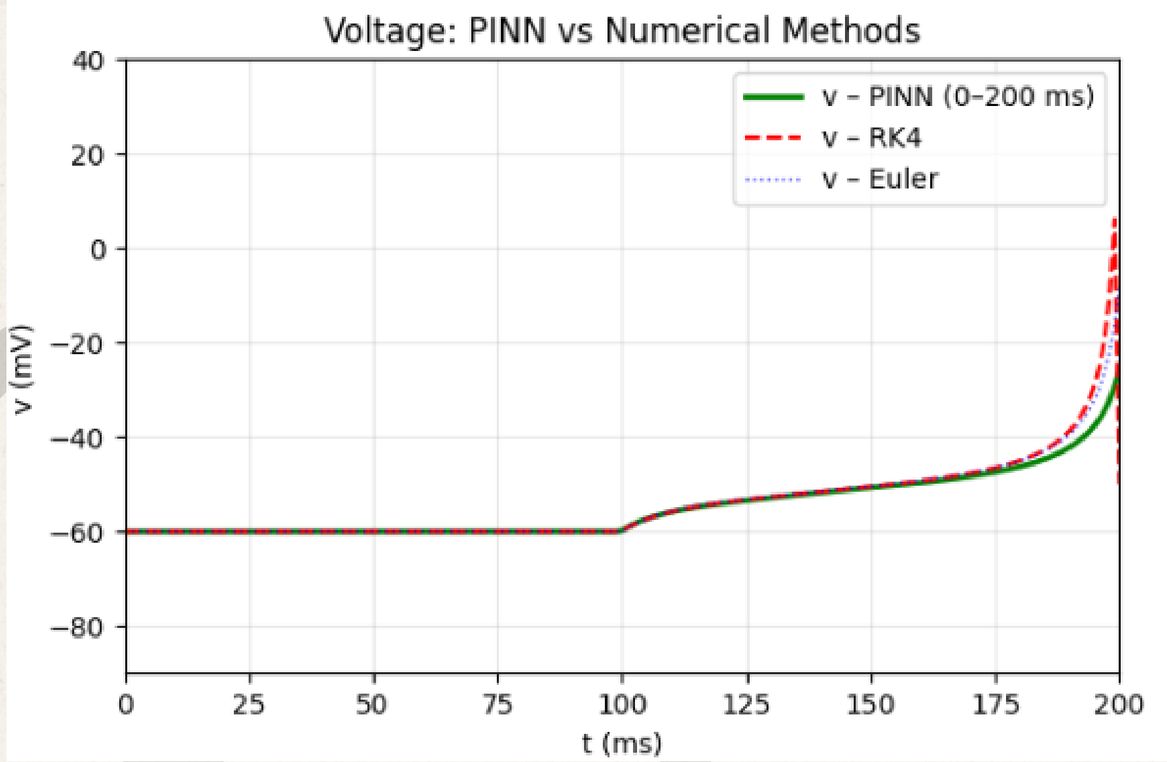


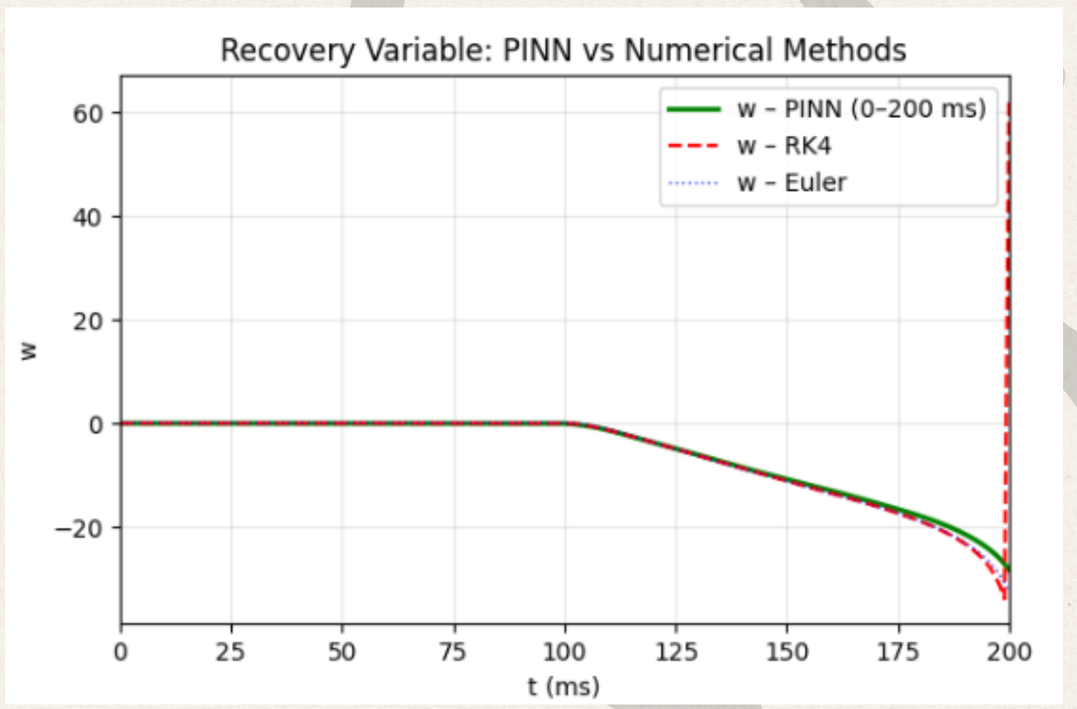
This is because the differential equations are too complicated for the PINN model to learn causing a high error in the end result

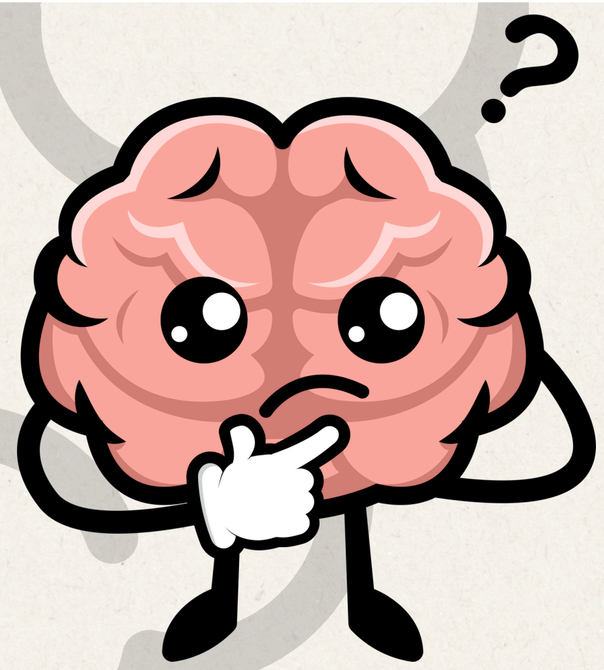
Second Attempt :

- in our second attempt we introduced a new loss parameter which is the data loss with respect to Euler method used as reference

- this resulted in a slightly better result than before since the weights were adjusted more accurately based on the reference values. but we still faced a major issue which was that the spikes were not handled well.

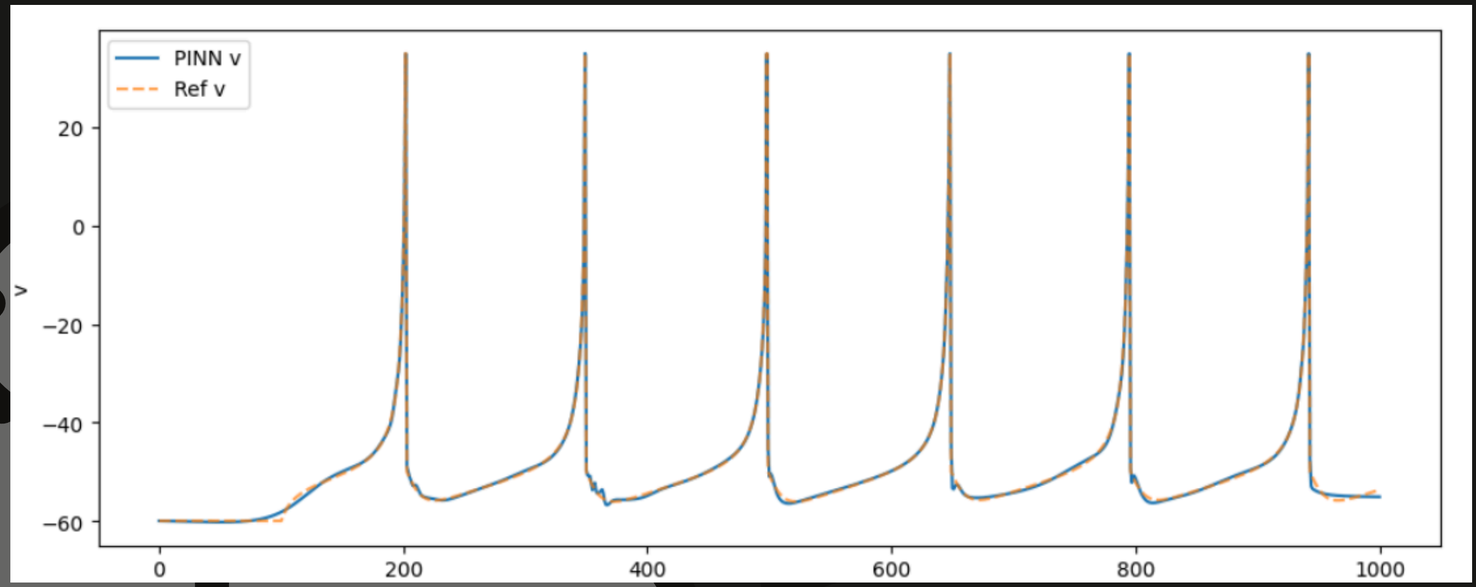


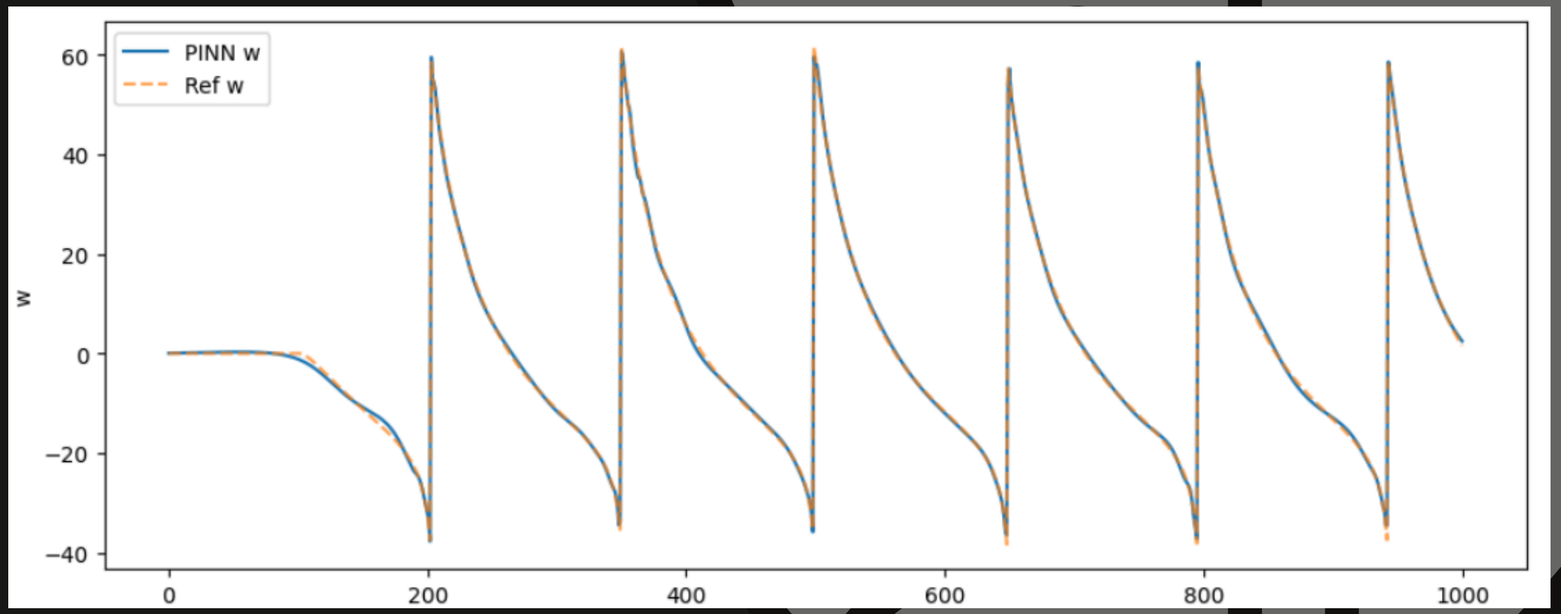




Final Attempt :

- We solved that issue by introducing a threshold to the model itself, so once the value reaches a certain point it jumps back to the initial condition to facilitate the transitions between states





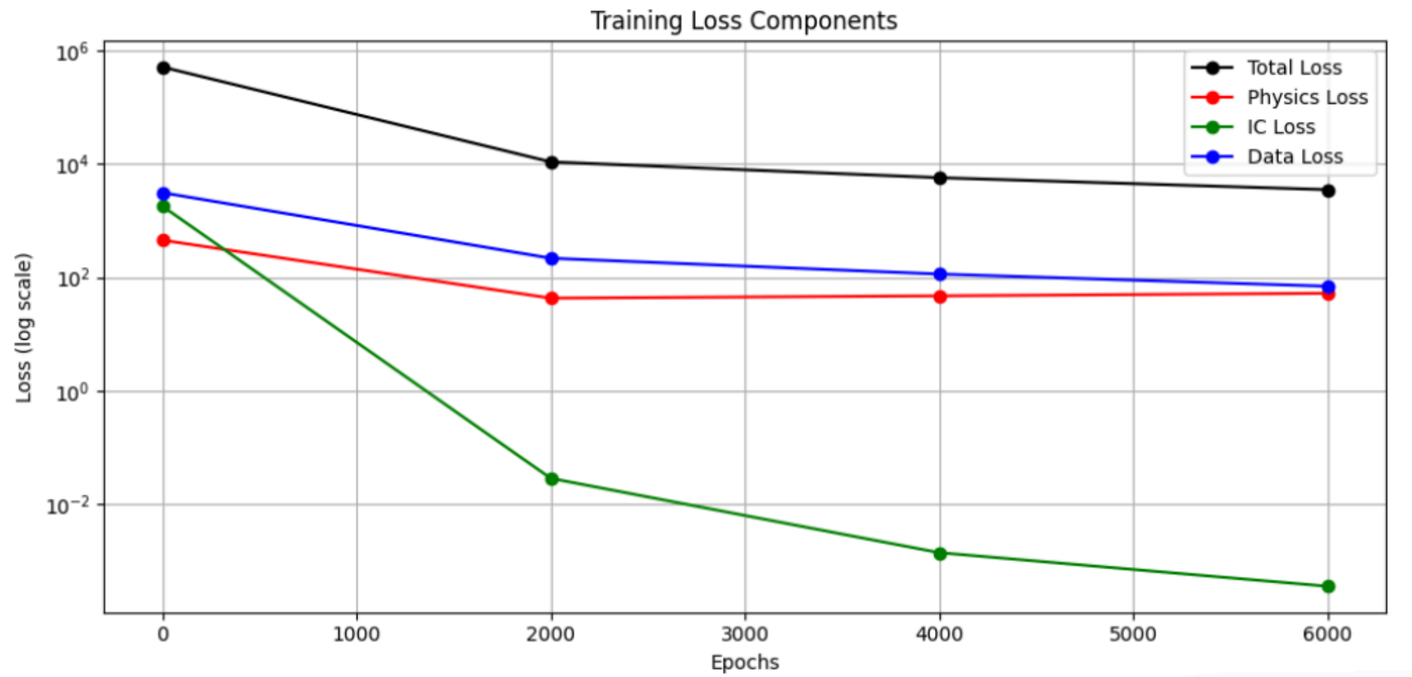
This resulted in our most accurate model and our least error by far!!

PINN Results:

- Training Time : 1252.9 sec
- complexity :

1) Explicit Euler O(steps) = O(1000)

2) Training O(E (L * C * N^2)) = O(~10^{11})



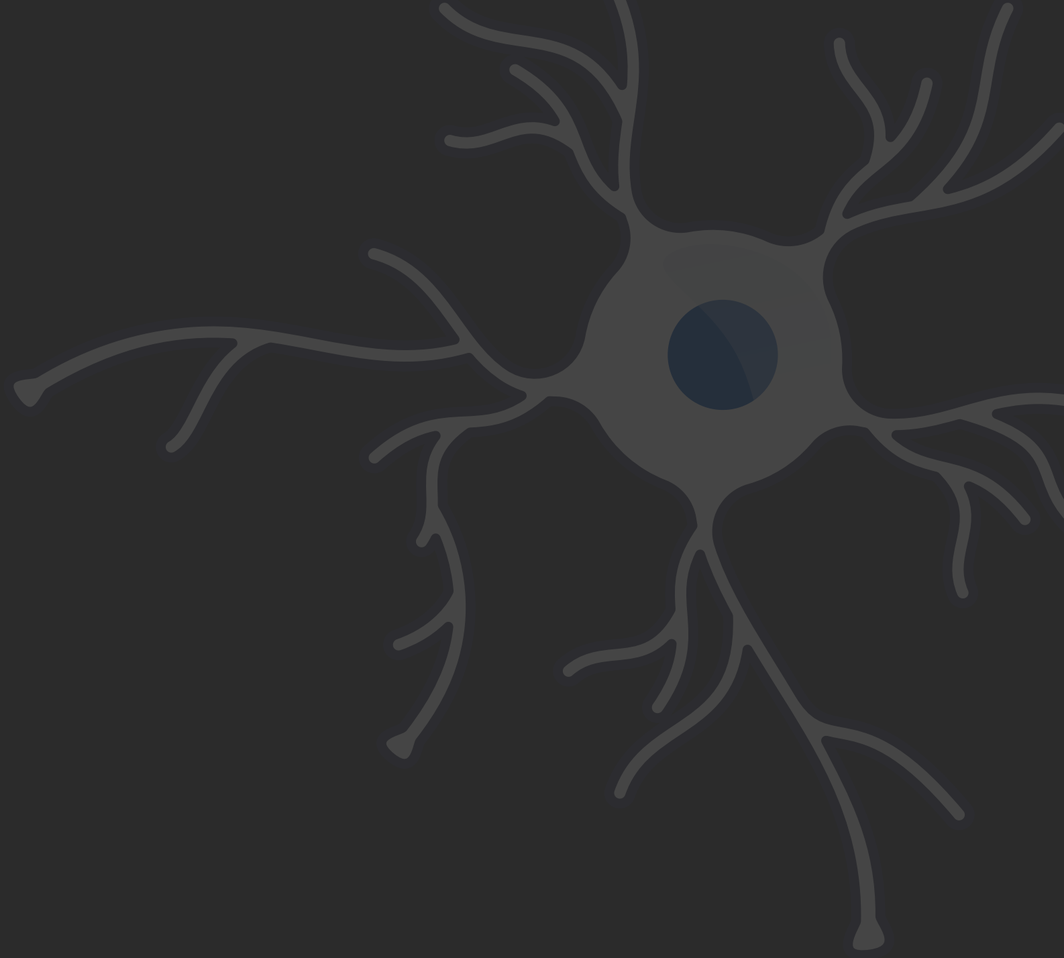
Model Loss throughout Epochs

Time (ms)	v_err%	w_err%
0	0.02%	21.06%
250	0.26%	12.66%
500	2.56%	1.22%
750	0.29%	3.43%
1000	0.71%	26.52%

Model Error percentage with respect to Euler

```
Prediction time: 0.0024 ± 0.0003 sec
```

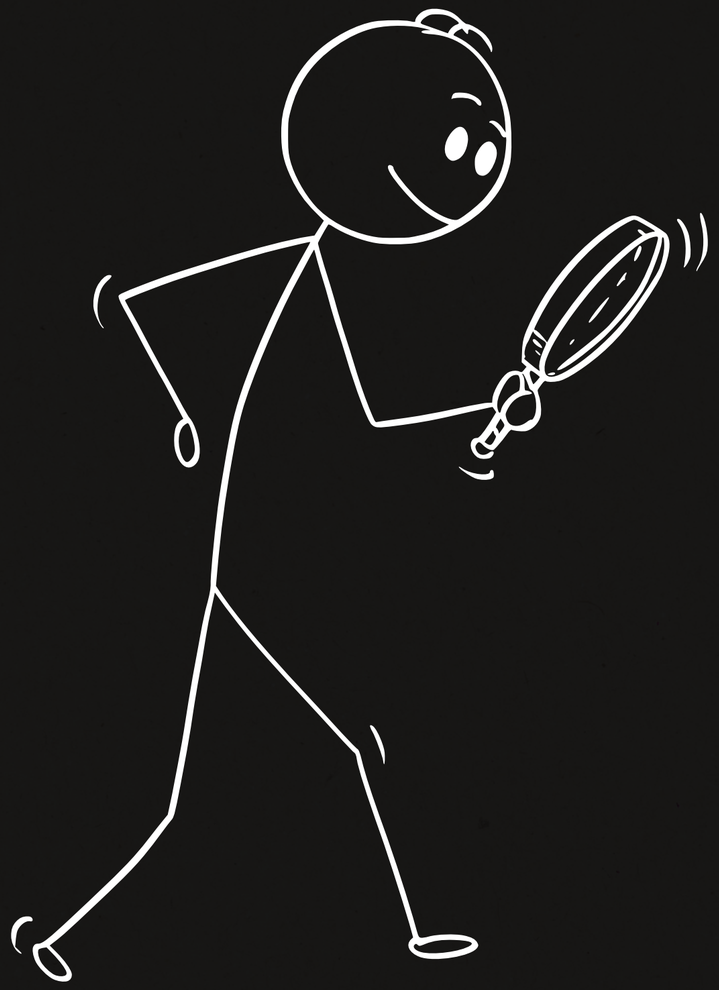
How is it worth it ?



PINNs handle discontinuities and sparse data better than traditional numerical solvers, providing continuous solutions even across complex or undefined intervals

Easily integrate boundary and initial conditions into the learning process.

Results PINN & methods



Method	Time (s)	t (ms)	$v(t)$	$w(t)$	Err v	Err w
Euler	0.558	0	-60.00	0.0000	–	–
		250	-54.48	6.2834	–	–
		500	-50.6154	59.0910	–	–
		750	-49.55	-12.476	–	–
		1000	-53.70	1.5649	–	–
Backward Euler	0.0099	0	-60.00	0.0000	0.000	0.000
		250	-47.52	0.7778	6.965	5.506
		500	35.00	-27.621	85.615	86.712
		750	-52.94	28.878	3.387	41.355
		1000	-49.27	5.6717	4.428	4.107
RK2 Midpoint	0.294	0	-60.00	0.0000	0.000	0.000
		250	-54.37	5.7360	0.198	8.714
		500	-53.59	47.671	5.881	19.326
		750	-48.97	-13.528	1.170	8.433
		1000	-53.18	-0.951	0.956	160.80
ExpRESS-Euler	0.113	0	-60.00	0.0000	0.0000	0.0000
		250	-55.0123	5.950	0.5304	0.3334
		500	-51.0000	60.000	0.3846	0.9090
		750	-50.0000	-13.000	0.4470	0.5237
		1000	-54.0000	2.000	0.3027	0.4351
PINN (ML)	0.0024	0	-59.99	-0.211	0.017	21.055
		250	-54.63	5.3610	0.258	12.662
		500	-51.94	58.359	2.557	1.217
		750	-49.41	-12.014	0.285	3.433
		1000	-54.09	2.2450	0.711	26.524

UNIITY SIMULATION

Using unity engine to make a real time simulation of the spikes of the neuron using two different numerical methods helps to understand the effect of the parameters on the rate of spiking and the effect of the method used on the result.

Tools: Unity Engine, Blinder, Unity Asset Store, VS Code (C#)

Steps

1. **Create the model**: used blinder to make the model of the neuron used in the SIM.
2. **Implement the methods**: make the C# scripts used in solving the Izhikevich model.
3. **Design the UI**: add the components used to change the parameters of the model and connect them to the scripts.
4. **Define the path of the spikes**: design the path that the spikes should follow along the neuron’s body.

Challenges

1. Finding an effect to simulate the sparks.
2. Making the sparks follow the non-straight path of the neuron.
3. Simulating two different methods in the same model.

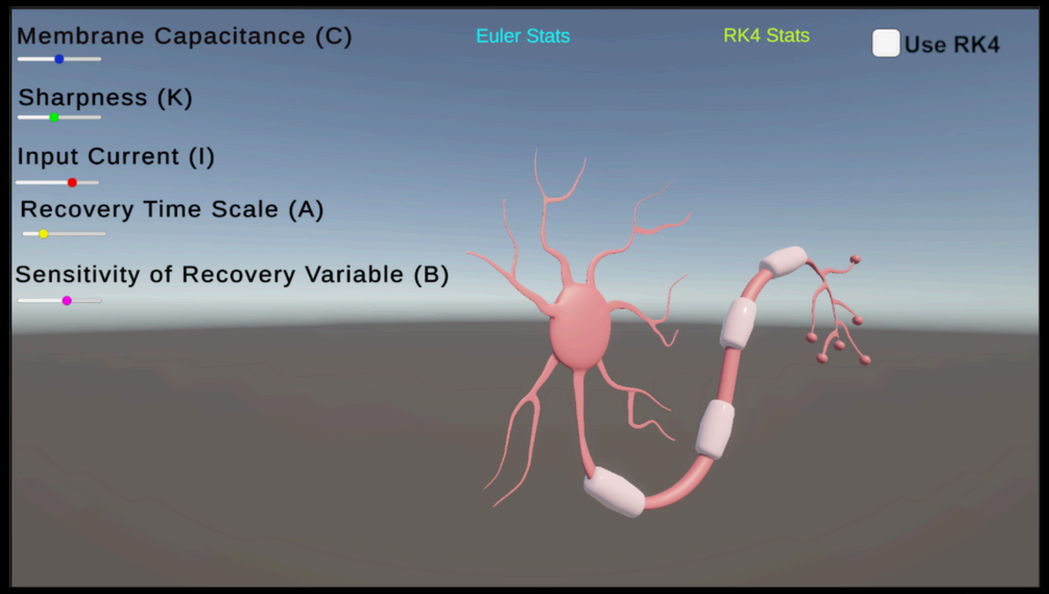
Solutions

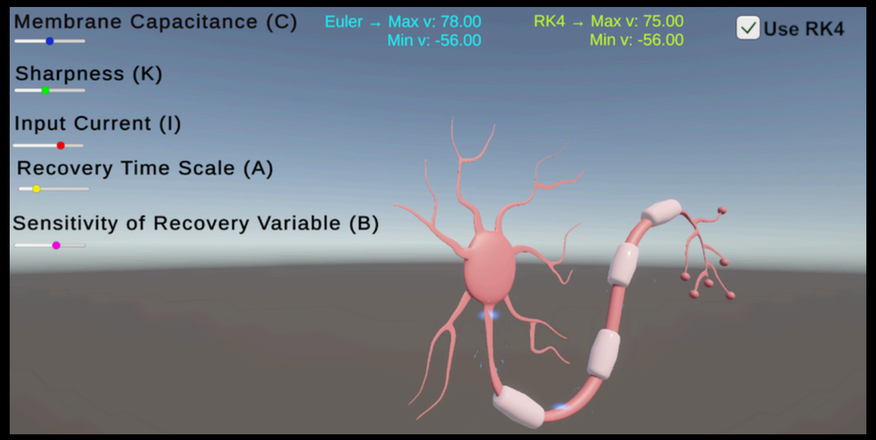
1. Imported an effect package from the asset store.
2. Used empty objects along the path and connected them to the script.
3. Used some of Unity's UI components to determine what method to use.

Results

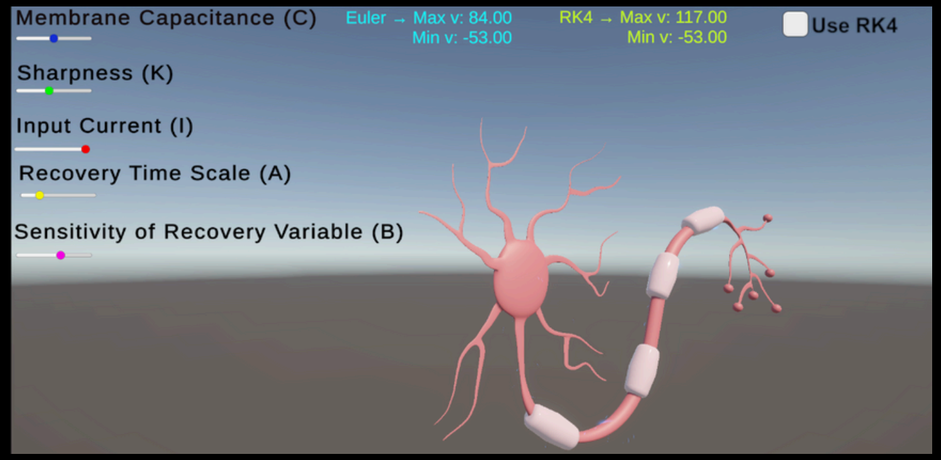
- A neuron model that spikes based on the value of the sliders to show instant effect.
- Change the value of the sliders to see the instant impact on the neuron spiking rate.
- Choose the solution method to be Euler or RK4 to see the difference in the solution based on the method used.

Program GUI

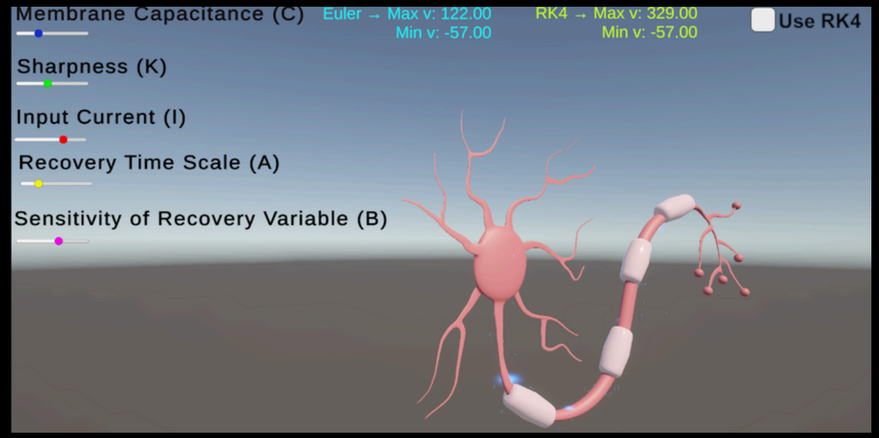




Normal Conditions Results



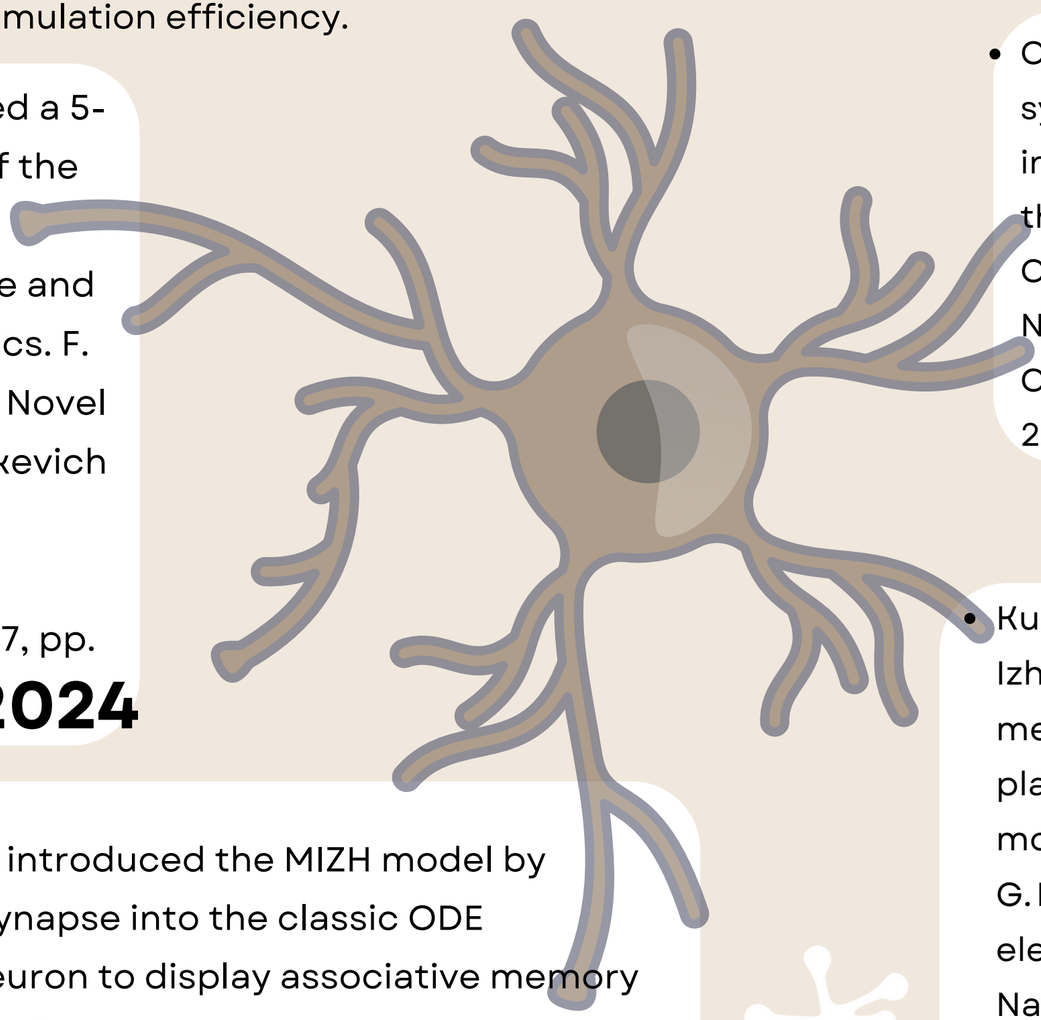
Maximum Input Current



Low Membrane Capacitance

Survey

- Recent advancements in computational neuroscience have led to multiple extensions of the Izhikevich neuron model to improve biological realism, adaptability, and simulation efficiency.



- Jia et al. (2024) proposed a 5-dimensional extension of the Izhikevich model incorporating memristive and electromagnetic dynamics. F. Jia, L. Zhou, and Y. Liu, “A Novel Coupled Memristive Izhikevich Neuron Model and Its Complex Dynamics,” Mathematics, vol. 12, no. 7, pp. 1021–1034, July 2024..

2024

- Chen & Campbell (2023) investigated the influence of synaptic transmission delays in Izhikevich networks, incorporating delay differential equations. This enabled the study of large-scale synchronization and bursting. Z. Chen and S. Campbell, “Synaptic Delays in Izhikevich Networks and Their Mean-Field Dynamics,” Journal of Computational Neuroscience, vol. 54, no. 3, pp. 211–228, 2023.

2023

- Kuzum, Gally & Qiao (2021) enhanced the Izhikevich neuron model by integrating memristor-based spike-timing-dependent plasticity (STDP), mimicking synaptic weight modulation in hardware-inspired neural circuits. G. Kuzum, W. Gally, and Q. Qiao, “Neuro-inspired electronics: Memristors in synaptic computing,” Nature Electronics, vol. 4, no. 1, pp. 31–42, Jan. 2021.

2021

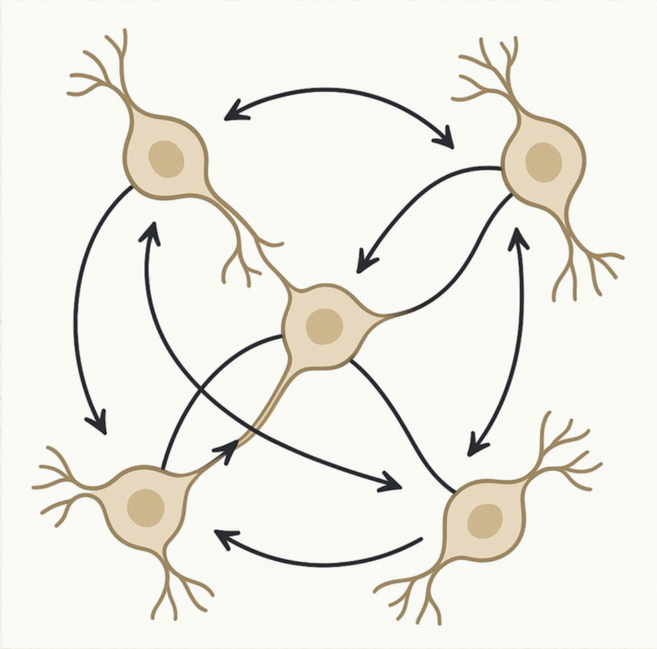
- Fang, Duan & Wang (2022) introduced the MIZH model by integrating a memristive synapse into the classic ODE system,This allowed the neuron to display associative memory and complex bursting behavior. L. Fang, L. Duan, and M. Wang, “Memristive Izhikevich Spiking Neuron for Oscillatory Associative Memory,” Frontiers in Neuroscience, vol. 16, pp. 1–11, 2022.

2022

improvements & Future work

Model Improvements:

- Extend the simulation to networks of interconnected neurons,where each neuron influences and is influenced by others through synaptic coupling,We simulate these interactions by modifying the input current I_i of each neuron to include signals from other neurons:
- $I_i = I_{i\_ext} + \Sigma_j g_{ij} \cdot S(v_j)$



- Introduce adaptive step-size control(Make small steps During spikes" to maintain accuracy and Take larger steps During rest to improve speed) , At each step, take two estimates of the next value:

1. One with a full step
2. One with two half steps

- Incorporate stochastic elements (e.g. noise, random spiking) to simulate realistic brain behavior:

1- Add Noise to the Input Current (I),Instead of a fixed current I

2- Use probabilistic spiking behavior

3- Simulate random synaptic events (e.g., Poisson-distributed spikes)

improvements & Future work

MACHINE LEARNING ENHANCEMENTS:

- Improve PINN stability and convergence using better architectures or loss weighting
- Use transfer learning for faster training on similar neuron models
- Use the PINN model to infer optimal neuron parameters from known V, IN, and W based on the dynamic neuron ODEs.

NUMERICAL METHODS :

- Use a higher-order solver such as the 4th-order Runge-Kutta method($O(h^4)$) to improve accuracy and capture sharper spike dynamics in neural simulations.

Our team hierarchy

Board

AE

MB

Ai team

Unity & Blender Team

&

AA

AA

KM

Numerical Methods Team

AS

OG

KH

Report & presentation Team

BA

EW

OA

REFERENCES

[1] A. L. Hodgkin and A. F. Huxley, “A quantitative description of mem-
brane current and its application to conduction and excitation in nerve,”
The Journal of Physiology, vol. 117, no. 4, pp. 500–544, 1952.

[2] E. M. Izhikevich, “Simple model of spiking neurons,” IEEE Transactions
on Neural Networks, vol. 14, no. 6, pp. 1569–1572, 2003.

[3] E. Schneider and J. Schultheis, “Fitzhugh–nagumo model,” Schol-
arpedia, http://www.scholarpedia.org/article/FitzHugh-Nagumo model,
2008.

[4] W. E. Schiesser, Differential Equation Analysis in Biomedical Science
and Engineering: Ordinary Differential Equation Applications with R.
John Wiley & Sons, 2014.

[5] E. Fehlberg, “Low-order classical runge-kutta formulas with step
size control and their application to some heat transfer problems,”
NASA Technical Report R-315, Tech. Rep., 1969. [Online]. Available:
https://maths.cnam.fr/IMG/pdf/RungeKuttaFehlbergProof.pdf

[6] E. Hairer, S. P. Nørsett, and G. Wanner, Solving Ordinary Differential
Equations I: Nonstiff Problems. Springer-Verlag, 1993.

[7] P. J. van der Houwen and B. P. Sommeijer, “Stabilized explicit methods
for stiff odes: applications to circuit simulation,” Journal of Computa-
tional and Applied Mathematics, vol. 6, no. 1, pp. 77–86, 1980.

[8] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics-informed
neural networks: A deep learning framework for solving forward and
inverse problems involving nonlinear partial differential equations,”
Journal of Computational Physics, vol. 378, pp. 686–707, 2019.

[9] N. Thuerey, P. Holl, X. Hu, T. Pfaff, K. Um, and S. Wiewel, “Physics-
based deep learning,” arXiv preprint arXiv:2109.05237, 2021.

[10] R. Kazmi et al., “Using unity for immersive biomedical visualization,”
IEEE Computer Graphics and Applications, vol. 41, no. 5, pp. 78–87,
2021.

Thank you

OUR CONTACTS

Emails : Team2 Contacts