Computational Biomaterials and Biomechanics - Exercise

Robin Steiner (11778873)

December 19, 2023

1 ODE solving & RC circuit

1.1 ODE solving

1.1.1

The smaller the step size the better both Euler methods approximate the analytical solution

For $tDt > 0.05$ the forward Euler method becomes instable in this case.

The instability of the forward Euler method at high step sizes stems from the fact that this method can over/under-shoot the solution. If this happens it will do the opposite on the next step, resulting in an oscilation around the actual solution. In general this overshoot can appear, because the method assumes a constant derivative along the time step, which is not the case, this mistake becomes more problematic for large time steps

1.1.2

The following code solves and plots the ODE $y' = -20y$ using the built-in solver 'RK45'.

```
def odefun(t, y): return -20 * y
sol = integrate.solve_ivp(odefun, [0, 1], [y0], method='RK45')

...
ax.plot(sol.t,sol.y[0],label='RK45')
...

```

The resulting plot can be seen in Figure **??**.

1.1.3

Next the following ODE is solved using RK45 and the backward Euler method. Again those results are compared to the analytical solution which is also found using Python.

$$y'(t) = -3y(t) + 9t, \quad y(0) = 9$$

The analytical solution as well as the forward Euler approximation are given by:

$$y(t) = 3e^{-3t} + 3t$$

and:

$$y_{n+1} = \frac{y_n + 9 \cdot t_{n+1} \cdot \text{timeStep}[n+1]}{1 + 3 \cdot \text{timeStep}[n+1]}$$

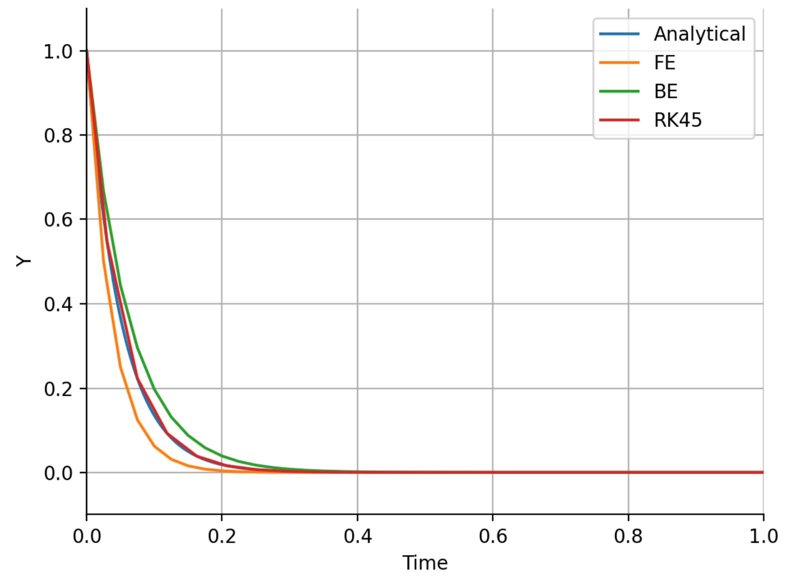

Figure 1: Plot of the solution for $y' = -20y$ using a forward and backward Euler method as well as 'RK45'. Additionally, it shows the analytical solution

The code is given by:

```
### ------------- ANALYTICAL SOLUTION -------------
t_sym = symbols('t')
y_sym = Function('y')
ode = Eq(y_sym(t_sym).diff(t_sym), -3*y_sym(t_sym) + 9*t_sym)
analytical_solution = dsolve(ode, y_sym(t_sym), ics={y_sym(0): y0})

### ------------- BACKWARD EULER ---------
yBEVec = np.zeros(timeSteps)  # Allocate memory
yBEVec[0] = y0  # Set initial condition
# Loop over time
for t in range(timeSteps-1):
    yBEVec[t+1] = (yBEVec[t] + 9*timeStep[t+1]*tDt) / (1 + 3*tDt)

### ------------- RK45 SOLVER ---------
def odefun(t, y): return -3 * y + 9 * t
sol = integrate.solve_ivp(odefun, [0, tStop], [y0], method='RK45')

```

The resulting plots are shown in Figure **??** and **??**.

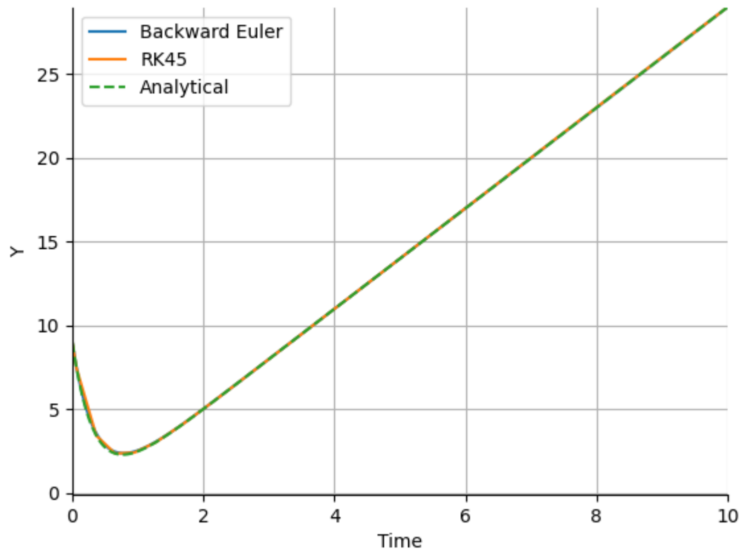

Figure 2: Plot of the solution for $y'(t) = -3y(t) + 9t$ using a backward Euler method as well as 'RK45'. Additionally, it shows the analytical solution

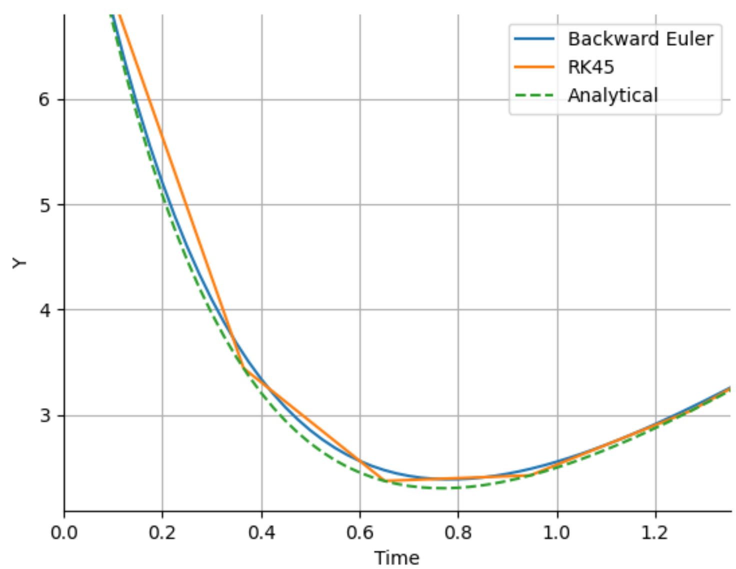

Figure 3: Zoomed in plot of the solution for $y'(t) = -3y(t) + 9t$ using a forward and backward Euler method as well as 'RK45'. Additionally, it shows the analytical solution

1.2 RC circuit / passive neuron

1.2.1

Looking at Figure **??** - **??** we can deduce the following:

- Increasing resistance (R) will lead to a higher maximal voltage.
- Increasing capacitance (C) will result in a slower charging and discharging of the capacitor.

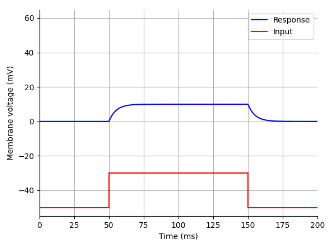

Figure 4: Response Plot for: $R = 10M\Omega, C = 0.5nF$

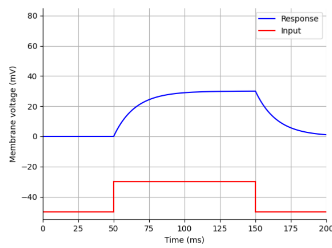

Figure 5: Response Plot for: $R = 30M\Omega, C = 0.5nF$

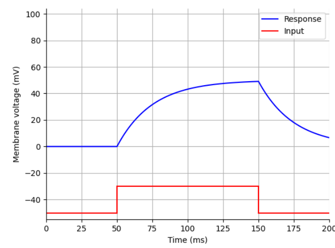

Figure 6: Response Plot for: $R = 50M\Omega, C = 0.5nF$

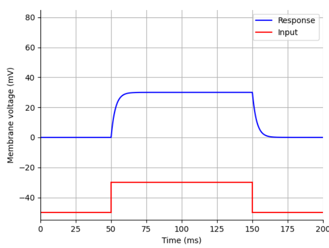

Figure 7: Response Plot for: $R = 30M\Omega, C = 0.1nF$

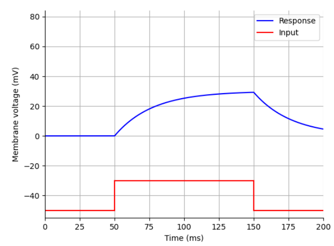

Figure 8: Response Plot for: $R = 30M\Omega, C = 0.9nF$

The time constant (τ) of an RC circuit is a measure of how quickly the circuit responds to changes. It is given by the product of resistance (R) and capacitance (C), i.e., $\tau = R * C$. More specifically it is the time it takes to charge the capacitor 63.2% of its final voltage in response to a step change in voltage.
To find the maximum voltage in response to a current step input, we can set $\frac{dV}{dt} = 0$, giving us the equation:

$$0 = -\frac{v}{R \cdot C} + \frac{I_{\text{Stim}}}{C}$$

Solving this for V gives us V_{max} as:

$$V_{max} = I_{\text{Stim}} \cdot R$$

The correctness of this can be easily checked by looking at the plots from Figure **??** - **??** (where $I_{Stim} = 1$)

Next we calculate how long it takes until the capacitor is charged up to 99% using. The voltage across a charging capacitor in an RC circuit is given by:

$$v(t) = V_0 \left(1 - e^{-\frac{t}{R \cdot C}}\right)$$

we can set $v(t)$ equal to that percentage of the final value (V_0) and solve for t:

$$0.99 \cdot V_0 = V_0 \left(1 - e^{-\frac{t}{R \cdot C}}\right)$$

$$0.99 = 1 - e^{-\frac{t}{R \cdot C}}$$

$$-\frac{t}{R \cdot C} = \ln(0.01)$$

Solve for t:

$$t = -R \cdot C \cdot \ln(0.01) \approx 4.605 \cdot R \cdot C$$

Therefore we can use factor 5 to approximate the time it takes for a capacitor to be fully charged up:

$$t \approx 5 \cdot R \cdot C$$

1.2.2

The following code shows the implementation of the RK45 solver with the provided code as a base. The result is plotted together with the results of a forward and backward Euler solver and can be seen in Figure **??**. The time step for the forward and backward Euler solver was set to 5ms to illustrate the differences.

Additionally, Figure **??** shows the dynamic time steps used by the RK45 solver.

```
...
### ------------- PARAMETERS ---------
solvers = ['BE', 'FE', 'RK45']
showTimeSteps = False
...
### ------------- SOLVING FOR ALL SOLVERS ---------
for solver in solvers:
    if solver != 'FE' and solver != 'BE':
        # Use scipy's solve_ivp to solve the ODE system for the built-in solver
        sol = solve_ivp(
            lambda t, v: ode_system(t, v, I, R, C, tDel, tDur, tDt, solver),
            [0, tStop],
            [0],  # Initial condition
            method=solver
        )
        # Extract the solution
        vVec_solvers[solver] = sol.y[0]
        timeSteps_solvers[solver] = sol.t
    else:
        # Solve the ODE system for Forward Euler and Backward Euler
        vVec = np.zeros(timeSteps)
        for t in range(0, timeSteps - 1):
            IStim = 0
            if t >= int(tDel / tDt) and t < int((tDel + tDur) / tDt):
                IStim = I  # in nA

            if solver == 'FE':
                vVec[t + 1] = vVec[t] + ((-vVec[t] / R + IStim) / C) * tDt
            elif solver == 'BE':
                vVec[t + 1] = (vVec[t] + IStim * (tDt / C)) / (1 + tDt / (R * C))

        vVec_solvers[solver] = vVec
        timeSteps_solvers[solver] = timeStep
...
### ------------- PLOTTING ---------
for solver in solvers:
```

```
            plot, = ax.plot(timeSteps_solvers[solver], vVec_solvers[solver], label=solver+' Nr. Timesteps='+str(timeSteps_solvers[solver].size))
            if showTimeSteps:
                for t_step in timeSteps_solvers[solver]:
                    ax.axvline(x=t_step, color=plot.get_color(), linestyle='--', linewidth=0.8)
    ...
    ### ------------- ODE ---------
    def ode_system(t, v, I, R, C, tDel, tDur, tDt, solver):
        # Stimulus current
        IStim = 0
        if t >= tDel and t < tDel + tDur:
            IStim = I  # in nA

        # Compute change of v
        if solver == 'BE':
            return (v + IStim * tDt / C) / (1 + tDt / (R * C))
        else:
            return (-v / R + IStim) / C
    ...
```

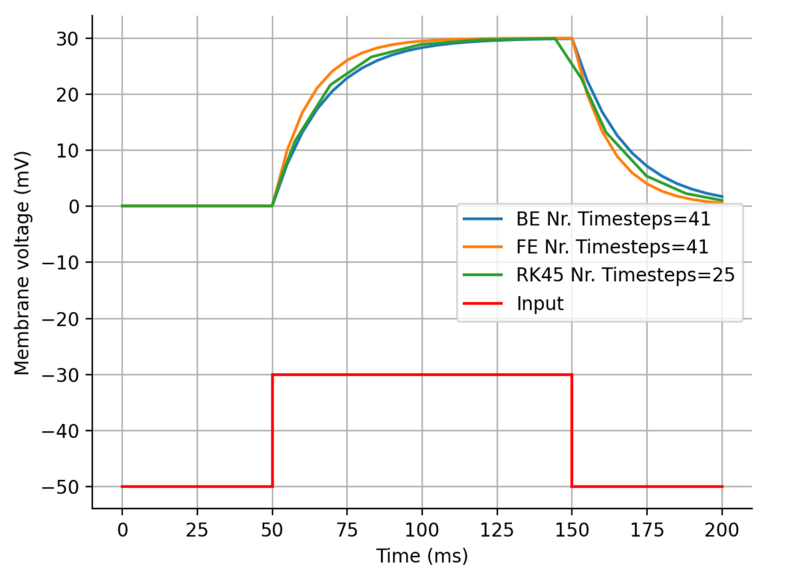

Figure 9: Plot of the response of an RC element calculated using a forward and backward Euler method as well as 'RK45'

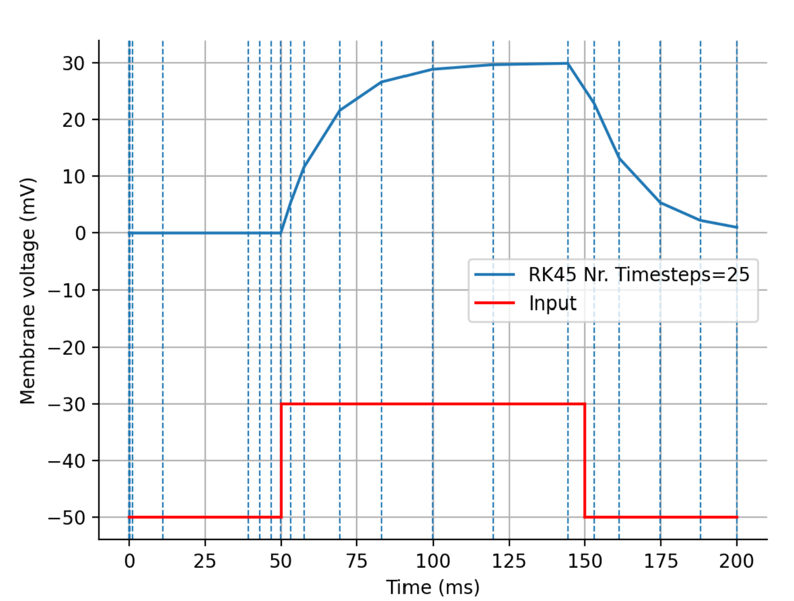

Figure 10: Plot of the response of an RC element calculated 'RK45' method. The colored vertical Lines indicate the time steps made for the calculation.

1.2.3

Using the following parameters generates the resulting plot shown in Figure ??. It gives a comparison between all the different available solvers.

```
        solvers = ['RK45', 'RK23', 'DOP853', 'Radau', 'BDF', 'LSODA']

```

Additionally, Figure ?? shows all the dynamic time steps of those solvers (color-coded). The number of steps are shown in the legend.

As we can see many solvers perform better in this case than the RK45 solver. The issue with the solution of RK45 is, that the dip happens too early (before the dip of the input signal).

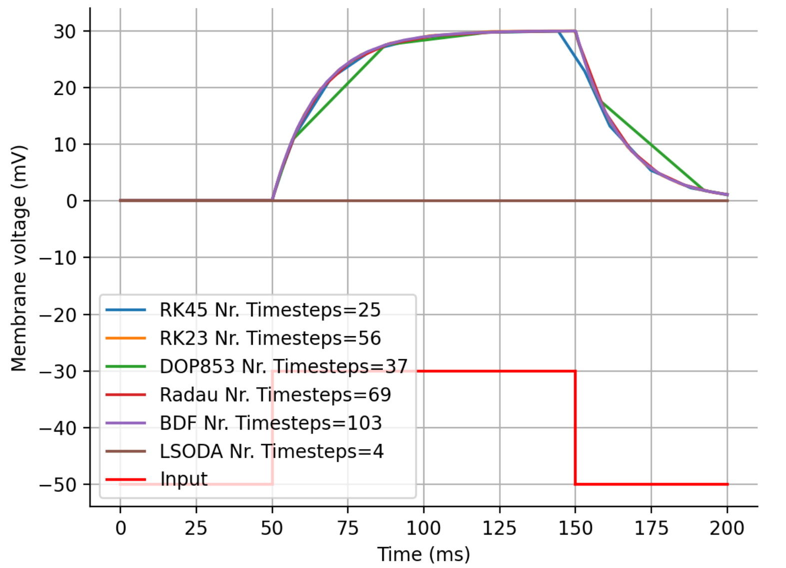

Figure 11: Plot of the response of an RC element calculated using a forward and backward Euler method as well as 'RK45'

Figure ?? shows a comparison between RK45 and RK23. In this specific case the RK23 solver actually performs better than the RK45 solver, since its closer to the real solution. It also uses more time steps though.

A second example of a better-performing solver is the Radau solver shown in Figure ??. It is intended to be used with stiff ODEs which our ODE can be classified as, at least to a degree. Radau however also uses almost three times the time steps compared to RK45.

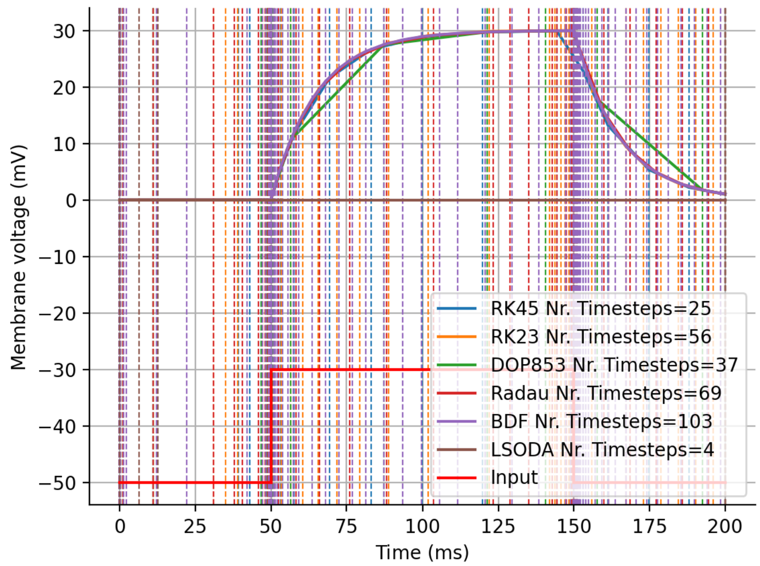

Figure 12: Plot of the response of an RC element calculated 'RK45' method. The colored vertical Lines indicate the time steps made for the calculation.

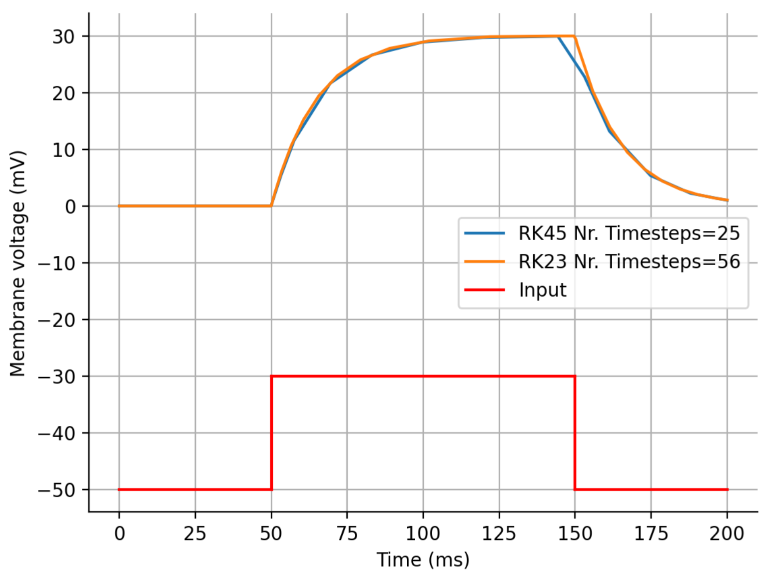

Figure 13: Plot of the response of an RC element calculated 'RK45' method. The colored vertical Lines indicate the time steps made for the calculation.

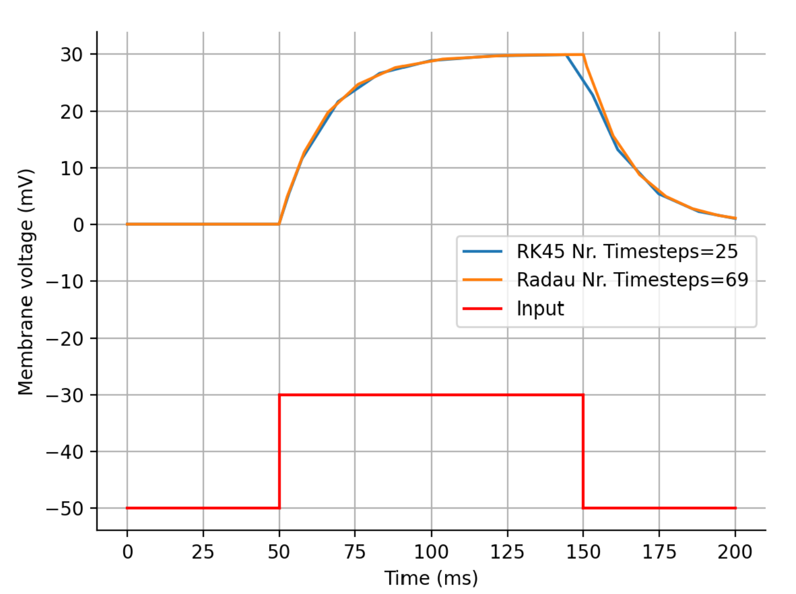

Figure 14: Plot of the response of an RC element calculated 'RK45' method. The colored vertical Lines indicate the time steps made for the calculation.

1.2.4

We can also try to 'tune' the RK45 solver in order to get better results. For this we can for example use the relative and absolute tolerances (r_{tol} and a_{tol}).
Figure **??** - **??** show a comparison between the results of the untuned and the tuned version of the RK45 solver with $r_{tol} = 5 \cdot 10^{-5}$ and $a_{tol} = 5 \cdot 10^{-7}$

As we can see, we can eliminate the early dip using those parameter values while still keeping a low amount of time steps. The lower we choose those parameter values the more time steps we will have and the more accurate are result will become.

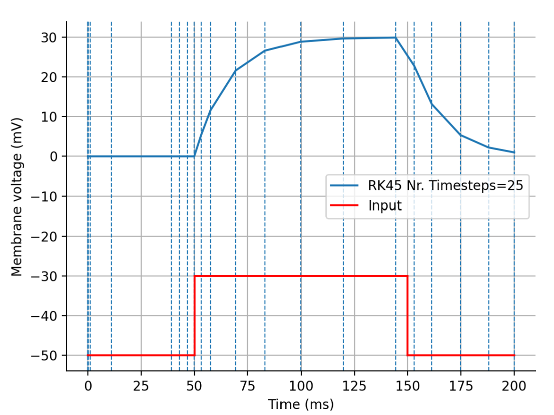

Figure 15: Response Plot of the RK45 solver with default parameters

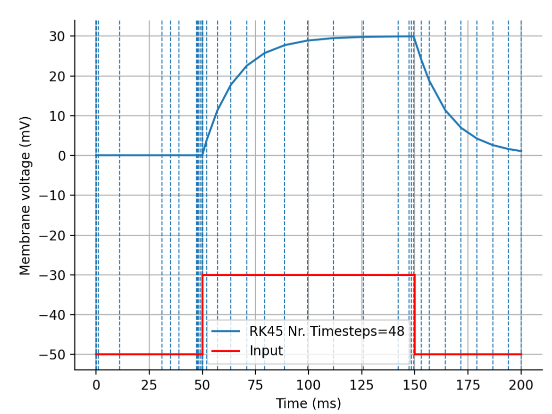

Figure 16: Response Plot of the RK45 solver with $r_{tol} = 5 \cdot 10^{-5}$ and $a_{tol} = 5 \cdot 10^{-7}$

2 Hodgkin-Huxley Model (single)

2.1

When varying the amplitude of the applied stimulus (I) for a 0.5 ms long pulse, we find, that the threshold current is at around $13.45uA/cm^2$ The resulting action potential can be seen in Figure **??**.
When the stimulus amplitude is doubled ($I = 26.9\ \mu A/cm^2$), the action potential is initiated earlier. The resulting plot can be seen in Figure **??**

2.2

As we can see in Figure **??** and **??**, when increasing the time step to $tDt = 0.1ms$ and $I = 20\mu A/cm^2$ we still get a good solution with the Backward Euler Solver. The Forward Euler however produces immense oscillations, indicating that the time step is to big for this solver. The reason why this happens was discussed in a previous section.

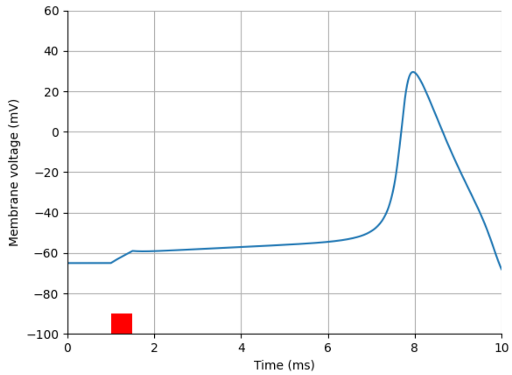

Figure 17: Membrane voltage over time for the spiking threshhold.

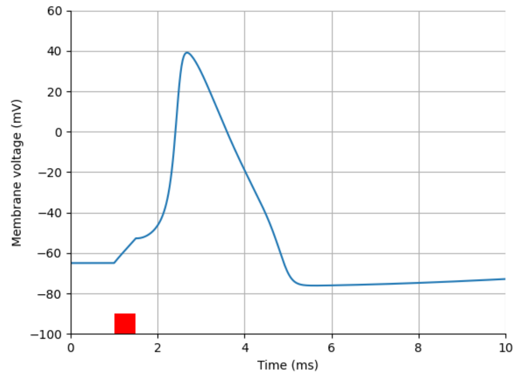

Figure 18: Membrane voltage over time for double the spiking threshhold.

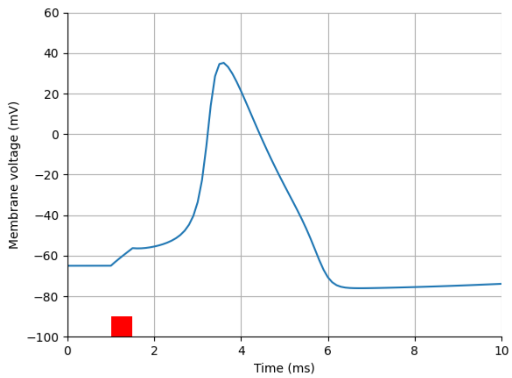

Figure 19: Action potential response solved with the Backward Euler Method

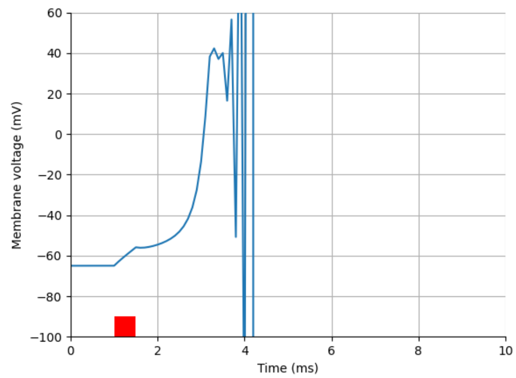

Figure 20: Action potential response solved with the Forward Euler Method

2.3

Figure ?? illustrates the behavior of the membrane voltage and the sodium and potassium current densities during the simulation. The code that produced this plot it given by:

```
        fig, ax1 = plt.subplots()
        ax1.grid()
        ax1.set_xlabel('Time (ms)')
        ax1.set_ylabel('Membrane voltage (mV)', color='tab:blue')
        ax1.plot(timeStep, vVec, label='Membrane Voltage', color='tab:blue')
        ax1.tick_params(axis='y', labelcolor='tab:blue')

        ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis
        ax2.set_ylabel('Current densities (uA/cm2)', color='tab:red')

        # Plot sodium and potassium current densities with legends
        ax2.plot(timeStep, gNa * mVec**3 * hVec * (vVec - eNa), '--', label='Sodium Current (iNa)', color='tab:red')
        ax2.plot(timeStep, gK * nVec**4 * (vVec - eK), '-.', label='Potassium Current (iK)', color='tab:green')

        ax2.tick_params(axis='y', labelcolor='tab:red')

        # Add legend
        lines, labels = ax1.get_legend_handles_labels()
        lines2, labels2 = ax2.get_legend_handles_labels()
        ax2.legend(lines + lines2, labels + labels2, loc='upper right')

        fig.tight_layout()  # ensure the shared x-axis labels are not slightly cut off
        plt.show()
```

The rise in the membrane voltage during the initial phase of the simulation is caused by the influx of sodium ions through voltage-gated sodium channels. This depolarization phase is a result of the activation of sodium channels (governed by variables m and h) and the subsequent increase in sodium current density. After reaching a peak, the membrane voltage starts to decline due to the inactivation of sodium channels and the activation of potassium channels. The potassium current density increases, leading to repolarization and the restoration of the resting membrane voltage.

2.4

As can be seen in Figure ?? and ?? higher temperatures generally lead to shorter action potential durations (widths) and decreased action potential amplitudes (heights). The peak tends to appear earlier for higher temperatures.
For temperatures above 15C we can no longer observe an action potential. This also means that with the previously chosen parameters we do not get am action potential at 37C. Still it has been shown many times be researcher, that the Hodgkin-Huxley model can be used for humans. For

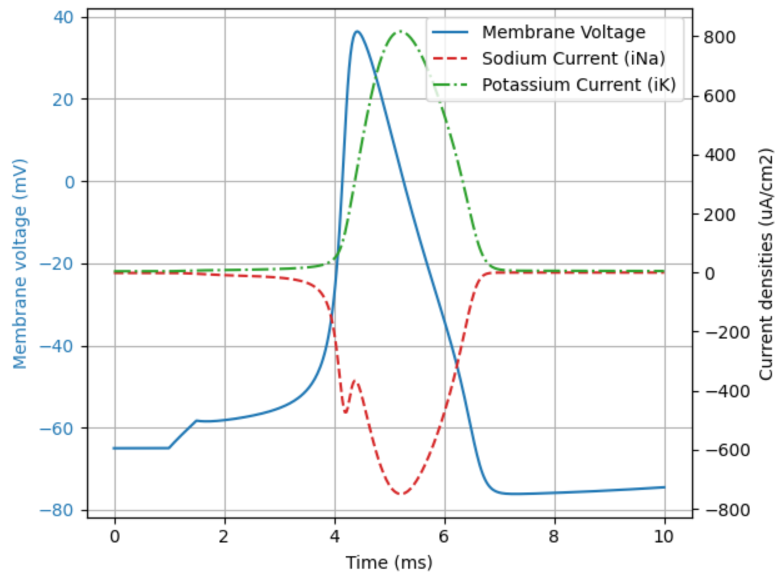

Figure 21: Membrane voltage and current densities over time. ($I = 15\mu A/cm^2$, $tDur = 0.5ms$)

this we would have to adapt our parameters accordingly, though.

2.5

The sodium current density (i_{Na}) in the Hodgkin & Huxley model is given by the equation:

$$i_{Na} = g_{Na} \cdot m^3 \cdot h \cdot (v - e_{Na})$$

where:

$$\begin{aligned} g_{Na} &: \text{Sodium channel maximum conductivity} \\ m &: \text{Activation gating variable} \\ h &: \text{Inactivation gating variable} \\ v &: \text{Membrane voltage} \\ e_{Na} &: \text{Sodium reversal/equilibrium potential} \end{aligned}$$

The conditions under which the sodium flux reverses its direction, leading to a repolarizing effect, occur when the membrane voltage exceeds the sodium reversal potential (e_{Na}).

The activation gating variable m responds to an increase in membrane voltage by increasing its value. The equation governing m in the model is:

$$\frac{dm}{dt} = \alpha_m \cdot (1 - m) - \beta_m \cdot m$$

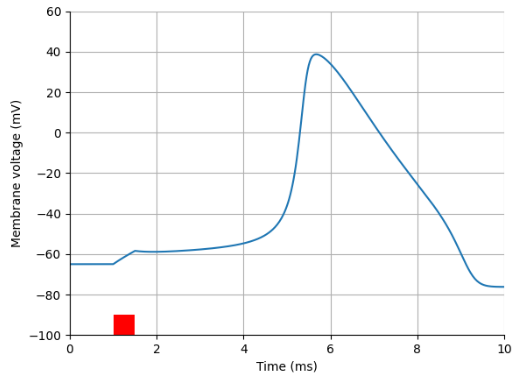

Figure 22: Action Potential at 2°C

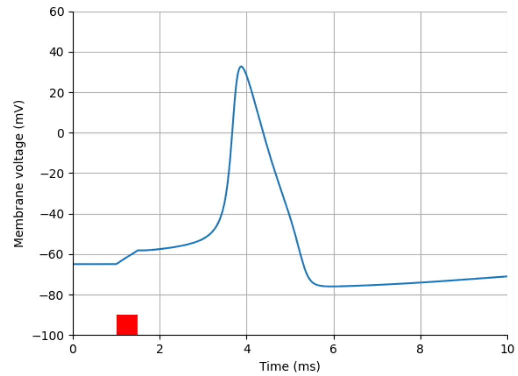

Figure 23: Action Potential at 10°C

where:

$$\alpha_m : \text{Rate of activation}$$
$$\beta_m : \text{Rate of deactivation}$$

An increase in membrane voltage generally leads to an increased activation of sodium channels, resulting in an increase in m and a higher probability of sodium channels being open.

The inactivation gating variable h responds to an increase in membrane voltage by decreasing its value. The equation governing h in the model is:

$$\frac{dh}{dt} = \alpha_h \cdot (1 - h) - \beta_h \cdot h$$

where:

$$\alpha_h : \text{Rate of inactivation}$$
$$\beta_h : \text{Rate of deinactivation}$$

An increase in membrane voltage typically leads to a decrease in the inactivation of sodium channels, resulting in a decrease in h and a lower probability of sodium channels being inactivated.

2.6

The following Python script computes the Strength-Duration (SD) curve for the single-compartment Hodgkin & Huxley model. The script uses a binary search algorithm to efficiently find the threshold for different pulse durations.

```
    def binary_search_threshold(time_step, stim_duration,
pulse_amplitude):
        low, high = 0, 1000  # Initial search range for threshold
        threshold = None
```

```python

    while high - low > 1e-6:
        current_amplitude = (low + high) / 2
        _, v, _, _, _ = HH_single(I=current_amplitude, tDur=stim_duration)
        max_voltage = np.max(v)

        if max_voltage > 0:
            high = current_amplitude
            threshold = current_amplitude
        else:
            low = current_amplitude

    return threshold
...

# Parameters
pulse_durations = np.logspace(-2, 2, 20)  # Pulse durations from 0.01 to 100 ms in 20 logarithmic steps
stim_duration_long_pulse = 100  # Duration of the long pulse for rheobase calculation

# Compute SD curve
thresholds = []

for duration in pulse_durations:
    threshold = binary_search_threshold(0.025, duration, 15)
    thresholds.append(threshold)
...

rheobase = binary_search_threshold(0.025, stim_duration_long_pulse, 15)
double_rheobase = 2 * rheobase
index_double_rheobase = np.argmin(np.abs(np.array(thresholds) - double_rheobase))
chronaxie = pulse_durations[index_double_rheobase]

```

From the computed SD curve we get the following values for the Rheobase and Chronaxie:

- Rheobase: 2.22 uA/cm^2
- Chronaxie: 2.07 ms

2.7

The following Python script computes the spiking probability for amplitudes between 0 and 30 $\mu A/cm^2$. For each amplitude, the simulation is run 50 times, and a logistic curve is fitted to the simulated data.

```python
def logistic_function(x, L, k, x0):
```

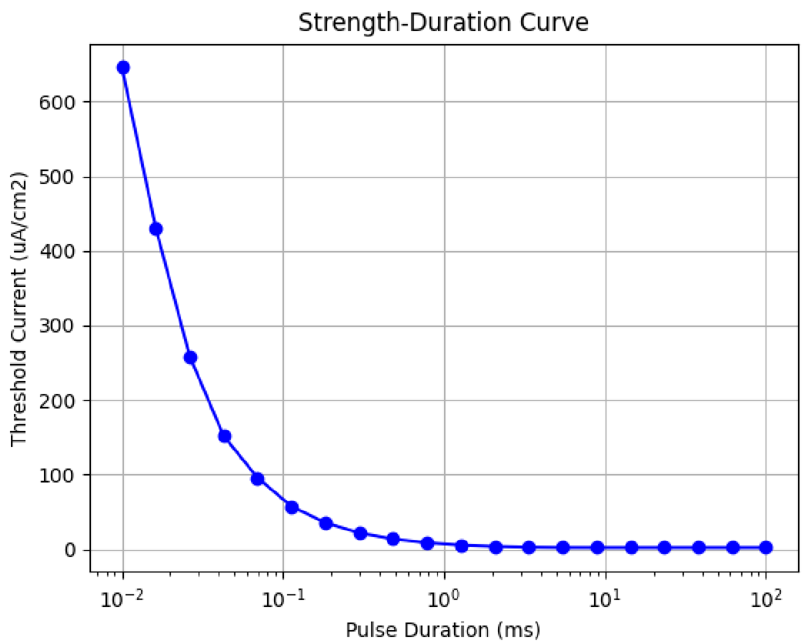

Figure 24: Strength-Duration Curve for the Hodgkin & Huxley Model.

```
        return L / (1 + np.exp(-k * (x - x0)))

    # Parameters
    amplitudes = np.arange(0, 30, 1)  # Stimulus amplitudes from 0 to 30 A/cm in 1 A/cm steps
    num_simulations = 50
    tDur = 0.5

    # Function to run simulation and compute spiking probability
    def run_simulation_and_compute_probability(amplitude):
        spiking_count = 0

        for _ in range(num_simulations):
            _, v, _, _, _ = HH_single(I=amplitude, tDur=tDur)
            if np.max(v) > 0:
                spiking_count += 1

        spiking_probability = spiking_count / num_simulations * 100
        return spiking_probability

    # Compute spiking probabilities
    spiking_probabilities = [run_simulation_and_compute_probability(amplitude) for amplitude in amplitudes]
```

```python

        # Fit a logistic curve to the spiking probabilities
        params, _ = curve_fit(logistic_function, amplitudes,
    spiking_probabilities, p0=[100, 1, 15])

```

Additionally, a random factor is added to the total ionic current:

```python
        iIon = iNa+iK+iL + 20 * np.random.randn() # in uA/cm2

```

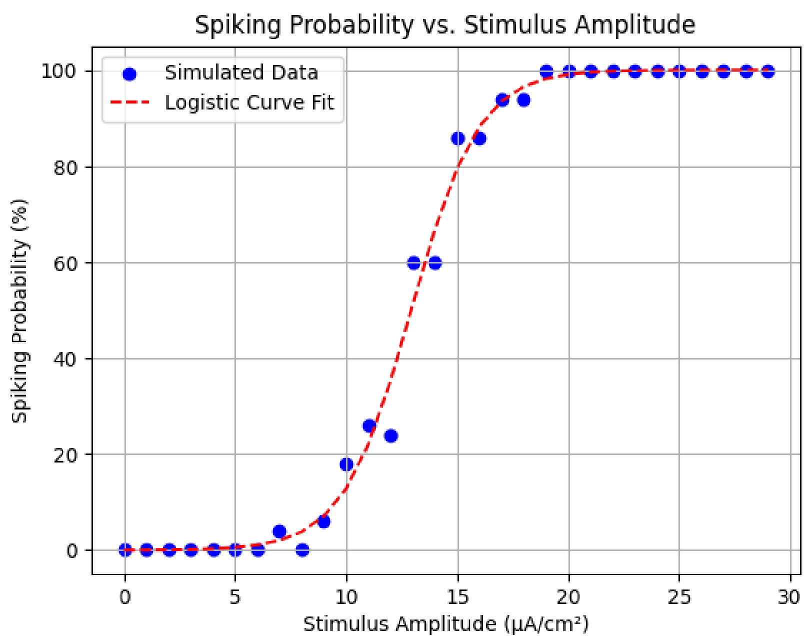

Figure 25: Spiking Probability vs. Stimulus Amplitude with Logistic Curve Fit.

The resulting plot can be seen in Figure **??** and the parameters of the fitted logistic curve are:

- Parameter L: 100.18
- Parameter k: 0.66
- Parameter x_0: 12.94

3 Multi-Compartment Hodgkin-Huxley Model

]
In this study, we applied an extracellular cathodic subthreshold pulse to a Hodgkin-Huxley multi-compartment model. The goal was to analyze the membrane voltage distribution along the fiber, particularly at the center and adjacent compartments.

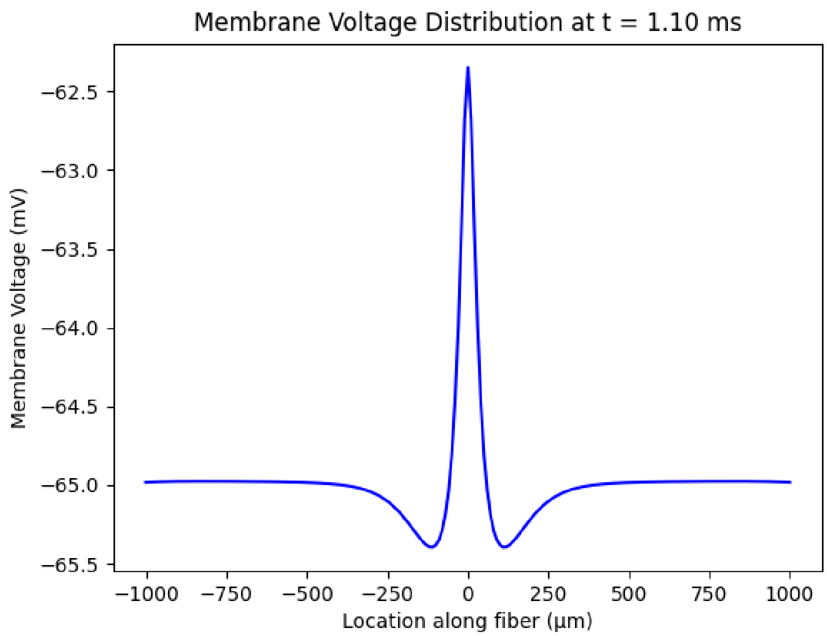

Figure 26: Spiking Probability vs. Stimulus Amplitude with Logistic Curve Fit.

The stimulation was set as extracellular with a cathodic subthreshold amplitude. The key modification in the script involved setting the `mode` parameter to `'extracellular'` and adjusting the `I` (stimulus amplitude) to a negative value representing the cathodic pulse.
The membrane voltage was plotted over the location of the fiber at a specific timestamp, which was chosen to be immediately after the stimulus onset. The resultant plot illustrates the voltage distribution along the fiber, indicating the effects of the applied subthreshold stimulation.
The observed phenomenon of opposite polarizations along the fiber, despite all compartments experiencing a negative V_e, can be explained by the passive electrical properties of the neuronal membrane. When a cathodic stimulus is applied, the area immediately beneath the electrode becomes hyperpolarized due to the influx of negative charges. This creates a local potential difference between the stimulated area and adjacent regions. Consequently, adjacent regions experience a relative depolarization, as positive charges flow towards the hyperpolarized area to balance the potential. This effect leads to opposite polarizations along the fiber: hyperpolarization directly under the electrode and depolarization in the neighboring compartments.
The study demonstrates how different regions of the fiber respond to a localized extracellular cathodic stimulus, with implications for understanding neuronal excitation and conduction phenomena. The observation of opposite polarizations provides insight into the complex interactions between electrical stimuli and the neuronal membrane's passive properties.