# NEURON Assignment

*Sarah Olesen, Tim Sit*

*October 14, 2018*

## Contents



# 1 Passive properties

## 1.1 Make a ball model and measure $R_i$, $m_\tau$ with current injection

We first define the function for creating a cell, injecting current, and recording voltage over time.

```
from neuron import h, gui
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
# default somalength = 100
# default soma diam = 500
def inject_current(cell, somaLength = 12.54, somaDiam = 12.54,
                  stimDur = 10, stimAmp = 0.05, stimDelay = 10, total_time = 25,
                  g_pas = 0.001,
                 printProperties = False):
    """
    Injects current to a cell and measure the voltage over time
    INPUT
    somaLength      | length of the soma (um)
    somaDiam        | diameter of the soma
    stimDur         | duration of the stimulus (ms)
```

```
    stimAmp          | amplitude of current injection (pA)
    stimDelay        | delay before the stimulus is given (ms)
    total_time       | total time of the simulation
    printProperties | if true, print the properties of the cell
    OUTPUT
    voltage
    time
    """

    cell.insert('pas')
    cell.L = somaLength
    cell.diam = somaDiam
    cell.g_pas = 0.001

    if printProperties is True:
        print('Cell properties')
        h.psection()

    # add point process: current injection
    stim = h.IClamp(cell(0.5))
    stim.delay = stimDelay
    stim.dur = stimDur # ms
    stim.amp = stimAmp # nA
    # measure voltage
    voltage = h.Vector()
    voltage.record(cell(0.5)._ref_v)

    # measure time
    time = h.Vector()
    time.record(h._ref_t)

    h.v_init = -70 # initial voltage
    h.load_file('stdrun.hoc')
    h.tstop = total_time
    h.run() # ms

    # time_np = time.as_numpy() # WARNING: don't output values with .as_numpy()
    # voltage_np = voltage.as_numpy()
    return voltage, time
```

We define a function to calculate the input resistance ($R_i$), from our cell recording using

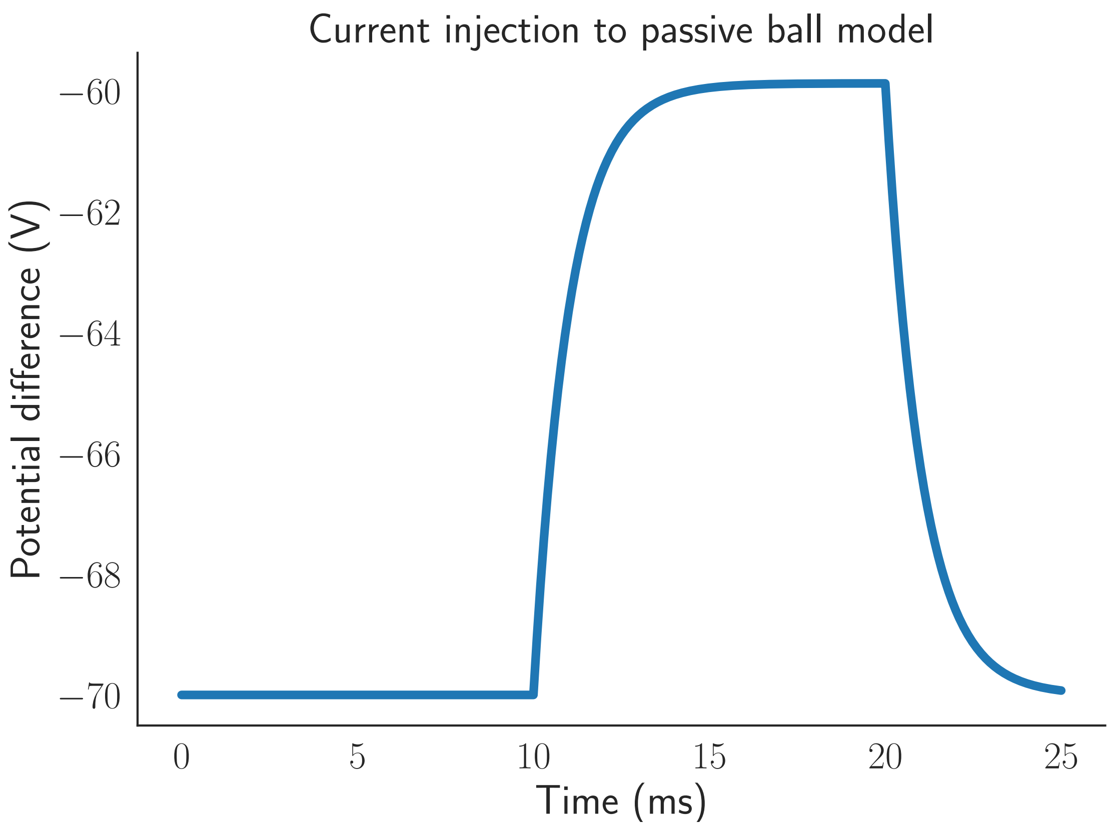
$$R_i = \frac{\Delta V}{\Delta I_e}$$

```
def cal_Ri(voltage_np, stimAmp = 0.05):

    delta_V = (max(voltage_np) - min(voltage_np)) * 10**(-3) # millivolts (mV)
    delta_Ie = stimAmp * 10**(-9) # nanoAmps
    input_resistance = delta_V / delta_Ie

    return input_resistance
```

We also define a function to calculate the membrane time constant $\tau_m$ from our recording. We are doing this by finding the time when the voltage has reached 63.2% of its peak voltage.

```
def cal_TauM(voltage, time, stimDelay = 10):
    # convert voltage to numpy
    voltage_np = voltage.as_numpy()
    time_np = time.as_numpy()
    # from "experiment"
    special_v = (max(voltage_np) - min(voltage_np)) * (1 - 1/np.e)
    special_vv = special_v + min(voltage_np)
    tauIndex = np.where(np.logical_and(voltage_np > special_vv - 0.05,
                                       voltage_np < special_vv + 0.05))
    if tauIndex[0].size > 1: # more than one matching value
        tau_m_samp = tauIndex[0][0]
    else:
        tau_m_samp = tauIndex[0]
    tau_m_exp = time_np[tau_m_samp]
    TauM = tau_m_exp - stimDelay

    return TauM
```

```
voltage, time = inject_current(cell = h.Section(name='soma'))
# plt.figure(figsize=(8, 4))
# ax = sns.lineplot(time, voltage)
plt.figure()
plt.plot(time, voltage)
plt.title('Current injection to passive ball model')
plt.xlabel('Time (ms)')
plt.ylabel('Potential difference (V)')
plt.show()
# ax.set_title('Current injection to passive ball model')
# ax.set(xlabel='Time (ms)', ylabel='Potential difference (V)')
```




```
input_resistance = cal_Ri(voltage)
# print('Input resistance is %.2f ohms' % input_resistance)
print('Input resistance is %.2f Mohms' % (input_resistance / 10**6))
```

```
## Input resistance is 202.41 Mohms
```

```
TauM = cal_TauM(voltage, time)
print('TauM is %.2f ms' % TauM)
```

```
## TauM is 1.00 ms
```

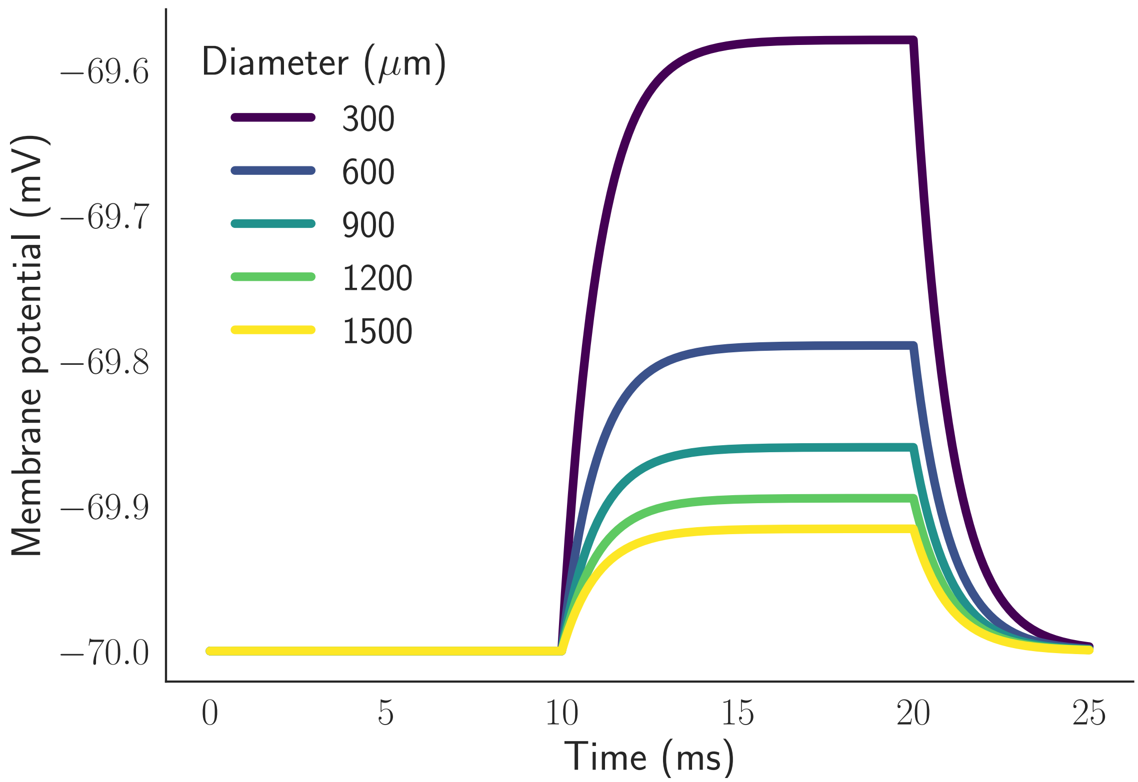
Due to the cells small size the input resistance is greater than we would normally expect. During our in vitro experiments this week we measured our input resistance to be around 5 Mohms.

### 1.2 Varying neuron variables

Change diameter, $R_m$, $R_a$, $C_m$ and see how it affects $R_i$ and $m_\tau$

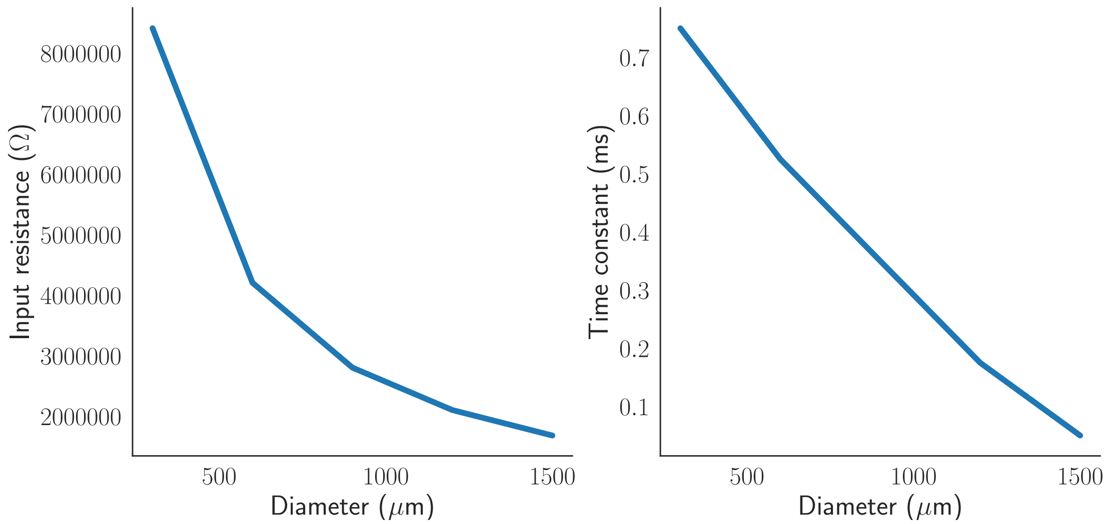
```
g_list = [0.001, 0.01, 0.1, 1, 10]
Ra_list = [10, 20, 30, 40, 50] # default is 35.4
diam_list = [300, 600, 900, 1200, 1500] # default is 500
Cm_list = [0.1, 0.3, 0.6, 1.0, 1.3] # default is 1
# circum =  soma.diam  * np.pi
# length = soma.L
```

```python
# A_list = (circum * 10**(-6)) * (length * 10 ** (-6))
input_resistance_list = list()
TauM_list = list()
# colormap
import matplotlib.pylab as pl
num_color = len(diam_list)
colors = pl.cm.viridis(np.linspace(0,1,num_color))
plt.figure()
for diam, n in zip(diam_list, np.arange(num_color)):
    voltage_np, time_np = inject_current(cell = h.Section(name='soma'),
                                         somaLength = 12.6157,
                                         somaDiam = diam,
                                         stimDur = 10,
                                         stimAmp = 0.05,
                                         stimDelay = 10)
    input_resistance_list.append(cal_Ri(voltage_np))
    TauM_list.append(cal_TauM(voltage_np, time_np))
    # plot voltage over time
    plt.plot(time_np, voltage_np,
             label = diam,
             color = colors[n])

plt.ylabel('Membrane potential (mV)')
plt.xlabel('Time (ms)')
plt.legend(frameon=False, title = 'Diameter ($\mu$m)')
plt.show()
```




### 1.2.1 Variation of input resistance and time constant with diameter

```
import seaborn as sns
aesthetics()
plt.figure()
plt.subplot(1, 2, 1)
ax = sns.lineplot(diam_list, input_resistance_list)
plt.xlabel('Diameter ($\mu$m)')
plt.ylabel('Input resistance ($\Omega$)')
plt.subplot(1, 2, 2)
# time constant vs. diameter
ax2 = sns.lineplot(diam_list, TauM_list)
plt.xlabel('Diameter ($\mu$m)')
plt.ylabel('Time constant (ms)')
plt.show()
```

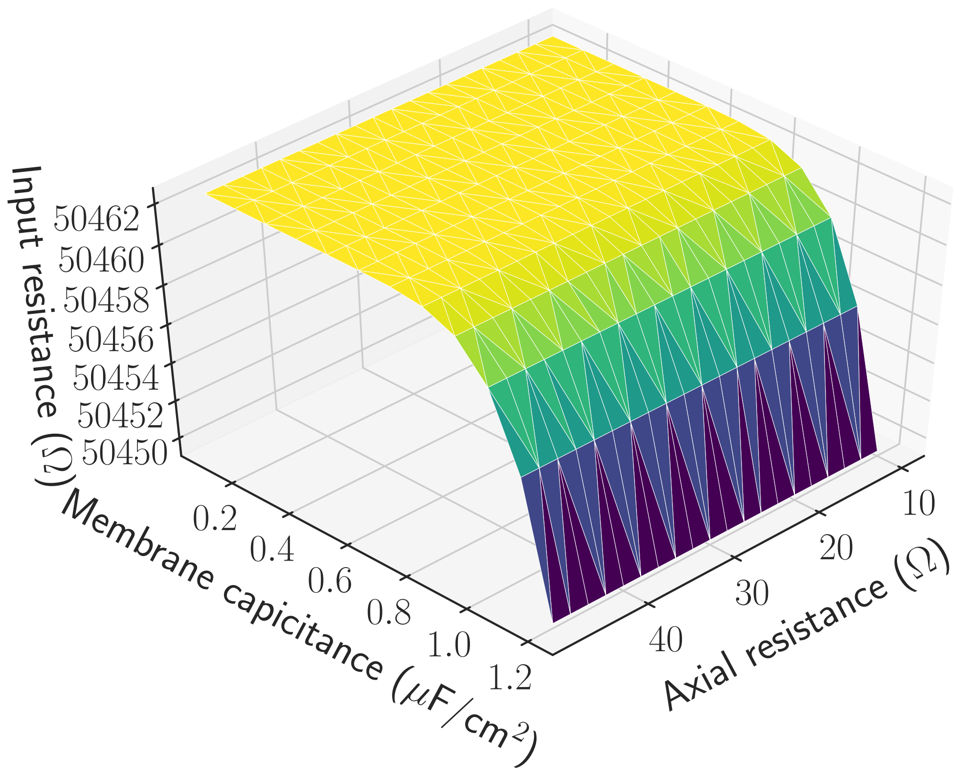



#### 1.2.2 Varying multiple variables simultaneously

```
import itertools
import pandas as pd
# variable lists
diam_list = [3, 6, 9, 12, 15]
Cm_list = [0.1, 0.3, 0.6, 1.0, 1.3]
Ri_list = list()
mTau_list = list()
# using ranges
Ra_list = np.arange(10, 50, 2)
Cm_list = np.arange(0.1, 1.3, 0.1)
# varying all 3 variables
# variableCart, variableIter = itertools.tee(
# itertools.product(Ra_list, diam_list, Cm_list))
# cartesian product of the variables
# using tee to make two copies of the same iterator so I can use it twice
# varying just 2 variables
diam = 3
variableCart, variableIter = itertools.tee(itertools.product(Ra_list, Cm_list))
# df = pd.DataFrame(list(variableCart),
#                   columns = ['Ra', 'Diam', 'Cm'])
df = pd.DataFrame(list(variableCart),
                  columns = ['Ra', 'Cm'])
for var_list in variableIter:
    cell = h.Section(name = 'cell')
    # 3 variables
    # cell.Ra = var_list[0] # axial resistance
    # diam = var_list[1]
    # cell.cm = var_list[2]
```

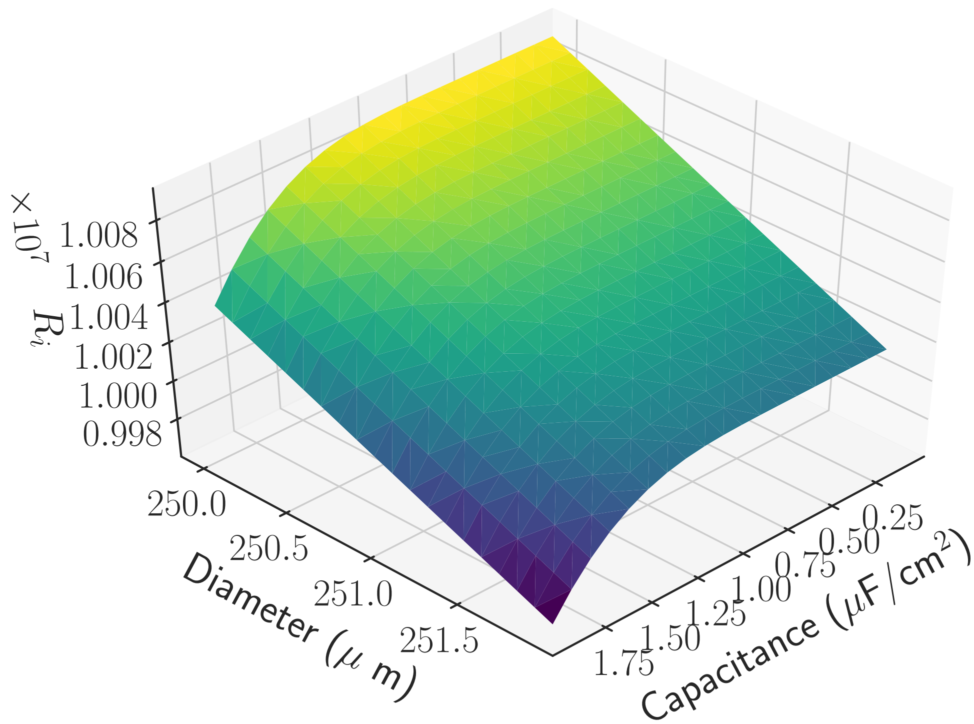
```
    # 2 variables
    cell.Ra = var_list[0]
    cell.cm = var_list[1]
    diam = 50000 # default to 500

    voltage, time = inject_current(cell = cell, somaLength = 12.6157,
somaDiam = diam, stimDur = 10, stimAmp = 0.05, stimDelay = 10)
    Ri_list.append(cal_Ri(voltage))
    mTau_list.append(cal_TauM(voltage, time))
df['Ri'] = Ri_list
df['mTau'] = mTau_list

# print preview of the table
# df
```

```
# 3D surface plot to show the effect of the 2 variables on input resistance
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
# Make the plot
fig = plt.figure(figsize=(8, 6))
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_trisurf(df['Ra'], df['Cm'], df['Ri'], cmap=plt.cm.viridis, linewidth=0.2)
ax.set(xlabel='Axial resistance ($\Omega$)',
       ylabel='Membrane capicitance ($\mu$F/cm$^2$)',
      zlabel = 'Input resistance ($\Omega$)')
# plt.show()

# to Add a color bar which maps values to colors.
surf=ax.plot_trisurf(df['Ra'], df['Cm'], df['Ri'], cmap=plt.cm.viridis, linewidth=0.2)
# fig.colorbar(surf, shrink=0.5, aspect=5) # toggle colorbar
# plt.show()
# padding for axis
ax.xaxis.labelpad = 20
ax.yaxis.labelpad = 20
ax.zaxis.labelpad = 20
# padding for tick mark
ax.tick_params(axis = 'z', pad = 10)
# Rotate it
ax.view_init(45, 45)
plt.show()
```




```
Cm_list = np.arange(0.1, 2, 0.1)
diam_list = np.arange(250, 252, 0.1) # default diam = 500
# lists to store input resistance and time constant
Ri_list = list()
mTau_list = list()
variableCart, variableIter = itertools.tee(itertools.product(Cm_list, diam_list))
df = pd.DataFrame(list(variableCart),
                  columns = ['Capacitance', 'Diameter'])
for var_list in variableIter:
    cell = h.Section(name = 'cell')

    # 2 variables
    # cell.insert('pas')
    # cell.g_pas = var_list[0]
    # diam = var_list[1]

    cell.cm = var_list[0] # default capacitance
    cell.Ra = 35.4 # default axial resistance
    diam = var_list[1]

    voltage, time = inject_current(cell = cell, somaLength = 12.6157,
somaDiam = diam, stimDur = 10, stimAmp = 0.05, stimDelay = 10)
    Ri_list.append(cal_Ri(voltage))
```

```
    mTau_list.append(cal_TauM(voltage, time))
df['Ri'] = Ri_list
df['mTau'] = mTau_list
# 3D surface plot to show the effect of the 2 variables on input resistance
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
# Make the plot
# fig = plt.figure(figsize=(8, 6), dpi = 300)
fig = plt.figure(figsize=(10, 7))
ax = fig.gca(projection='3d')
ax.plot_trisurf(df['Capacitance'], df['Diameter'], df['Ri'],
                cmap=plt.cm.viridis, linewidth=0,
                edgecolor='none',
                antialiased=True)
ax.set(xlabel='Capacitance ($\mu$F/cm$^2$)', ylabel='Diameter ($\mu$ m)',
      zlabel = 'Input resistance ($\Omega$)')
# plt.show()

# to Add a color bar which maps values to colors.
# surf=ax.plot_trisurf(df['Capacitance'], df['Diameter'], df['Ri'],
#                      cmap=plt.cm.viridis, linewidth=0.2)
# fig.colorbar(surf, shrink=0.5, aspect=5)
# plt.show()
# padding for axis
ax.xaxis.labelpad = 20
ax.yaxis.labelpad = 20
ax.zaxis.labelpad = 20
# change the label so it doens't overlap
ax.set_zlabel(zlabel='$R_i$')
# padding for tick marks
ax.ticklabel_format(axis = 'z', style = 'sci', scilimits = [0, 0])
ax.tick_params(axis = 'z', pad = 10)

# Rotate it
ax.view_init(45, 45)
plt.show()
```



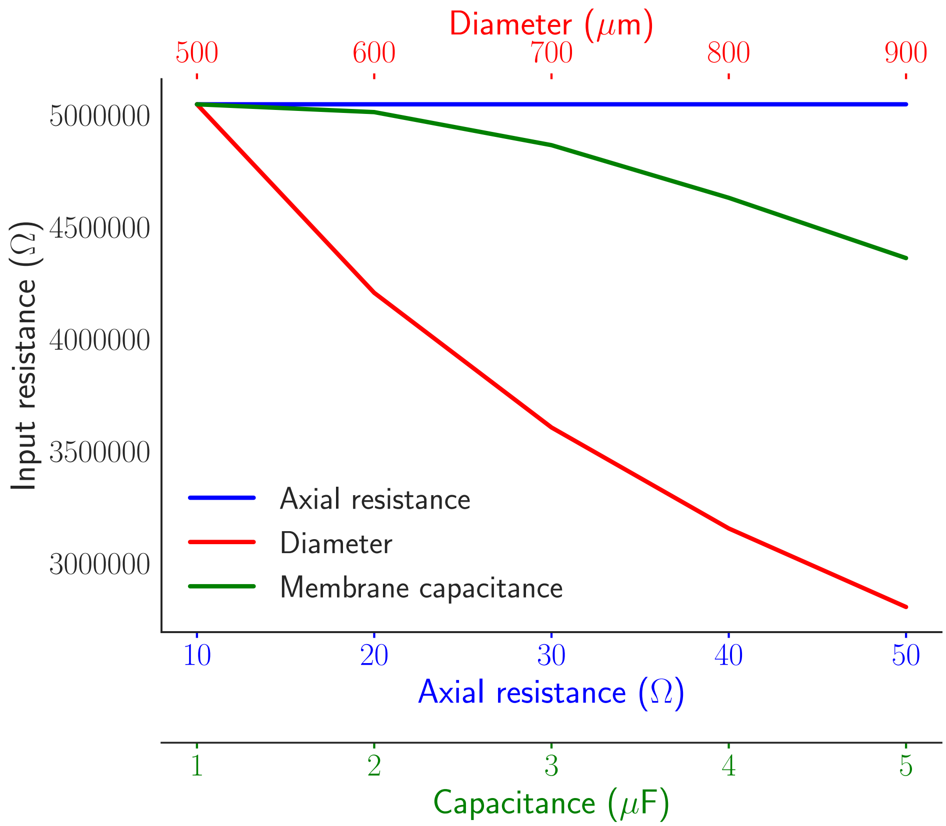

#### 1.2.2.1 Holding all variables except one

```
# second approach
# holding everything else in their default values, then varying one variable
# use multiple y-axis for visualisation
Ra_default = 10 # uOhm (?)
diam_default = 500 # uM (?)
Cm_default = 1 # uF/cm^2 (?)
Ra_list = [10, 20, 30, 40, 50]
diam_list = [500, 600, 700, 800, 900]
Cm_list = [1, 2, 3, 4, 5]
Ri_Ra_list = list()
Ri_diam_list = list()
Ri_Cm_list = list()
mTau_Ra_list = list()
mTau_diam_list = list()
mTau_Cm_list = list()
# the effect of axial resistance on input resistance and time constant
for Ra in Ra_list:
    cell = h.Section(name = 'cell')
    cell.Ra = Ra
    diam = diam_default
```

```
    cell.cm = Cm_default
    voltage, time = inject_current(cell = cell, somaLength = 12.6157,
somaDiam = diam, stimDur = 10, stimAmp = 0.05, stimDelay = 10)
    Ri_Ra_list.append(cal_Ri(voltage))
    mTau_Ra_list.append(cal_TauM(voltage, time))

# the effect of diameter on input resistance and time constant
for diam in diam_list:
    cell = h.Section(name = 'cell')
    cell.Ra = Ra_default
    diam = diam
    cell.cm = Cm_default
    voltage, time = inject_current(cell = cell, somaLength = 12.6157,
somaDiam = diam, stimDur = 10, stimAmp = 0.05, stimDelay = 10)
    Ri_diam_list.append(cal_Ri(voltage))
    mTau_diam_list.append(cal_TauM(voltage, time))
# the effect of membrane capacitance on input resistance and time constant
for Cm in Cm_list:
    cell = h.Section(name = 'cell')
    cell.Ra = Ra_default
    diam = diam_default
    cell.cm = Cm
    voltage, time = inject_current(cell = cell, somaLength = 12.6157,
somaDiam = diam, stimDur = 10, stimAmp = 0.05, stimDelay = 10)
    Ri_Cm_list.append(cal_Ri(voltage))
    mTau_Cm_list.append(cal_TauM(voltage, time))
```

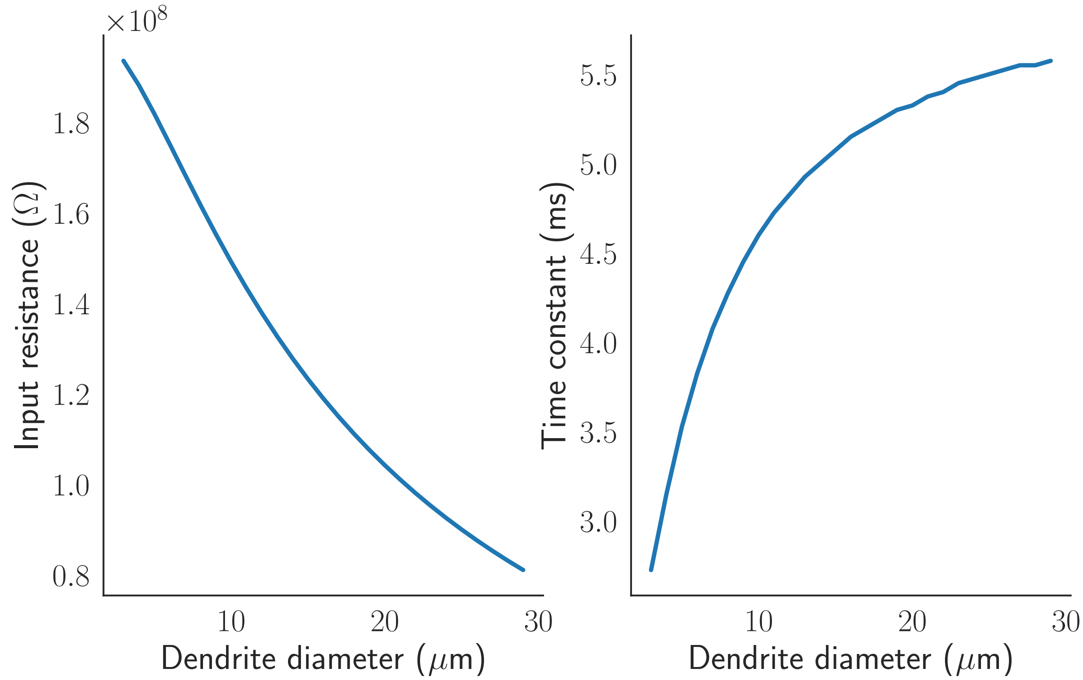
```
# based on: https://matplotlib.org/gallery/ticks_and_spines/multiple_yaxis_with_spines.html
# but instead of twinx, I am using twiny, and so the top rather than right axis is reset
import matplotlib.pyplot as plt
aesthetics(font_scale = 2, line_width = 3) # 2, 3
def make_patch_spines_invisible(ax):
    ax.set_frame_on(True)
    ax.patch.set_visible(False)
    for sp in ax.spines.values():
        sp.set_visible(False)
fig, host = plt.subplots()
fig.subplots_adjust(right=0.75)
par1 = host.twiny()
par2 = host.twiny()
# Offset the right spine of par2.  The ticks and label have already been
# placed on the right by twinx above.
# par2.spines["right"].set_position(("axes", 1.2))
par2.xaxis.set_ticks_position('bottom') # set the position of the second x-axis to bottom
par2.xaxis.set_label_position('bottom') # set the position of the second x-axis to bottom
par2.spines["bottom"].set_position(('axes', -0.2))
# Having been created by twinx, par2 has its frame off, so the line of its
# detached spine is invisible.  First, activate the frame but make the patch
# and spines invisible.
make_patch_spines_invisible(par2)
# Second, show the right spine.
# par2.spines["right"].set_visible(True)
par2.spines["bottom"].set_visible(True)
p1, = host.plot(Ra_list, Ri_Ra_list, "b-", label="Axial resistance")
```

```
p2, = par1.plot(diam_list, Ri_diam_list, "r-", label="Diameter")
p3, = par2.plot(Cm_list, Ri_Cm_list, "g-", label="Membrane capacitance")
host.set_xlabel("Axial resistance ($\Omega$)")
host.set_ylabel("Input resistance ($\Omega$)")
par1.set_xlabel('Diameter ($\mu$m)')
par2.set_xlabel('Capacitance ($\mu$F)')
host.xaxis.label.set_color(p1.get_color())
par1.xaxis.label.set_color(p2.get_color())
par2.xaxis.label.set_color(p3.get_color())
tkw = dict(size=4, width=1.5)
host.tick_params(axis='x', colors=p1.get_color(), **tkw)
par1.tick_params(axis='x', colors=p2.get_color(), **tkw)
par2.tick_params(axis='x', colors=p3.get_color(), **tkw)
host.tick_params(axis='y', **tkw)
lines = [p1, p2, p3]
host.legend(lines, [l.get_label() for l in lines], frameon = False)
plt.subplots_adjust(bottom = 0.5) # adjust bottom margin to show second bottom x-axis
plt.show()
```


Diameter ($\mu$m)
500
600
700
800
900
5000000
4500000
4000000
3500000
3000000
Input resistance ($\Omega$)
Axial resistance
Diameter
Membrane capacitance
10
20
30
40
50
Axial resistance ($\Omega$)
1
2
3
4
5
Capacitance ($\mu$F)

### 1.3 Adding dendrites and axons with increasing total area

```
from neuron import h, gui
# dendrite_diam_list = [3, 6, 9, 12, 15]
dendrite_diam_list = np.arange(3, 30)
axon_diam_list = [3, 6, 9, 12, 15]
input_resistance_list = list()
TauM_list = list()
theSoma = h.Section(name='theSoma')
dend = h.Section(name='dendrite')
# connect dendrite to soma
dend.connect(theSoma(1))
# h.psection()
# h.topology()
# axon = neuron.h.Section(name='axon')
# axon.connect(soma(0))
# inject current
for dend_diam in dendrite_diam_list:
    dend.diam = dend_diam
    # print(dend.psection())
    voltage, time = inject_current(cell = theSoma, somaLength = 12.6157, somaDiam = 12.6157,
                   stimDur = 10, stimAmp = 0.05, stimDelay = 10, total_time = 25,
                  printProperties = False)
    input_resistance = cal_Ri(voltage)
    TauM = cal_TauM(voltage, time)

    input_resistance_list.append(input_resistance)
    TauM_list.append(TauM)


import seaborn as sns
plt.figure(figsize = (12, 7))
plt.subplot(1, 2, 1)
ax = sns.lineplot(dendrite_diam_list, input_resistance_list)
ax.set(xlabel='Dendrite diameter ($\mu$m)',
      ylabel='Input resistance ($\Omega$)')
plt.subplot(1, 2, 2)
ax = sns.lineplot(dendrite_diam_list, TauM_list)
ax.set(xlabel='Dendrite diameter ($\mu$m)', ylabel='Time constant (ms)')
plt.show()
```




# 2 Active properties

## 2.1 Adding HH channels

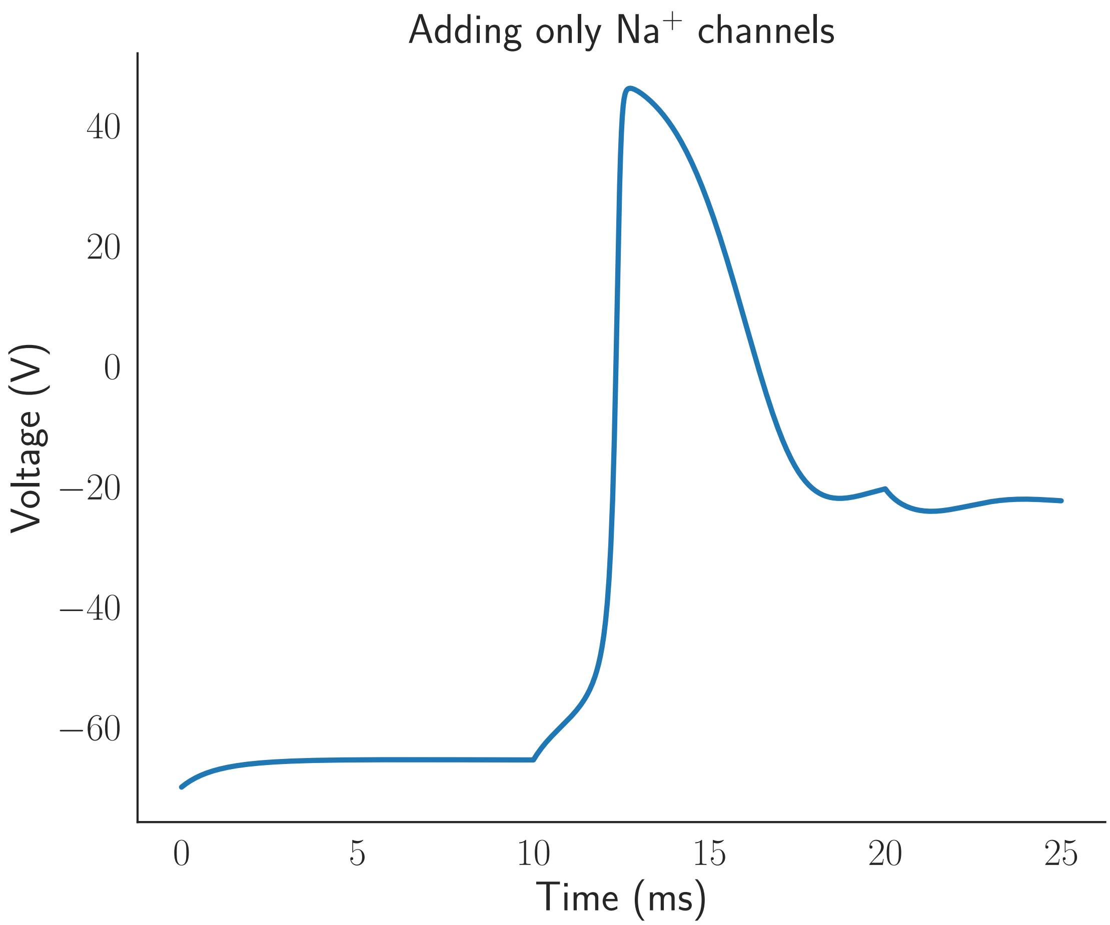
Add HH Na and K voltage-gated channel to the ball model and inject current steps (add one at a time to see the effectof each one, and then combine).

```
from neuron import h, gui
def make_hh_cell(cellName = 'cell',
                 gNa = 0.12, gK = 0.036, gL = 0.0003, eL = -54.3,
                 printProperties = False):
    """
    makes cell with specified channel conductances
    INPUT
    gNa | Sodium channel conductance (S / cm2)
    gK  | Potassium channel conductance
    gL  | Leak conductance
    eL  | Reversal potential (mV)

    Note that default values set in this function are the same as in cell.insert('hh')
    Other implicitly defined properties:
    ena = 50
    ek = -77
    cm = 1
    diam = 500, L = 100, Ra = 35.4
    """
```

```python
    cell = h.Section(name = cellName)
    cell.insert('hh')
    cell.gnabar_hh = gNa
    cell.gkbar_hh = gK
    cell.gl_hh = gL
    cell.el_hh = eL

    if printProperties is True:
        h.topology()
        h.psection()

    return cell
```

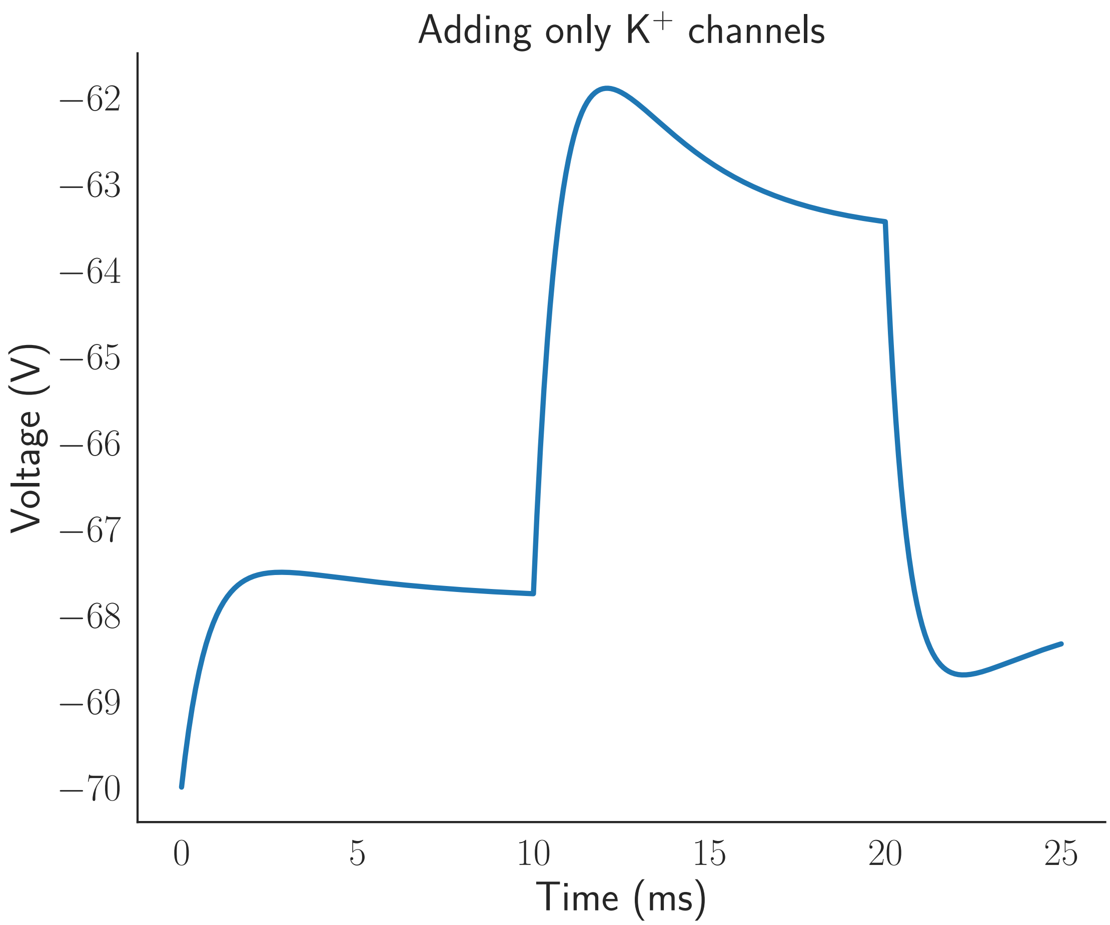
### 2.1.1 Only adding Na voltage-gated channel

```python
from neuron import h, gui
import seaborn as sns

soma_hh = h.Section(name='soma_hh')
soma_hh.insert('hh')
soma_hh.gnabar_hh = 0.12 # peak sodium conductance
soma_hh.gkbar_hh = 0 # remove sodium conductance
soma_hh.el_hh = -54.3 # leaak conductance
# inject current
voltage_np, time_np = inject_current(cell = soma_hh, somaLength = 12.6157,
somaDiam = 12.6157, stimDur = 10, stimAmp = 0.05, stimDelay = 10, total_time = 25)
# plot
plt.figure(figsize = (10, 8))
ax = sns.lineplot(time_np, voltage_np)
ax.set_title('Adding only Na$^+$ channels')
ax.set(xlabel='Time (ms)', ylabel='Voltage (V)')
sns.despine()
plt.show()
# print(time_np)
```



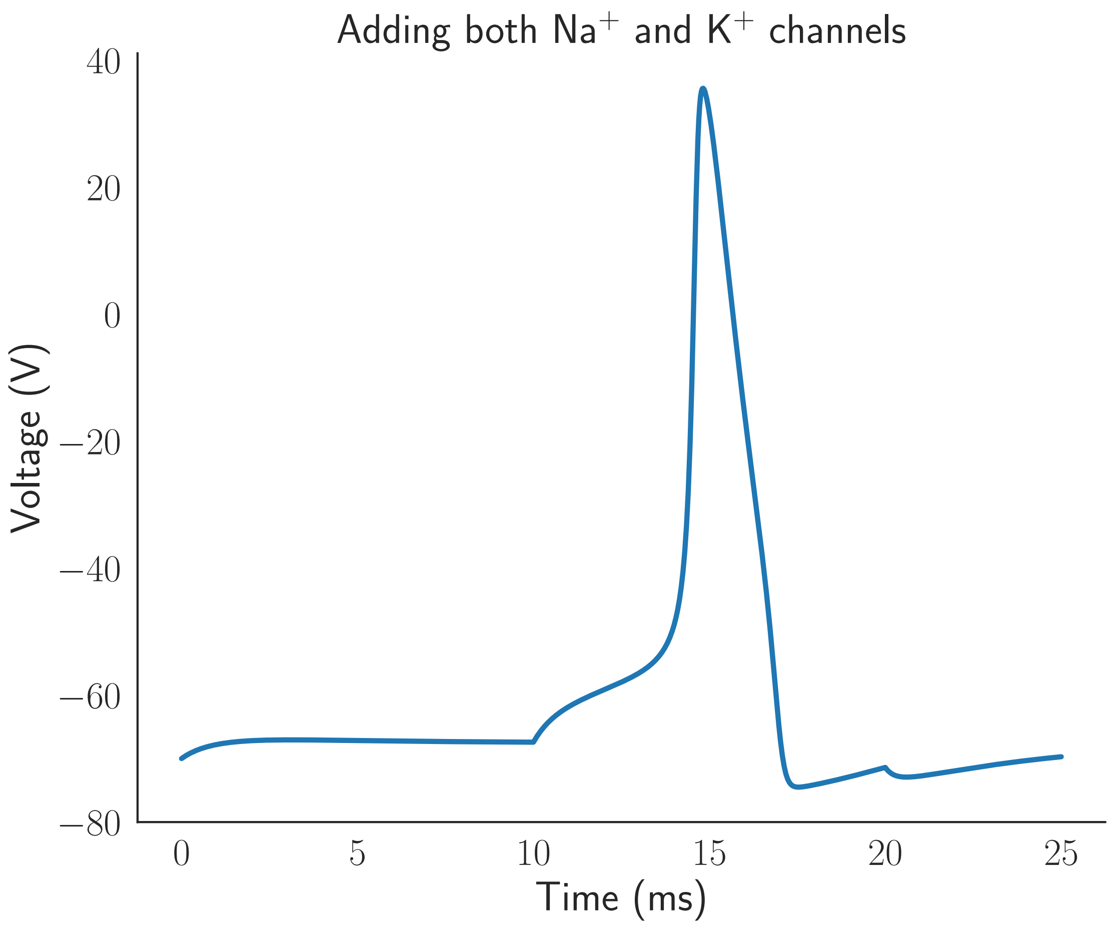

#### 2.1.2 Only adding K+ voltage-gated channels

```
k_only_cell = make_hh_cell(cellName = 'cell', gNa = 0, gK = 0.036, gL = 0.0003)
voltage_np, time_np = inject_current(cell = k_only_cell, somaLength = 12.6157,
somaDiam = 12.6157, stimDur = 10, stimAmp = 0.05, stimDelay = 10, total_time = 25)
plt.figure(figsize = (10, 8))
ax = sns.lineplot(time_np, voltage_np)
ax.set_title('Adding only K$^+$ channels')
ax.set(xlabel='Time (ms)', ylabel='Voltage (V)')
sns.despine()
plt.show()
```




#### 2.1.3 Adding both Na+ and K+ voltage-gated channels

```
hh_cell = make_hh_cell(cellName = 'hh_cell', gNa =  0.12, gK = 0.036, gL = 0.0003)
voltage_np, time_np = inject_current(cell = hh_cell, somaLength = 12.6157,
somaDiam = 12.6157, stimDur = 10, stimAmp = 0.05, stimDelay = 10, total_time = 25)
# aesthetics()
plt.figure(figsize = (10, 8))
ax = sns.lineplot(time_np, voltage_np)
ax.set_title('Adding both Na$^+$ and K$^+$ channels')
ax.set(xlabel='Time (ms)', ylabel='Voltage (V)')
sns.despine()
plt.show()
```




### 2.1.4 Measure gNa and gKv as a function of current injection at the soma and plot activation curves

We first define a function to measure the conductance at the soma.

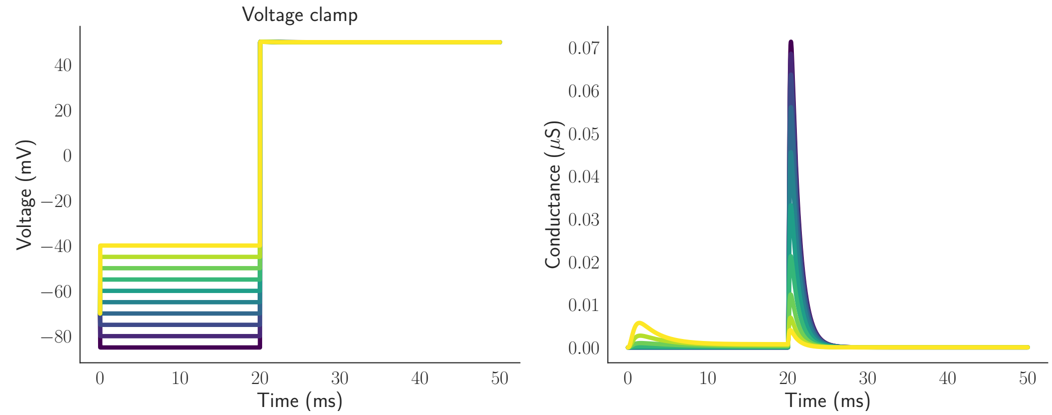
```
from neuron import h, gui
import seaborn as sns
import matplotlib.pyplot as plt
currMin = 0
currMax = 10
currStep = 1
curr_amp_list = np.arange(currMin, currMax, currStep)
gNa_measured = list()
gKv_measured = list()
max_voltage_list = list()
def record_conductance(cell, somaLength = 12.6157, somaDiam = 12.6157,
                       stimDur = 10, stimAmp = 0.05, stimDelay = 10,
                       total_time = 25, vInit = -70,
```

```
                   v_clamp_amp1 = 10, v_clamp_amp2 = 30, v_clamp_amp3 = 10,
                   v_clamp_dur1 = 10, v_clamp_dur2 = 20, v_clamp_dur3 = 20,
              printProperties = False,
                  injectCurrent = False):

    # cell.insert('pas')
    cell.L = somaLength
    cell.diam = somaDiam

    if printProperties is True:
        print('Cell properties')
        h.psection()

    cell.insert('hh') # insert HH properties
    # set Vm
    # cell.el_hh = vInit

    # set resting potential
    # for seg in cell:
    #     seg.hh.el = vInit

    if printProperties is True:
        h.psection(cell)

    # add point process: current injection
    if injectCurrent is True:
        stim = h.IClamp(cell(0.5))
        stim.delay = stimDelay
        stim.dur = stimDur # ms
        stim.amp = stimAmp # nA

    v_clamp = h.SEClamp(cell(0.5))
    v_clamp.amp1 = v_clamp_amp1
    v_clamp.amp2 = v_clamp_amp2

    # series resistance
    v_clamp.rs = 0.01

    v_clamp.dur1 = v_clamp_dur1
    v_clamp.dur2 = v_clamp_dur2
    v_clamp.dur3 = v_clamp_dur3


    # measure voltage
    voltage = h.Vector()
    voltage.record(cell(0.5)._ref_v)

    gNa = h.Vector()
    gNa.record(cell(0.5)._ref_gna_hh)

    gKv = h.Vector()
    gKv.record(cell(0.5)._ref_gk_hh)
```

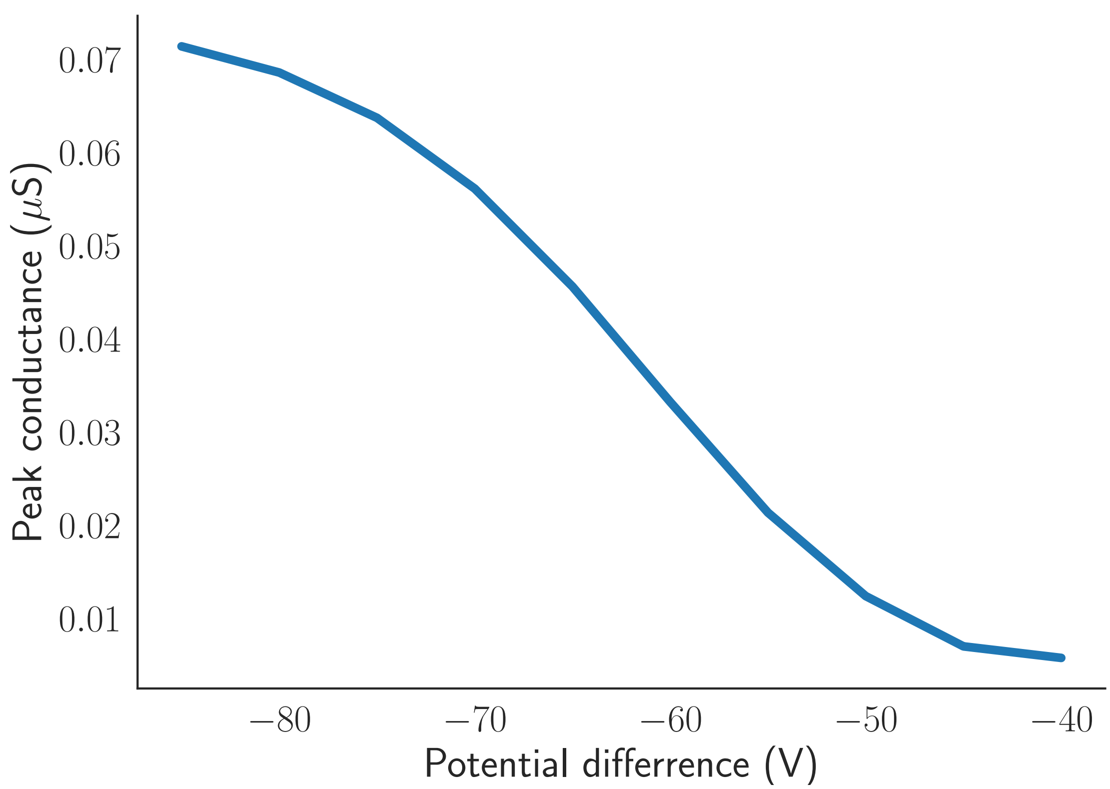
```
    # measure time
    time = h.Vector()
    time.record(h._ref_t)

    # h.v_init = vInit # initial voltage
    # h.finitialize(vInit)

    h.load_file('stdrun.hoc')
    h.tstop = total_time
    h.run() # ms


    return gNa, gKv, voltage, time
```

```
# Vm_list = [-70, -60, -50, -40, -30]
# v_clamp_amp1_list = [-100, -90, -80, -70, -60, 20]
# v_clamp_amp1_list = np.arange(-150, 50, 5)
v_clamp_amp1_list = np.arange(-85, -35, 5)
v_clamp_amp2 = 50
v_clamp_dur1 = 20
v_clamp_dur2 = 30
curr_amp = 0.5
gNa_measured = list()
gKv_measured = list()
aesthetics()
# colormap
import matplotlib.pylab as pl
num_color = len(v_clamp_amp1_list)
colors = pl.cm.viridis(np.linspace(0,1,num_color))
plt.figure(figsize = (20, 7))
ax1 = plt.subplot(1, 2, 1)
ax2 = plt.subplot(1, 2, 2)
for v_clamp_amp1, n in zip(v_clamp_amp1_list, np.arange(num_color)):
    conductance_cell = h.Section(name='conductance_cell')
    gNa, gKv, voltage, time = record_conductance(conductance_cell, stimAmp = curr_amp,
    vInit = -70, stimDelay = 10, total_time = 50,
    v_clamp_amp1 = v_clamp_amp1, v_clamp_amp2 = v_clamp_amp2,
    v_clamp_dur1 = v_clamp_dur1, v_clamp_dur2 = v_clamp_dur2,
    printProperties = False)
    gNa_measured.append(max(gNa))
    gKv_measured.append(max(gKv))

    ax1.plot(time, voltage,
             label = v_clamp_amp1,
             color = colors[n])
    ax2.plot(time, gNa,
             label = v_clamp_amp1,
             color = colors[n])
ax1.set_title('Voltage clamp')
ax1.set_xlabel('Time (ms)')
ax1.set_ylabel('Voltage (mV)')
ax2.set_xlabel('Time (ms)')
ax2.set_ylabel('Conductance ($\mu$S)')
# plt.legend(frameon = False, title = 'Initial holding voltage \n (mV)')
```

```
plt.show()
```

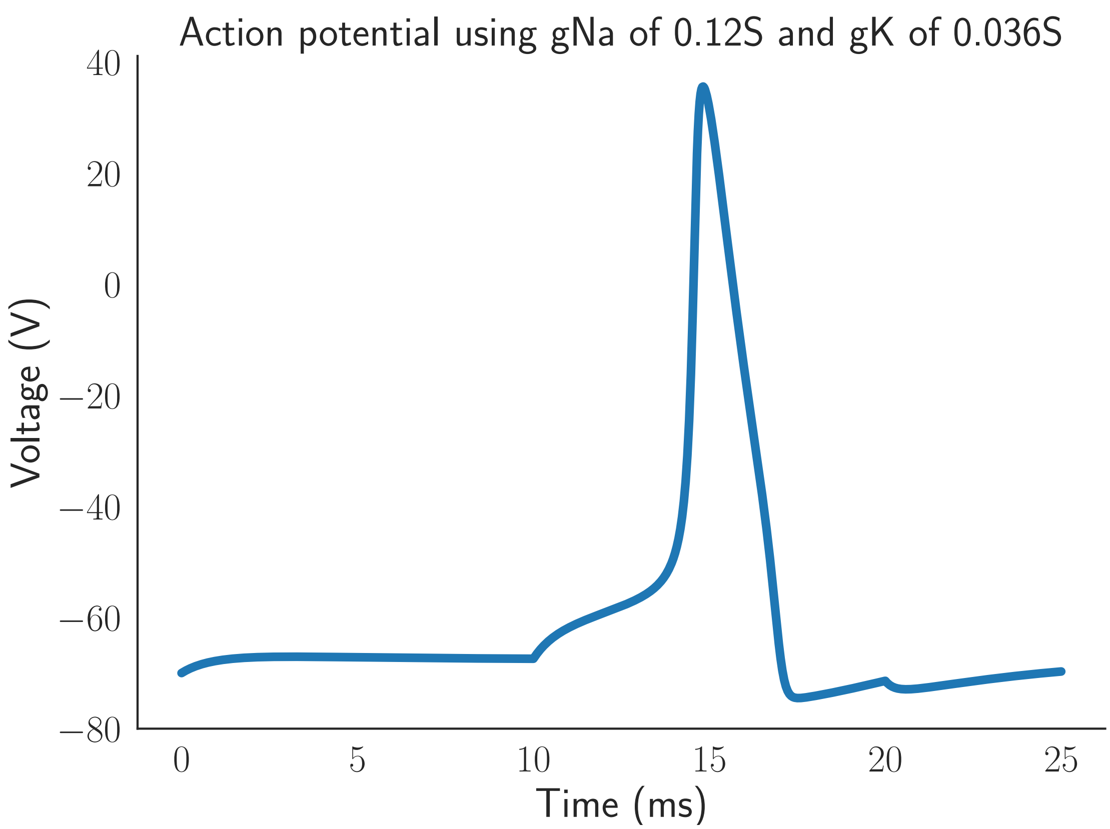



```
plt.figure()
ax = sns.lineplot(v_clamp_amp1_list, gNa_measured)
plt.xlabel('Potential differrence (V)')
plt.ylabel('Peak conductance ($\mu$S)')
plt.show()
```



### 2.2 Play with gNa and gK density to make an action potential

```
gNa_custom = 0.12
gK_custom = 0.036
ap_cell = make_hh_cell(cellName = 'ap_cell', gNa = gNa_custom, gK = gK_custom, gL = 0.0003, eL = -54.3,
voltage_np, time_np = inject_current(cell = ap_cell, somaLength = 12.6157,
somaDiam = 12.6157, stimDur = 10, stimAmp = 0.05, stimDelay = 10, total_time = 25)
aesthetics()
plt.figure()
ax = sns.lineplot(time_np, voltage_np)
ax.set_title('Action potential using gNa of %.2fS and gK of %.3fS' % (gNa_custom, gK_custom))
ax.set(xlabel='Time (ms)', ylabel='Voltage (V)')
sns.despine()
plt.show()
```




## 3 Syanptic integration

### 3.1 Add a single alpha excitatory synapse in the soma and then in the dendrite

Progressively far away, and record at the soma. What changes?

Background:

- A synapse (for the purpose of this simulation) is a location of transmitter release, which then binds to post-synaptic receptors at the soma to cause change in coductance for one or more ions
- The conductance change (usually increase) can be described by a mathematical function
- An alpha synapse is a synapse in which the conductance change in the post-synapse cell can be described by the *alpha function*
- In `NEURON`, the alpha synapse is a point process (ie. it is assumed to be a point source of current along the cell)

There seems to be multiple mathematical definition of the alpha function, and `NEURON` implements the alpha function (see ref 3) via:

$$g(t) = g_{\max}\alpha t e^{1-\alpha t}$$

References:

- AlphaSynapse
- Alpha function, Wolfram
- NEURON forum on alpha function

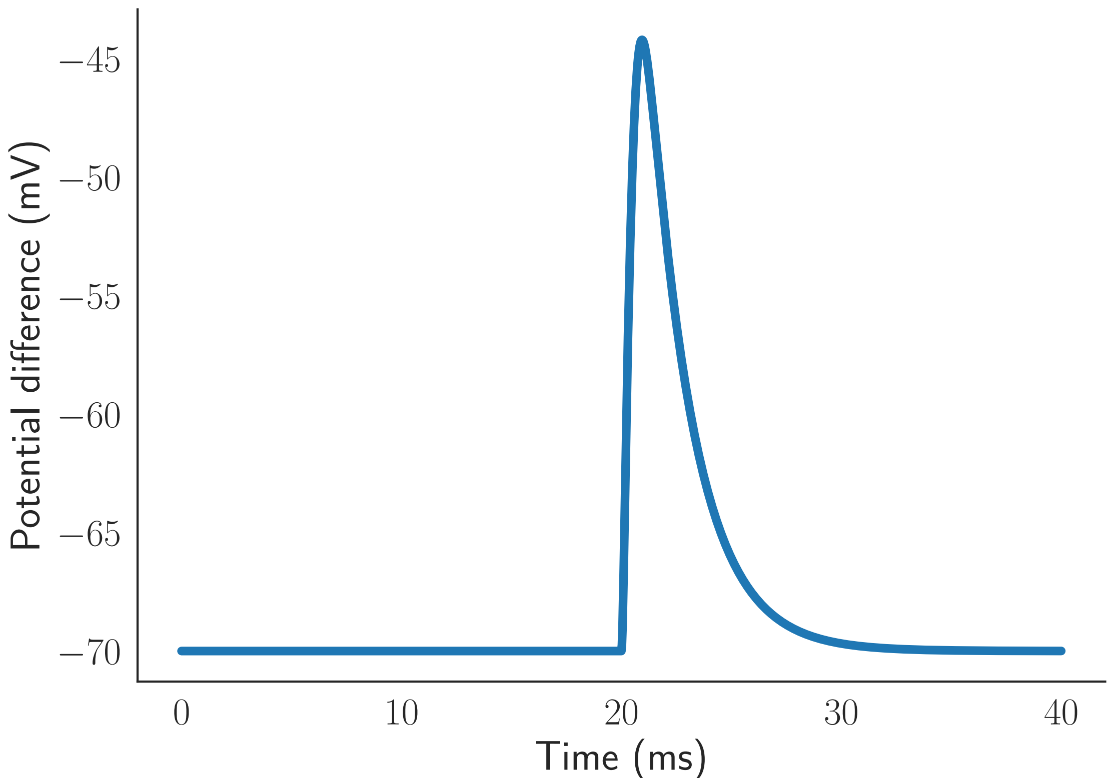
We first define functions to make a cell with alpha synapse directed to the dendrite and to record the potential at the soma.

```
from neuron import h, gui
import seaborn as sns
import matplotlib.pyplot as plt
def sim_alpha_synapse(alpha_loc = 0.5, record_loc = 0.5, simDur = 40, vInit = -70,
                      alpha_onset = 20, alpha_gmax = 1, printProperties = False,
                      nSomaSegment = 1, nDendSegment = 300,
                     somaLength = 1000, recordSegment = 'soma',
                      dendriteNum = 0):
    """
    Create alpha synapse, then simulate to record potential difference over time
    INPUT
    alpha_loc  | location of inserting alpha syanapse
    record_loc | location of recording the alpha synapse
    simDur     | duration of the simulation (ms)
    """
    soma = h.Section(name='soma')
    soma.insert('pas')

    soma.L = 20 #somaLength
    soma.diam = 20 #

    # alpha synapse on soma
    # soma.nseg = nSomaSegment
    # asyn = h.AlphaSynapse(soma(alpha_loc))

    # alpha synapse on dendrite
    dend = h.Section(name = 'dend')
    dend.L = 300
    dend.diam = 1
    dend.nseg = nDendSegment
    asyn = h.AlphaSynapse(dend(alpha_loc))
    dend.connect(soma(0))
```

```
    asyn.onset = alpha_onset
    asyn.gmax = alpha_gmax


    if printProperties is True:
        h.psection(soma)
        h.psection(dend)
        # print(dir(asyn))

    v_vec = h.Vector()             # Membrane potential vector
    t_vec = h.Vector()             # Time stamp vector
    if recordSegment == 'soma':
        # print('Recording at the soma')
        v_vec.record(soma(record_loc)._ref_v)
    elif recordSegment == 'dendrite':
        # print('Recording at the dendrite')
        v_vec.record(dend(record_loc)._ref_v)
    t_vec.record(h._ref_t)
    h.v_init = vInit
    h.tstop = simDur
    h.run()
    return t_vec, v_vec
```

Then we run the simulation:

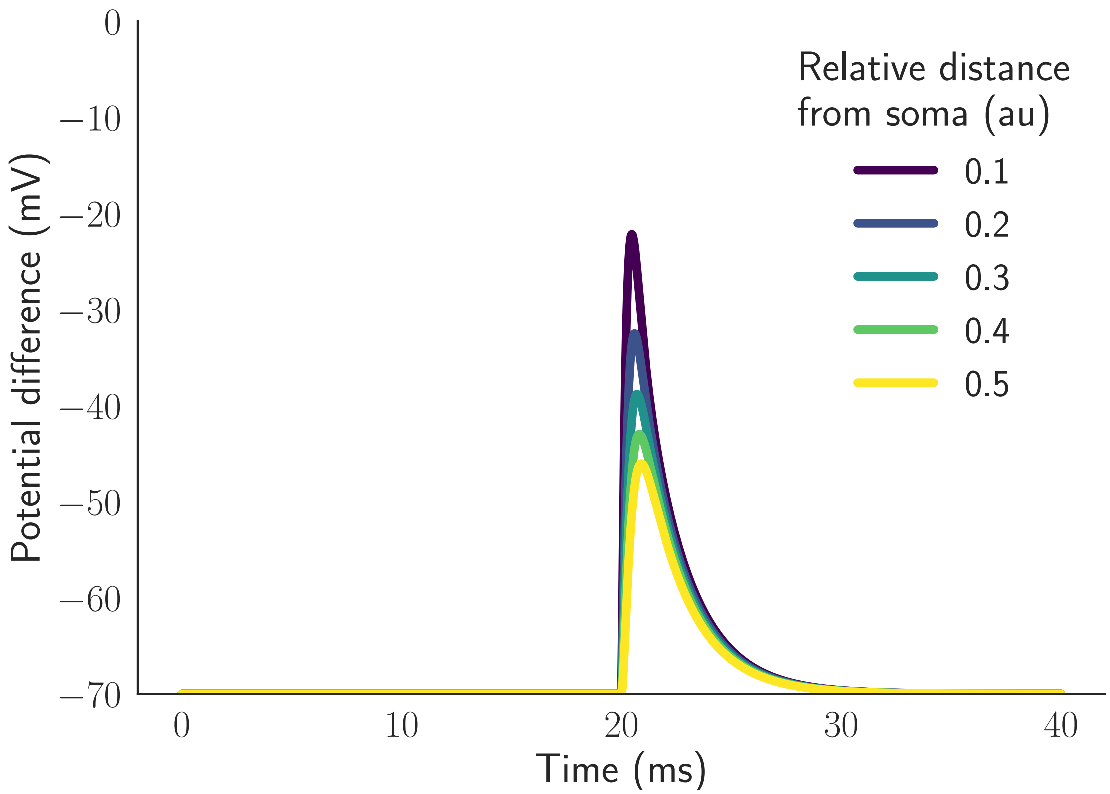
```
time, voltage = sim_alpha_synapse()
import matplotlib.pyplot as plt
plt.figure()
plt.plot(time, voltage)
plt.xlabel('Time (ms)')
plt.ylabel('Potential difference (mV)')
plt.show()
```



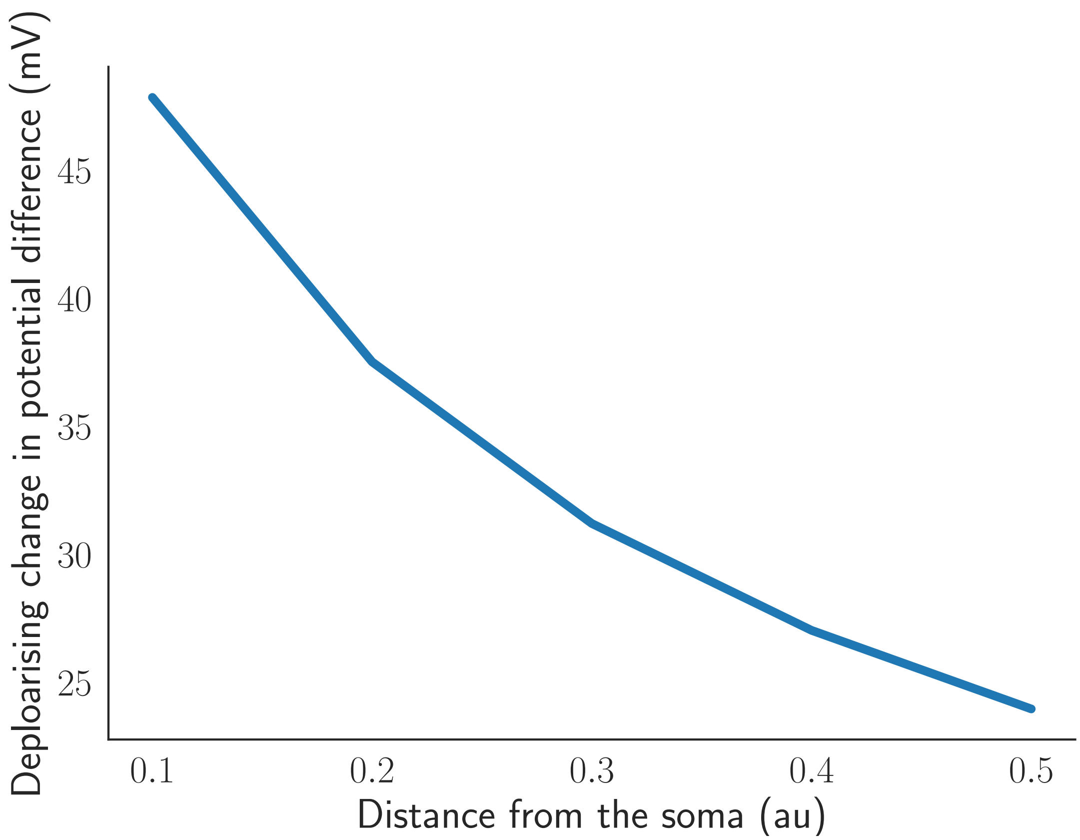

#### 3.1.1 Moving the alpha synapse progressively far away

```
# progressively far away alpha synapse
location_list = [0.1, 0.2, 0.3, 0.4, 0.5]
alpha_onset_list = [1, 5, 10, 15, 20]
peak_voltage = []
# colormap
import matplotlib.pylab as pl
num_color = len(location_list)
colors = pl.cm.viridis(np.linspace(0,1,num_color))
plt.figure()
for loc, n in zip(location_list, np.arange(num_color)):
    time, voltage = sim_alpha_synapse(alpha_loc = loc, record_loc = 0.5, simDur = 40,
                                      nSomaSegment = 100, somaLength = 1000,
                                      alpha_gmax = 0.5,
                                      printProperties = False)
    sns.lineplot(time, voltage,
                 label = loc,
                 color = colors[n])
    peak_voltage.append(max(voltage) - min(voltage))
    plt.xlabel('Time (ms)')
```

```
    plt.ylabel('Potential difference (mV)')
plt.ylim([-70, 0])
plt.legend(frameon=False, title = 'Relative distance \n from soma (au)')
plt.show()
```




```
# plot the peak potential difference against the distance
plt.figure()
sns.lineplot(location_list, peak_voltage)
plt.ylabel('Deploarising change in potential difference (mV)')
plt.xlabel('Distance from the soma (au)')
plt.show()
```

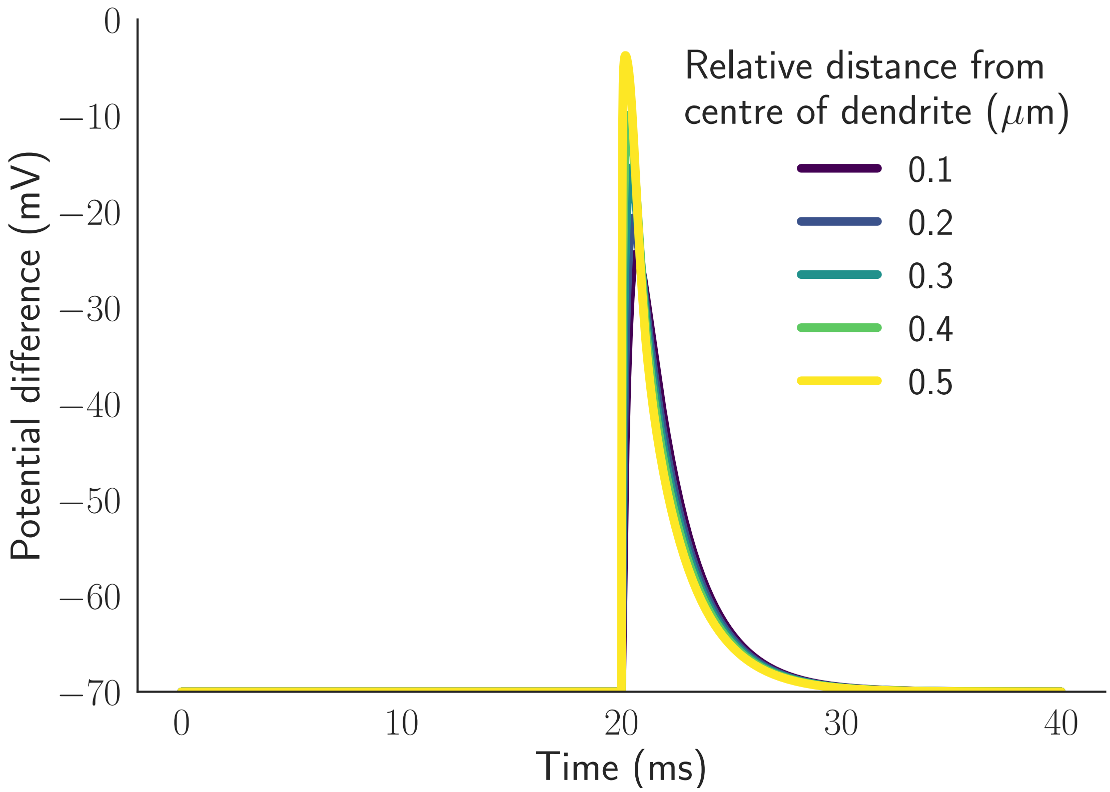



As expected we see that the amplitude of the voltage change elicted by the alphasynapse decreases with distance due to the passive properties of the cell. As a result when the alphasynapse is further away from the some the depolarising change recorded as the soma is smaller then when it is closer to the soma.

### 3.2 Record at the dendrite where the synapse is whilst moving the alpha synapse progressively far away

```
plt.figure()
location_list = [0.1, 0.2, 0.3, 0.4, 0.5]
# colormap
import matplotlib.pylab as pl
num_color = len(location_list)
colors = pl.cm.viridis(np.linspace(0,1,num_color))
for loc, n in zip(location_list, np.arange(num_color)):
    # print(loc)
    time, voltage = sim_alpha_synapse(alpha_loc = loc, record_loc = 0.5, simDur = 40,
                                      nSomaSegment = 100, somaLength = 1000,
                                      alpha_gmax = 0.5, recordSegment = 'dendrite',
                                      printProperties = False)
    sns.lineplot(time, voltage,
                 label = loc,
```

```
                 color = colors[n])
    # peak_voltage.append(max(voltage) - min(voltage))
    plt.xlabel('Time (ms)')
    plt.ylabel('Potential difference (mV)')
plt.ylim([-70, 0])
plt.legend(frameon=False, title = 'Relative distance from \n centre of dendrite ($\mu$m)',
           loc = 'upper right')
plt.show()
```




When we record the voltage at the dendrite as the alphasynapse is activated we observe that the size of the voltage change is higher than when the synapse was on near the soma. The high voltage change is due to the dendrites smaller size.