ElectroChemical

Peter Gawthrop. *peter.gawthrop@unimelb.edu.au*

December 15, 2020

Contents

Note:

- This example is discussed in detail by Gawthrop and Pan [2020] available here.
- This is the ElectroChemical.ipynb notebook. The PDF version is available here.

1 Introduction

1.1 Faraday equivalent potential

The bond graph approach uses the notion of energy covariables: a pair of variables whose product is power. Thus, for example, electrical systems have voltage (with units V) and current (with units A) as covariables and the product has units of power (W or $\text{J}\,\text{s}^{-1}$). Chemical system covariables are chemical potential μ (with units $\text{J}\,\text{C}^{-1}$) and molar flow f (with units $\text{mol}\,\text{s}^{-1}$)[Oster et al., 1971, 1973, Gawthrop and Crampin, 2014]; again the product has units of power (W or $\text{J}\,\text{s}^{-1}$).

The commonality of power over different physical domains makes the bond graph approach particularly appropriate to model multi-domain systems, in particular chemoelectrical systems [Gawthrop et al., 2017]. Noting that the conversion factor relating the electrical and chemical domains is *Faraday's constant* $F \approx 96\,485\,\text{C}\,\text{mol}^{-1}$. As discussed by Karnopp [1990] and Gawthrop et al. [2017], this conversion can be represented by the bond graph transformer (**TF**) component. An alternative approach introduced by Gawthrop [2017] is to divide the covariables μ and f by F to give the pair of covariables ϕ and f where:

$$\text{Faraday-equivalent chemical potential} \qquad \phi = \frac{\mu}{F}\text{V} \tag{1}$$

$$\text{Faraday-equivalent flow} \qquad f = Fv\text{A} \tag{2}$$

1.2 Chemical properties

The **Ce** components representing chemical species generate Faraday-equivalent potential (FEP) ϕ (measured in Volts) in terms of the amount of species x as:

$$
\begin{aligned}
\phi &= \phi^{\ominus} + \phi_N \ln \frac{x}{x^{\ominus}} && (3)\\
&= \phi_N \ln Kx && (4)\\
\text{where } K &= \frac{K^{\ominus}}{x^{\ominus}} && (5)\\
V_N &= \frac{RT}{F} \approx 26mV && (6)\\
\text{and } K^{\ominus} &= \ln \frac{\phi^{\ominus}}{\phi_N} && (7)
\end{aligned}
$$

$\phi^{\ominus}$ in the standard potential at the standard amount $x^{\ominus}$. R is the universal gas constant and F Faraday's constant.

The amount of species x is the integral of the species flow f:

$$x = \int^t f(\tau)d\tau \tag{8}$$

The formula can also be expressed in terms of concentration c as:

$$
\begin{aligned}
\phi &= \phi_N \ln K'_C c && (9)\\
\text{where } K'_c &= \frac{K^{\ominus}}{c^{\ominus}} && (10)\\
& && (11)
\end{aligned}
$$

$c^{\ominus}$ is the concentration at standard conditions. ## Electrical properties The **C** components representing electrical capacitance generate electrical potential ϕ (measured in Volts) in terms of the amount of positively charges x and electrical capacitance C as:

$$
\begin{aligned}
\phi &= \frac{x}{C} && (12)\\
&= \phi_N K_E x_E && (13)\\
\text{where } K_E &= \frac{1}{x_N} && (14)\\
\text{and } x_N &= C\phi_N && (15)
\end{aligned}
$$

The amount of charge x_E is the integral of the charge flow (current) f_E:

$$x_E = \int^t f_E(\tau)d\tau \tag{16}$$

[1]:
```python
## Some useful imports
import BondGraphTools as bgt
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt

## Stoichiometric analysis
```

```
import stoich as st

## SVG
import svgBondGraph as sbg

## Display (eg disp.SVG(), disp.
import IPython.display as disp

quiet = True

## Fix the concentrations via chemostats
Fix_conc = False
```

[2]:
```
## Concentrations in nM for Na and K in Giant Squid Axon
## From Keener & Sneyd Table 2.1
conc_e = {'Na':437, 'K':20}
conc_i = {'Na':50, 'K':397}
```

2 Electrodiffusion

Cellular membranes have pores though which chemical species can diffuse. If the species are charged, the diffusion both depends on and creates an electrical potential. This section looks at a single ionic species with generic name I^+; this can be thought of as Na^+ or K^+.

The bond graph representation of a charged ion has three components: a **Ce** component to represent the *chemical* properties of the ion, a **C** component to repesent the *electrical* properties of the ion and a **1** junction to make the flow into the two components identical.

The resultant potential is then the sum of the chemical and electrical components:

$$\phi = \phi_C + \phi_E \tag{17}$$

$$\text{where } \phi_C = \phi_N \ln Kx \tag{18}$$

$$\text{and } \phi_E = \phi_N K_E x_E \tag{19}$$

If the ion has *two* charges (I^{++}) the bold bonds in the diagram would each be replaced by *two* bonds; alternatively, if the ion had a *negative* charge (I^-) the bold bonds in the diagram would each be replaced by a bond with *reversed* direction.

The bond graph of the pore itself has two pools of charged ions: internal and external connected by a reaction (**Re**) component. As the ion in each pool is the same, the property K' is the same for each pool. Thus the reaction potential Φ is the difference of the potentials of the internal and external ion pools:

$$\Phi = \phi_N \left(\ln K'c_i - \ln K'c_e \right) + \left(\phi_{Ei} - \phi_{Ee} \right) \tag{20}$$

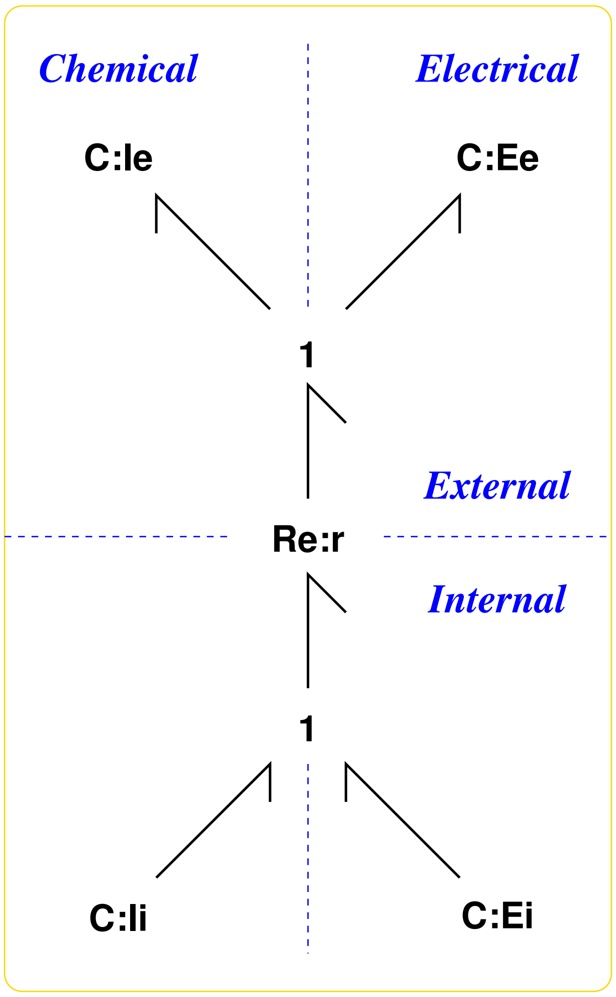
Defining $\Delta E = \phi_{Ei} - \phi_{Ee}$ and noting that at equilibrium $\Phi = 0$:

$$\Delta E = \phi_N \ln \frac{c_e}{c_i} \tag{21}$$

This is the expresion for the *Nernst potential* for a species with a single positive charge.

[3]:
```python
## Electrodiffusion
sbg.model('Electrodiffusion_abg.svg')
import Electrodiffusion_abg
disp.SVG('Electrodiffusion_abg.svg')
```

[3]:



[4]:
```python
## Stoichiometry: linear Re
s = st.stoich(Electrodiffusion_abg.model(),linear=['Ei','Ee','r'],quiet=quiet)

if Fix_conc:
    chemostats = ['Ii','Ie']
else:
    chemostats = []

sc = st.statify(s,chemostats=chemostats)
#print(s['species'])
```

[5]:
```python
## Stoichiometric matrix
disp.Latex(st.sprintl(s,'N'))
#print(st.sprintl(s,'species'))
```

[5]:

$$N = \left(\begin{matrix}1\\-1\\1\\-1\end{matrix}\right) \tag{22}$$

[6]:
```
## Reactions
disp.Latex(st.sprintrl(s,chemformula=True,all=True))
```

[6]:

$$\mathrm{Ei} + \mathrm{Ii} \xrightleftharpoons{r} \mathrm{Ee} + \mathrm{Ie} \tag{23}$$

[7]:
```
## Flows
disp.Latex(st.sprintvl(s))
```

[7]:

$$v_r = \kappa_r \left(-K_{Ee}x_{Ee} + K_{Ei}x_{Ei} - V_N \left(\log\left(K_{Ie}x_{Ie}\right) - \log\left(K_{Ii}x_{Ii}\right)\right)\right) \tag{24}$$

[8]:
```
## Stoichiometry: nonlinear Re
s = st.stoich(Electrodiffusion_abg.model(),linear=['Ei','Ee'],quiet=quiet)

if Fix_conc:
    chemostats = ['Ii','Ie']
else:
    chemostats = []

sc = st.statify(s,chemostats=chemostats)
#print(s['species'])
```

[9]:
```
## Reactions
disp.Latex(st.sprintrl(s,chemformula=True,all=True))
```

[9]:

$$\mathrm{Ei} + \mathrm{Ii} \xrightleftharpoons{r} \mathrm{Ee} + \mathrm{Ie} \tag{25}$$

[10]:
```
## Flows
print(st.sprintvl(s))
disp.Latex(st.sprintvl(s))
```

```
\begin{align}
v_{r} &= \kappa_{r} \left(- K_{Ie} x_{Ie} e^{\frac{K_{Ee} x_{Ee}}{V_{N}}} +
K_{Ii} x_{Ii} e^{\frac{K_{Ei} x_{Ei}}{V_{N}}}\right)
\end{align}
```

[10]:

$$v_r = \kappa_r \left(-K_{Ie}x_{Ie}e^{\frac{K_{Ee}x_{Ee}}{V_N}} + K_{Ii}x_{Ii}e^{\frac{K_{Ei}x_{Ei}}{V_N}}\right) \tag{26}$$

[ ]:

```

```

[11]:

```
#disp.Latex(st.sprintl(s,'species'))
```

[12]:

```
#disp.Latex(st.sprintl(s,'N'))
```

[13]:

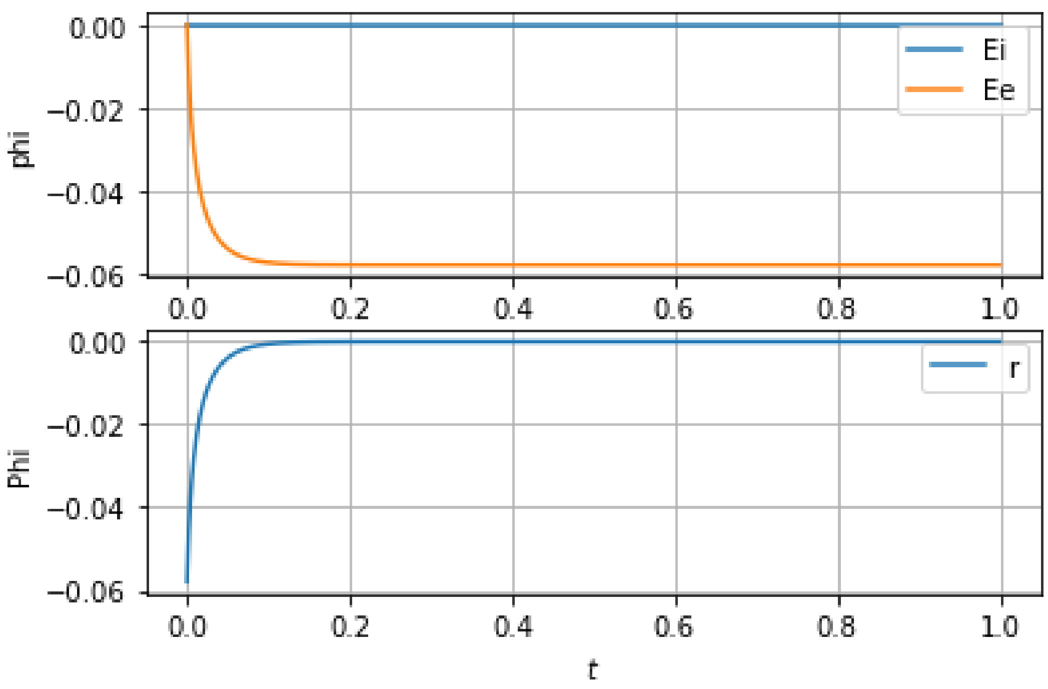
```
## Set non-unit parameters
K_Ii = 1e-3
K_Ie = 1e-3
C = 1
def setPar(s,C=1,conc_i=1,conc_e=1,prefix=['']):

    #V_N = st.V_N()
    K_E = 1/C
    #print(K_E)


    ## Parameters
    parameter = {}
    parameter['K_Ei'] = 0
    parameter['K_Ee'] = K_E

    ## Initial state
    sp = s['species']
    re = s['reaction']
    X0 = np.ones(s['n_X'])
    X0[sp.index('Ei')] = 0
    X0[sp.index('Ee')] = 0
    for p in prefix:

        ## Parameters
        KK = 'K_'+p
        kk = 'kappa_'+p
        parameter[KK+'Ii'] = K_Ii
        parameter[KK+'Ie'] = K_Ie

        ## States and kappa
        if len(p) == 0:
            Ion = 'Na'
        else:
            Ion = p[0:len(p)-1]
            #X0[sp.index('Ee')] = 0.077/K_E

        print(Ion)
        X0[sp.index(p+'Ii')] = conc_i[Ion]/K_Ii
        X0[sp.index(p+'Ie')] = conc_e[Ion]/K_Ie
        parameter[kk+'r'] = 1/conc_i[Ion]

    return parameter,X0
```

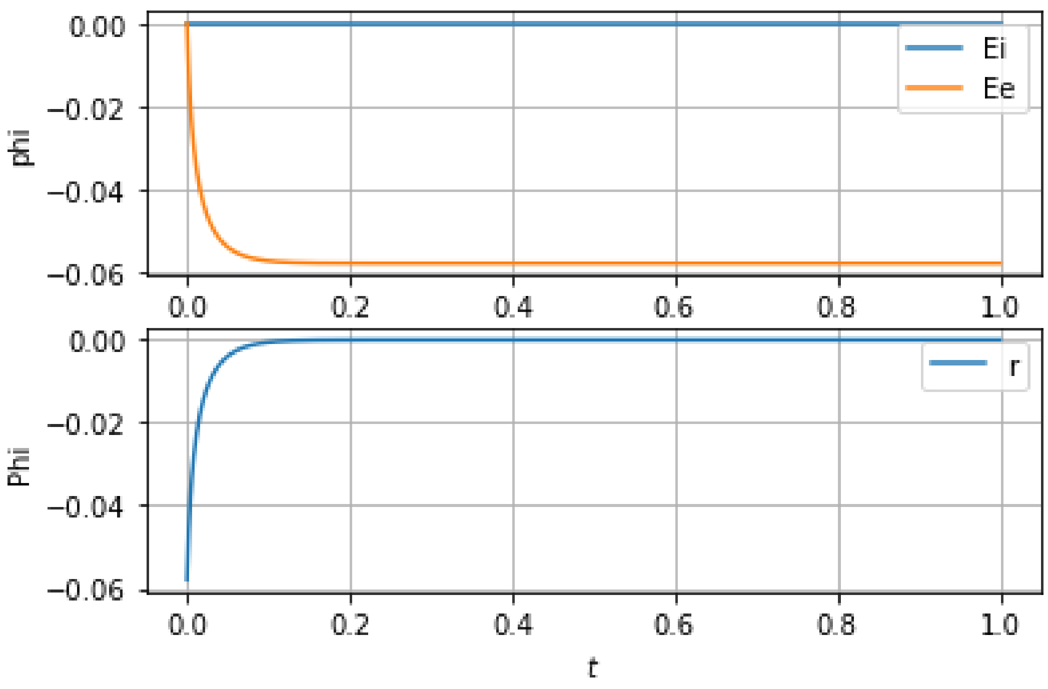
```
[14]: def CheckTheory(dat):

          if 'Ii' in s['species']:
              ## Check Nernst potential
              t = dat['t']
              phi_Ei = dat['phi'][:,s['species'].index('Ei')]
              phi_Ee = dat['phi'][:,s['species'].index('Ee')]
              x_Ii = dat['X'][:,s['species'].index('Ii')]
              x_Ie = dat['X'][:,s['species'].index('Ie')]
          #     v = dat['V'][:,s['reaction'].index('r')]
              V_N = st.V_N()

          #     v_ss = v[-1]
              dV = (phi_Ei[-1]-phi_Ee[-1] )
              dV_theory = V_N*np.log(x_Ie[-1]/x_Ii[-1])
          #     print(f'Steady-state flow is {v_ss:0.2}')
              print(f'dV = {dV*1000:4.1f}mV')
              print(f'dV_Theory = {dV_theory*1000:4.1f}mV')
```

```
[15]: def Simulate(s,sc,T=1,X_chemo=None,prefix=['']):

          ## Time
          t = np.linspace(0,T,500)

          ## Parameters and initial state
          parameter,X0 = setPar(s,C=C,conc_i=conc_i,conc_e=conc_e,prefix=prefix)

          ## Simulate
          dat = st.
       ↪sim(s,sc=sc,t=t,parameter=parameter,X0=X0,X_chemo=X_chemo,quiet=True)

          CheckTheory(dat)

          return dat
```

```
[16]: dat = Simulate(s,sc)
      #st.plot(s,dat)
      st.plot(s,dat,plotPhi=True,species=['Ei','Ee'])
      st.plot(s,dat,plotPhi=True,species=['Ei','Ee'],filename='Figs/
       ↪electrodiffusion.pdf')
```

```
Na
dV = 57.9mV
dV_Theory = 57.9mV
```





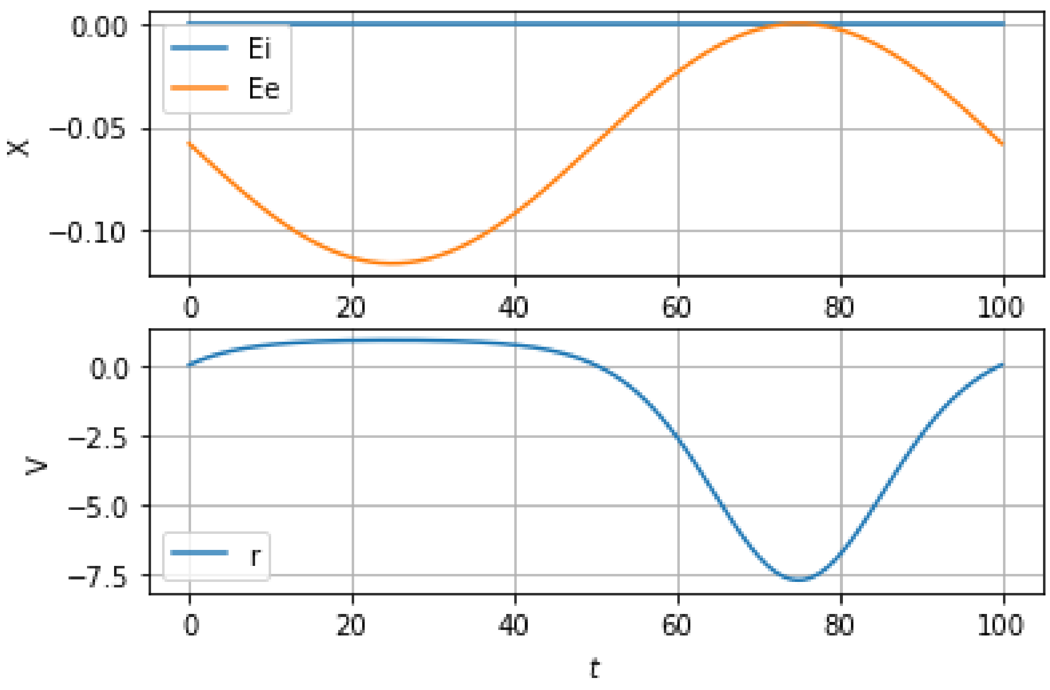
2.1 Voltage clamp

The voltage agross the membrane is clamped by setting C:Ei and C_Ee as chemostats. This allows the voltage-current relationship to be plotted. It is compared with the Hodgkin-Huxley (linear) model and the Goldman-Huxley-Katz model. The bond graph model can be modified to reflect the other two models Gawthrop et al. [2017].

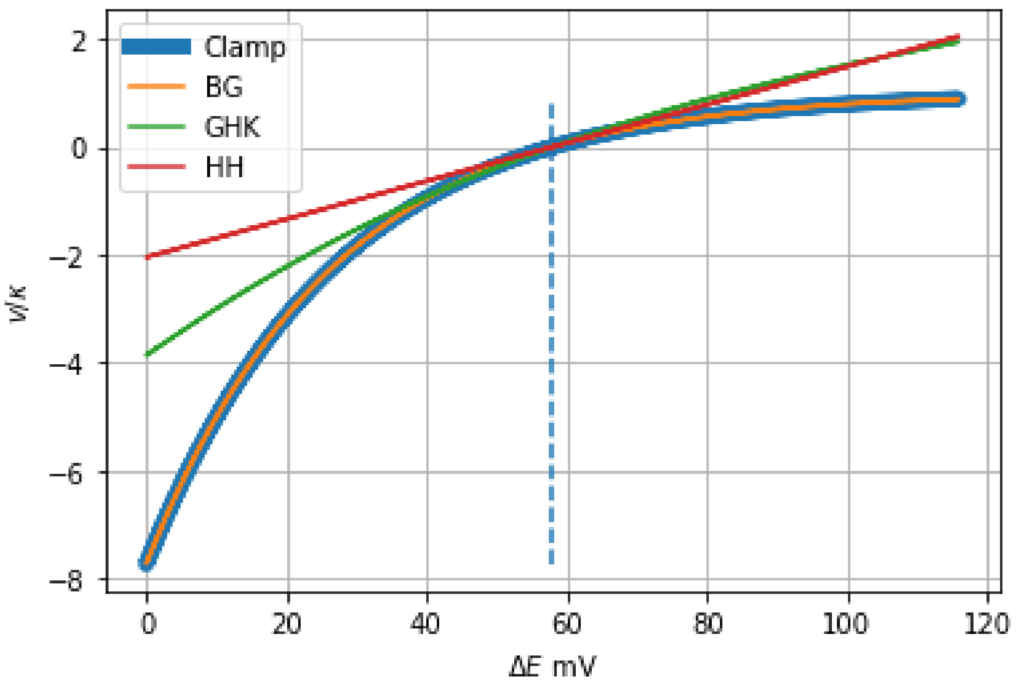
[17]:
```
## Stoichiometry
##chemostats = ['Ii','Ie','Ei','Ee']
chemostats = chemostats + ['Ei','Ee']
scc = st.statify(s,chemostats=chemostats)
#print(s['species'])
```

[18]:
```
X_chemo = {}
V_Nernst = st.V_N()*np.log(conc_e['Na']/conc_i['Na'])
#print(f'V_Nernst = {1000*V_Nernst:4.1f} mV')
T = 100

CV = C*V_Nernst
# x_chemo = f'{CV}*(np.sin({2*np.pi/T}*t))'
# #X_chemo['Ee'] = f'{-CV/2}-'+x_chemo
# X_chemo['E'] = x_chemo
x_chemo = f'-{CV}*(1+1.0*np.sin({2*np.pi/T}*t))'
X_chemo['Ee'] = x_chemo


#print(X_chemo)

dat = Simulate(s,scc,T=T,X_chemo=X_chemo)
#st.plot(s,dat)
st.plot(s,dat,species=['Ei','Ee'])
```

```
Na
dV = 57.9mV
dV_Theory = 57.8mV
```



[19]:
```python
def PlotClamp():
    t = dat['t']
    phi_Ei = dat['phi'][:,s['species'].index('Ei')]
    phi_Ee = dat['phi'][:,s['species'].index('Ee')]
    x_Ii = dat['X'][:,s['species'].index('Ii')]
    x_Ie = dat['X'][:,s['species'].index('Ie')]
    v = dat['V'][:,s['reaction'].index('r')]
    V_N = st.V_N()

    dV = phi_Ei-phi_Ee

    ## BG
    v_BG = (1/x_Ii)*(x_Ii - x_Ie*np.exp(-dV/V_N))

    ## GHK
    v_GHK = 0.5*v_BG*(dV/V_N)/(1-np.exp(-dV/V_N))

    ## HH
    v_HH = (np.exp(-V_Nernst))*(dV - V_Nernst)/V_N

    plt.plot(dV*1000,v,label='Clamp',lw = 6)
    plt.plot(dV*1000,v_BG,label='BG')
    plt.plot(dV*1000,v_GHK,label='GHK')
    plt.plot(dV*1000,v_HH,label='HH')
    plt.vlines(1000*V_Nernst,min(v),max(v),linestyle='dashed')
    plt.grid()
    plt.legend()
    plt.xlabel('$\Delta E$ mV')
    plt.ylabel('$v/\kappa$')
    plt.savefig('Figs/clamp.pdf')

PlotClamp()
```

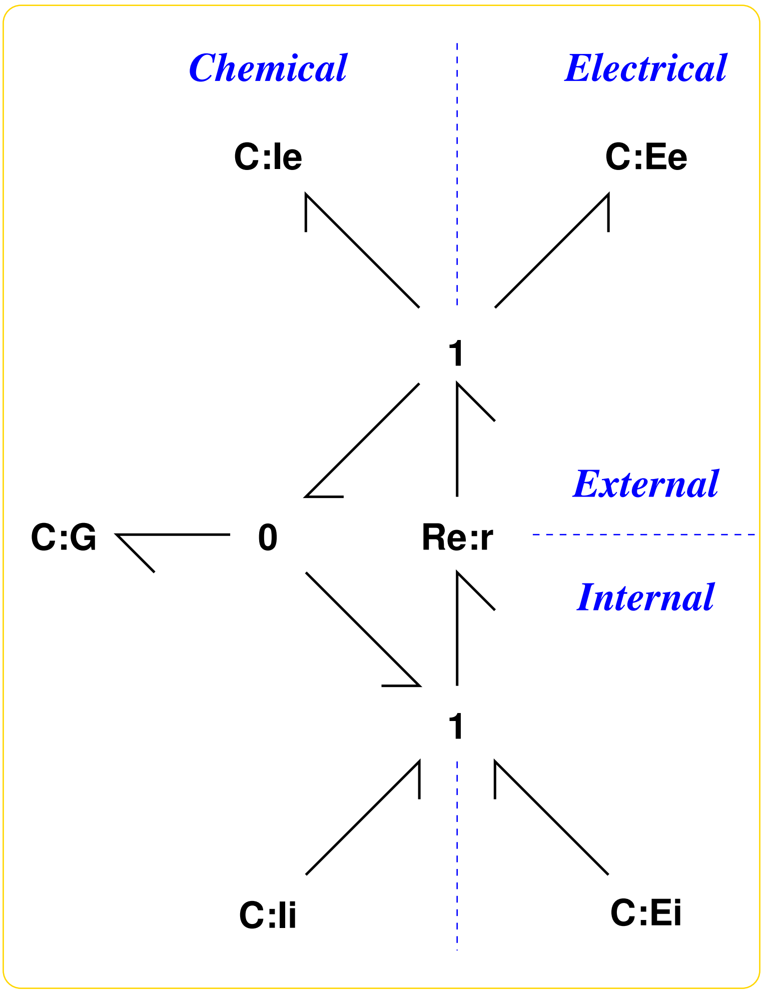


```
[ ]:
```

3 Gated ion channel

```
[20]: ## Ion Channel
      sbg.model('IonChannel_abg.svg')
      import IonChannel_abg
      disp.SVG('IonChannel_abg.svg')
```

```
[20]:
```



[21]:
```
## Stoichiometry
s = st.stoich(IonChannel_abg.model(),linear=['Ei','Ee'],quiet=quiet)
if Fix_conc:
    chemostats = ['Ii','Ie','G']
else:
    chemostats = ['G']
sc = st.statify(s,chemostats=chemostats)
print(s['species'])
```

```
['Ee', 'Ei', 'G', 'Ie', 'Ii']
```

[22]:
```
## Reactions
disp.Latex(st.sprintrl(s,chemformula=True,all=True))
```

[22]:

$$\mathrm{Ei} + \mathrm{G} + \mathrm{Ii} \xrightleftharpoons{r} \mathrm{Ee} + \mathrm{G} + \mathrm{Ie} \tag{27}$$

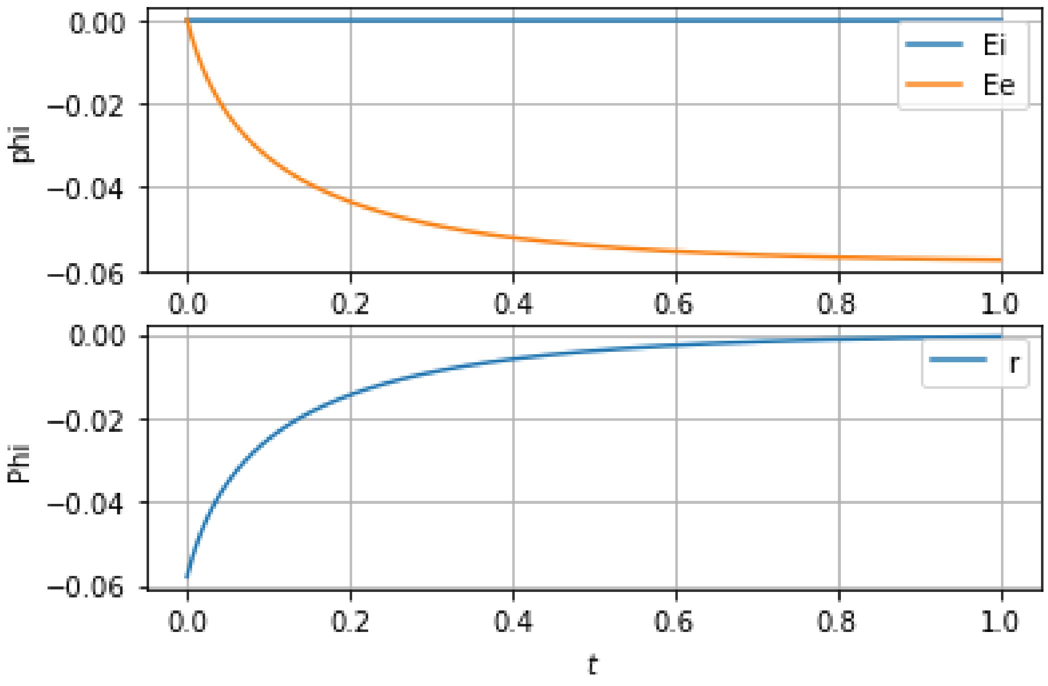
[23]:
```
## Flows
disp.Latex(st.sprintvl(s))
```

[23]:

$$v_r = K_G \kappa_r x_G \left(-K_{Ie} x_{Ie} e^{\frac{K_{Ee} x_{Ee}}{V_N}} + K_{Ii} x_{Ii} e^{\frac{K_{Ei} x_{Ei}}{V_N}} \right) \tag{28}$$

[24]:
```
X_chemo = {'G':'0.1'}
dat = Simulate(s,sc,X_chemo=X_chemo)
#st.plot(s,dat)
st.plot(s,dat,plotPhi=True,species=['Ei','Ee'])
```

```
Na
dV = 57.4mV
dV_Theory = 57.9mV
```



4 Interacting ion channels

Two instances of the ion channel module are combined; one corresponds to Na^+ and one to K^+. The species concentations are encapsulated in the individual modules, but the electrical capaciter are shared. This is a simplified version of the Hodgkin-Huxley model of the squid giant axon and the correponding Na^+ and K^+ concentrations are used.

The simulations use piecewise constant gating variables G_{Na} and G_K:

$$G_K = \begin{cases} 10^{-6} & \text{for } 0.3 < t < 0.35 \\ 1 & \text{otherwise} \end{cases} \tag{29}$$

$$G_{Na} = \begin{cases} 1 & \text{for } 0.3 < t < 0.35 \\ 4.3 \times 10^{-3} & \text{otherwise} \end{cases} \tag{30}$$

The time course of the membrane potential ΔE can be explained as follows.

$t < 0.3$ ΔE moves from the initial condition of zero to a *resting potential* of about -65mV.

This corresponds to the value in Table 2.1 of Keener & Sneyd; the resting potential depends not only on Nernst potentials of chNa+ and K^+ (which in turn depends on the concentrations) but also on the values of the gating potential.

$0.3 < t < 0.35$ ΔE undergoes a typical action potential as the Na^+ gate opens and moves toward the Nernst potential for Na^+ until the gate closes.

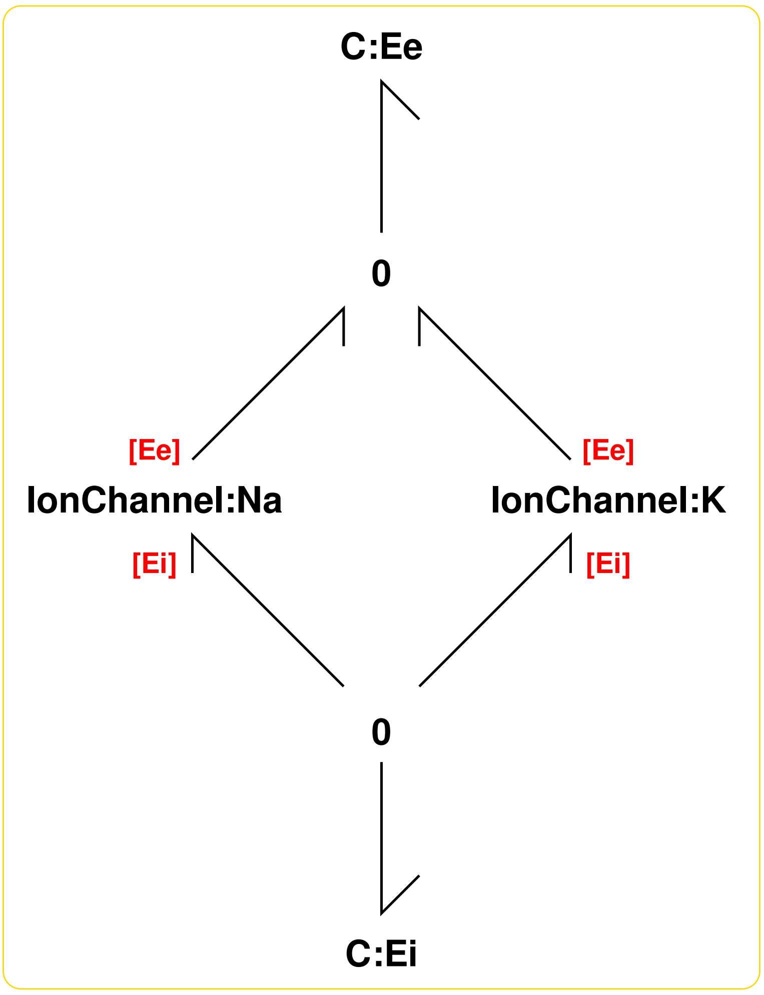
$t > 0.35$ ΔE returns to the resting potential.

In this simple example the gating variables G_{Na} and G_K are independent variables, in reality, and in the HH model, the gating variables are modulated by the membrane potential ΔE. This is discussed in a bond graph context by Gawthrop et al. [2017].

[25]:
```
## Ion Channels
sbg.model('IonChannels_abg.svg')
import IonChannels_abg
disp.SVG('IonChannels_abg.svg')
```

```
Creating subsystem: IonChannel:K
Creating subsystem: IonChannel:Na
```

[25]:



```
[26]: ## Stoichiometry
      s = st.stoich(IonChannels_abg.model(),linear=['Ei','Ee'],quiet=quiet)
      if Fix_conc:
          chemostats = ['Na_Ii','Na_Ie','Na_G', 'K_Ii','K_Ie','K_G']
      else:
          chemostats = ['Na_G','K_G']
      sc = st.statify(s,chemostats=chemostats)
      print(s['species'])
```

```
['Ee', 'Ei', 'K_G', 'K_Ie', 'K_Ii', 'Na_G', 'Na_Ie', 'Na_Ii']
```

```
[27]: ## Reactions
      disp.Latex(st.sprintrl(s,chemformula=True,all=True))
```

[27]:

$$\mathrm{Ei + K_G + K_Ii} \xrightleftharpoons{\mathrm{K_r}} \mathrm{Ee + K_G + K_Ie} \tag{31}$$

$$\mathrm{Ei + Na_G + Na_Ii} \xrightleftharpoons{\mathrm{Na_r}} \mathrm{Ee + Na_G + Na_Ie} \tag{32}$$

```
[28]: ## Flows
      disp.Latex(st.sprintvl(s))
```

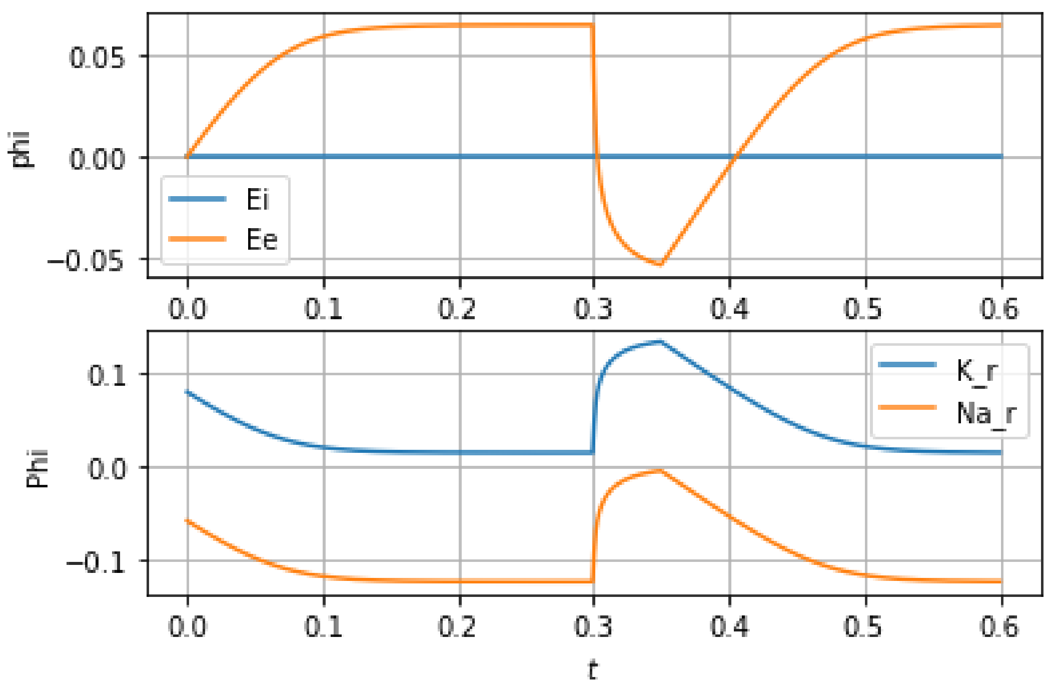
[28]:

$$v_{Kr} = K_{KG}\kappa_{Kr}x_{KG}\left(-K_{KIe}x_{KIe}e^{\frac{K_{Ee}x_{Ee}}{V_N}} + K_{KIi}x_{KIi}e^{\frac{K_{Ei}x_{Ei}}{V_N}}\right) \tag{33}$$

$$v_{Nar} = K_{NaG}\kappa_{Nar}x_{NaG}\left(-K_{NaIe}x_{NaIe}e^{\frac{K_{Ee}x_{Ee}}{V_N}} + K_{NaIi}x_{NaIi}e^{\frac{K_{Ei}x_{Ei}}{V_N}}\right) \tag{34}$$

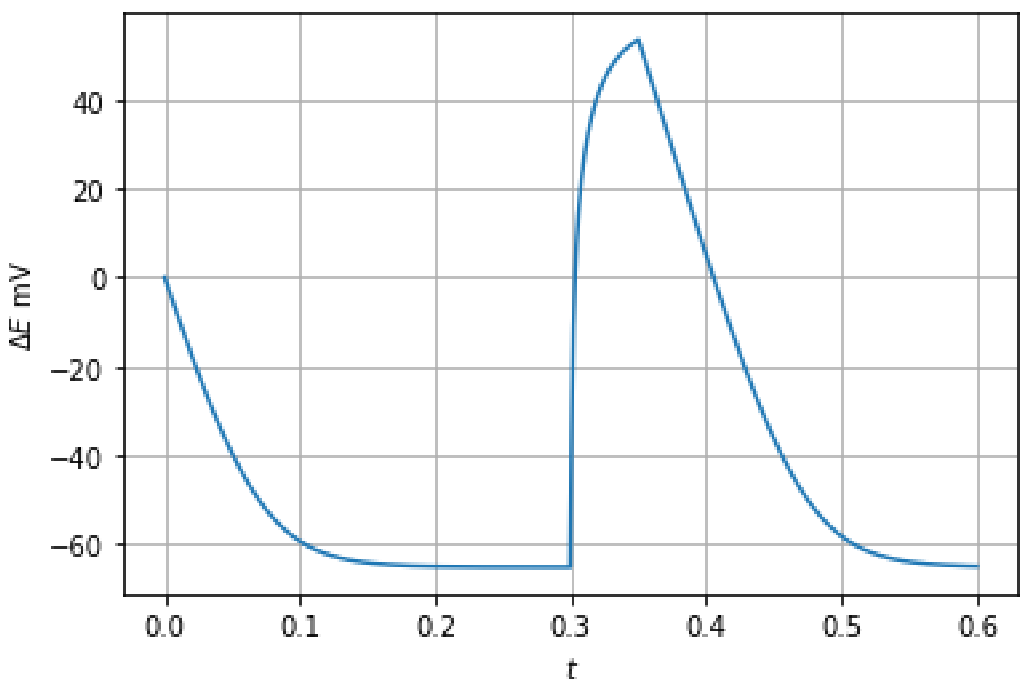
```
[29]: t0_Na = 0.3
      t1_Na = 0.35
      t0_K = 0.35
      t1_K = 1.0

      G_K_0 = 1e-6
      G_Na_0 = 4.3e-3

      G_K = f'{G_K_0}+np.heaviside(t,1)-np.heaviside(t-{t0_Na},1)+np.
       ↪heaviside(t-{t0_K},1)-np.heaviside(t-{t1_K},1)'
      G_Na = f'{G_Na_0}+np.heaviside(t-{t0_Na},1)-np.heaviside(t-{t1_Na},1)'
      # G_K = '1e-1'
      # G_Na = '1e-6'
      X_chemo = {'Na_G':G_Na,'K_G':G_K}
      dat = Simulate(s,sc,X_chemo=X_chemo,prefix=['Na_','K_'],T=0.6)
      #st.plot(s,dat)
      st.plot(s,dat,plotPhi=True,species=['Ei','Ee'])
```

```
Na
K
```

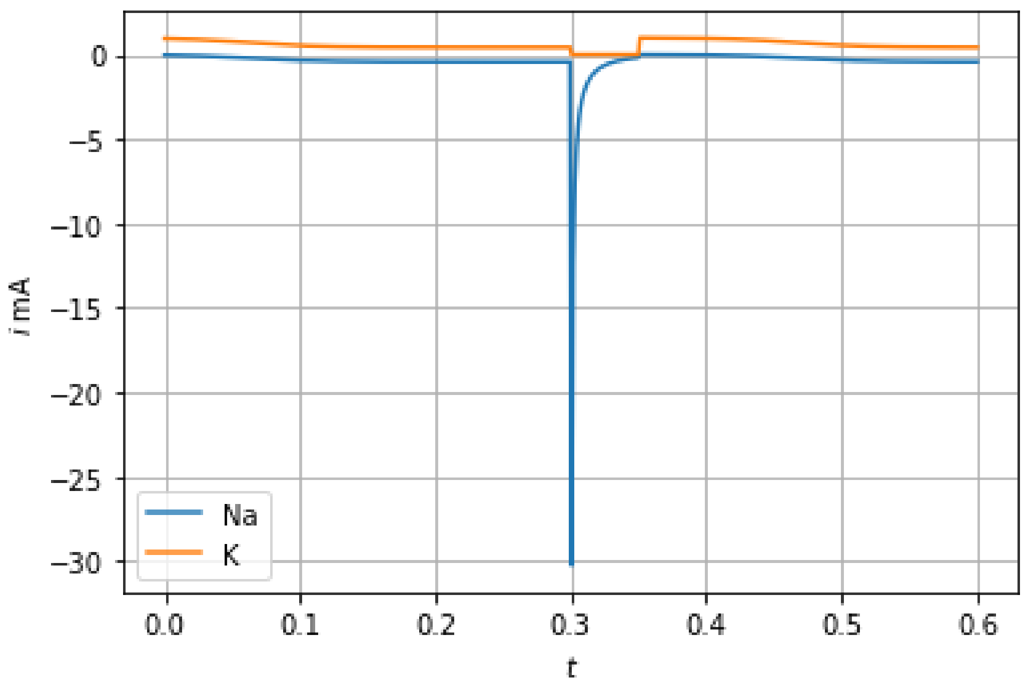


[30]:
```
def PlotAction():
    t = dat['t']
    phi_Ei = dat['phi'][:,s['species'].index('Ei')]
    phi_Ee = dat['phi'][:,s['species'].index('Ee')]
    dE = phi_Ei-phi_Ee

    print(f'Resting potential = {1000*dE[-1]:.2f} mV')

    X_G_K = dat['X'][:,s['species'].index('K_G')]
    X_G_Na = dat['X'][:,s['species'].index('Na_G')]

    v_Na = dat['V'][:,s['reaction'].index('Na_r')]
    v_K  = dat['V'][:,s['reaction'].index('K_r')]


    conc_Na_e = K_Ie*dat['X'][:,s['species'].index('Na_Ie')]
    conc_Na_i = K_Ii*dat['X'][:,s['species'].index('Na_Ii')]
    conc_Na_e_0 = conc_Na_e[0]
    conc_Na_i_0 = conc_Na_i[0]

    conc_K_e = K_Ie*dat['X'][:,s['species'].index('K_Ie')]
    conc_K_i = K_Ii*dat['X'][:,s['species'].index('K_Ii')]
    conc_K_e_0 = conc_K_e[0]
    conc_K_i_0 = conc_K_i[0]

    plt.plot(t,1000*dE)
    plt.grid()
    plt.ylabel('$\Delta E$ mV')
    plt.xlabel('$t$')
    plt.savefig('Figs/action.pdf')
    plt.show()

    plt.plot(t,v_Na,label='Na')
    plt.plot(t,v_K,label='K')
    plt.grid()
    plt.legend()
    plt.ylabel('$i$ mA')
    plt.xlabel('$t$')
    plt.savefig('Figs/action_current.pdf')
    plt.show()

    plt.plot(t,100*(conc_Na_e-conc_Na_e_0)/conc_Na_e_0,label='Na_e')
    plt.plot(t,100*(conc_Na_i-conc_Na_i_0)/conc_Na_i_0,label='Na_i')
    plt.plot(t,100*(conc_K_e-conc_K_e_0)/conc_K_e_0,label='K_e')
    plt.plot(t,100*(conc_K_i-conc_K_i_0)/conc_K_i_0,label='K_i')

    plt.legend()
    plt.grid()
    plt.ylabel(r'$\Delta c (\%)$')
    plt.xlabel('$t$')
    plt.savefig('Figs/action_conc.pdf')
    plt.show()

    plt.plot(t,X_G_K,label='G_K')
    plt.plot(t,X_G_Na,label='G_Na')
    plt.legend()
    plt.grid()
    plt.ylabel('Gating')
    plt.xlabel('$t$')
    plt.savefig('Figs/action_gating.pdf')
    plt.show()



PlotAction()
```

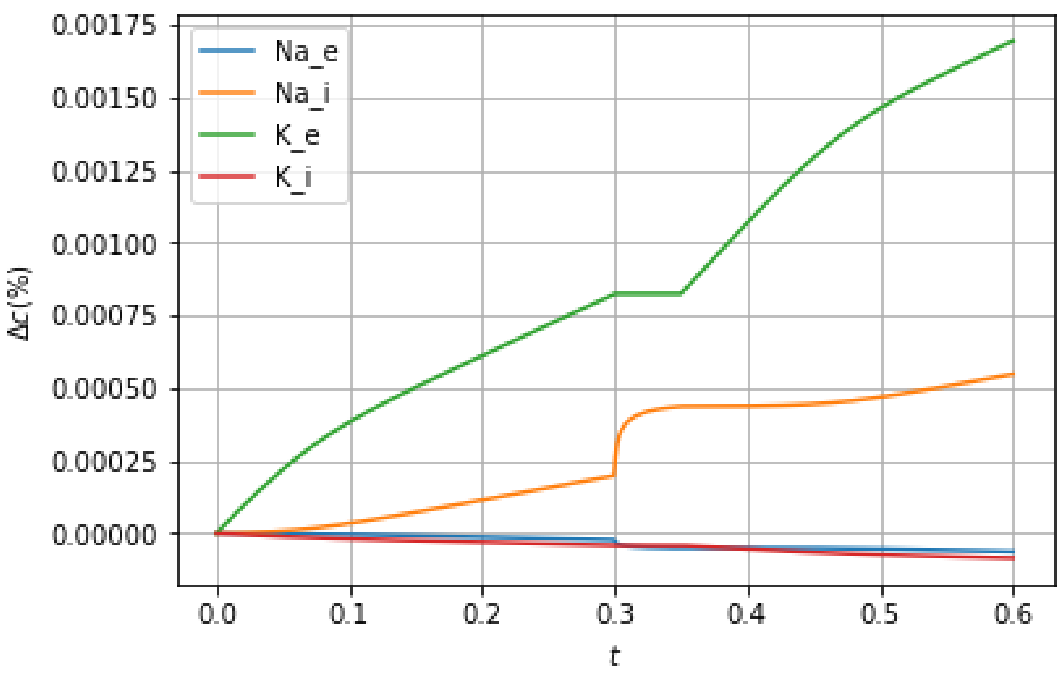
```
Resting potential = -64.90 mV
```

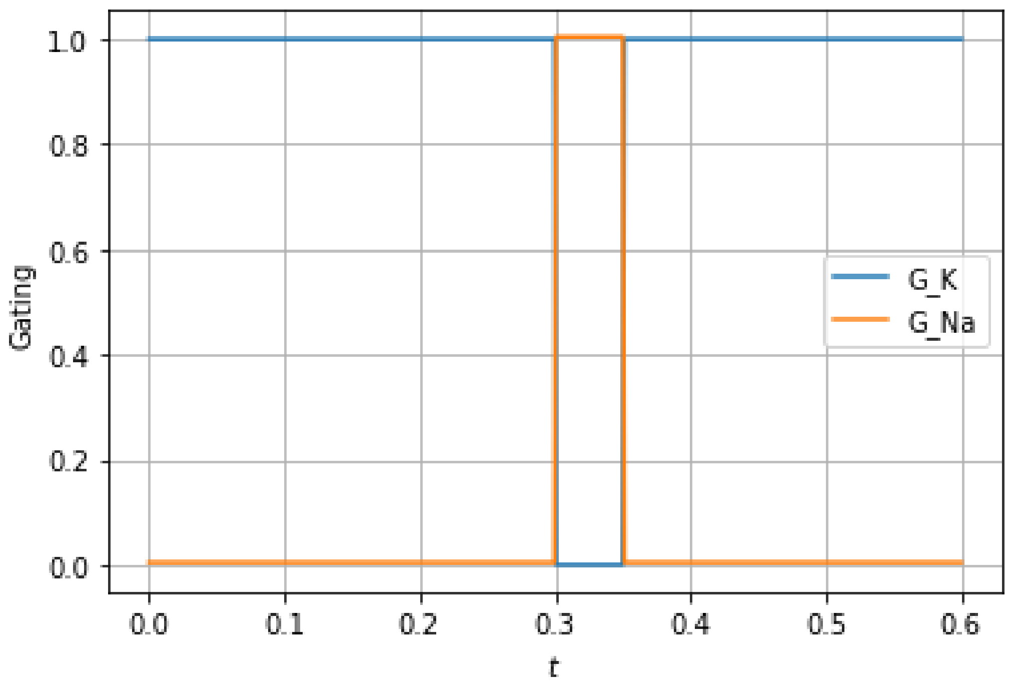


40
20
0
−20
−40
−60
ΔE mV
0.0
0.1
0.2
0.3
0.4
0.5
0.6
t



0
−5
−10
−15
−20
−25
−30
i mA
Na
K
0.0
0.1
0.2
0.3
0.4
0.5
0.6
t





References

P. J. Gawthrop. Bond graph modeling of chemiosmotic biomolecular energy transduction. *IEEE Transactions on NanoBioscience*, 16(3):177–188, April 2017. ISSN 1536-1241. doi: 10.1109/TNB.2017.2674683. Available at arXiv:1611.04264.

P. J. Gawthrop, I. Siekmann, T. Kameneva, S. Saha, M. R. Ibbotson, and E. J. Crampin. Bond graph modelling of chemoelectrical energy transduction. *IET Systems Biology*, 11(5):127–138, 2017. ISSN 1751-8849. doi: 10.1049/iet-syb.2017.0006. Available at arXiv:1512.00956.

Peter J. Gawthrop and Edmund J. Crampin. Energy-based analysis of biochemical cycles using bond graphs. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, 470(2171):1–25, 2014. doi: 10.1098/rspa.2014.0459. Available at arXiv:1406.2447.

Peter J. Gawthrop and Michael Pan. Network thermodynamical modelling of bioelectrical systems: A bond graph approach. Available at arXiv:2009.02217, 2020.

Dean Karnopp. Bond graph models for electrochemical energy storage : electrical, chemical and thermal effects. *Journal of the Franklin Institute*, 327(6):983 – 992, 1990. ISSN 0016-0032. doi: 10.1016/0016-0032(90)90073-R.

George Oster, Alan Perelson, and Aharon Katchalsky. Network thermodynamics. *Nature*, 234: 393–399, December 1971. doi: 10.1038/234393a0.

George F. Oster, Alan S. Perelson, and Aharon Katchalsky. Network thermodynamics: dynamic modelling of biophysical systems. *Quarterly Reviews of Biophysics*, 6(01):1–134, 1973. doi: 10.1017/S0033583500000081.