Cyclic Flow Modulation

Peter Gawthrop (peter.gawthrop@unimelb.edu au)

September 24, 2020

Contents

Note: this is the CyclicFlowModulation.ipynb notebook. The PDF version "Cyclic Flow Modulation" is available here.

1 Introduction

The reaction F_6P + ATP $\overset{PFK}{\rightleftharpoons}$ $F_{16}P$ + ADP catalysed by the enzyme PFK is a key step in glycolysis where:

- PFK phosphofructokinase
- F_6P fructose-6-phosphate
- $F_{16}P$ fructose-1,6-biphosphate

As pointed out by (Cornish-Bowden, 2013), section 12.1.1., the PFK-catalysed reaction forms a cycle with the reaction: $F_{16}P$ + H_2O $\overset{FBP}{\rightleftharpoons}$ F_6P + Pi where:

- FBP fructose biphosphatase
- Pi inorganic phosphate

This cycle is *modulated* by a number of species which simultaneously activate the PFK reaction and inhibit the FBP reaction or *vice-versa*.
(Cornish-Bowden, 2013) [section 12.1.1], (Garrett and Grisham, 2017) [sections 18.3c,22.1 (3), 22.2a]. Indeed (Garrett and Grisham, 2017) [section 22.2b] explicitly states that "substrate cycles provide metabolic control mechanisms".
The species which activate PFK and inhibit FBP include:

- AMP
- $F_{26}P$ fructose-2,6-phosphate

The species which inhibit PFK and activate FBP include:

- ATP
- Cit citrate

Because of the cyclic nature of these two reactions, and the fact that flow is modulated, the term **Cyclic Flow Modulation** (CFM) is used to decribe such reaction systems.

- This note gives a bond graph (Gawthrop and Crampin, 2014) interpretation of such Cyclic Flow Modulation and uses BondGraphTools (Cudmore et al., 2019) to build an analyse a simple example of Cyclic Flow Modulation.
- The note also provides an example of graphical computational modularity where graphical represention in SVG format are converted using svgBondGraph -- see Tutorial svgBondGraph
- A more detaied discussion is found in (Gawthrop, 2020).

1.1 Import some python code

The bond graph analysis uses a number of Python modules:

In [1]:
```
## Some useful imports

import BondGraphTools as bgt
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
#plt.rcParams.update({'font.size': 25})

import IPython.display as disp

## Stoichiometric analysis
import stoich as st

## SVG bg representation conversion
import svgBondGraph as sbg

## Stoichiometry to BG
import stoichBondGraph as stbg


## Modular bond graphs
import modularBondGraph as mbg

## Control systems package
import control as con


## Data structure copy
import copy

## For reimporting: use imp.reload(module)
import importlib as imp

## Saving and loading data
import pickle

## Set quiet=False for verbose output
quiet = True

## Model can be reinitialised by setting True
## If False, processed models read from file
Initialise_model = False
```

```
In /home/peterg/.local/lib/python3.6/site-packages/matplotlib/mpl-data/stylelib/_classic_test.mp
The text.latex.unicode rcparam was deprecated in Matplotlib 3.0 and will be removed in 3.2.
```

```
In /home/peterg/.local/lib/python3.6/site-packages/matplotlib/mpl-data/stylelib/_classic_test.mp
The savefig.frameon rcparam was deprecated in Matplotlib 3.1 and will be removed in 3.3.
In /home/peterg/.local/lib/python3.6/site-packages/matplotlib/mpl-data/stylelib/_classic_test.mp
The pgf.debug rcparam was deprecated in Matplotlib 3.0 and will be removed in 3.2.
In /home/peterg/.local/lib/python3.6/site-packages/matplotlib/mpl-data/stylelib/_classic_test.mp
The verbose.level rcparam was deprecated in Matplotlib 3.1 and will be removed in 3.3.
In /home/peterg/.local/lib/python3.6/site-packages/matplotlib/mpl-data/stylelib/_classic_test.mp
The verbose.fileo rcparam was deprecated in Matplotlib 3.1 and will be removed in 3.3.
```

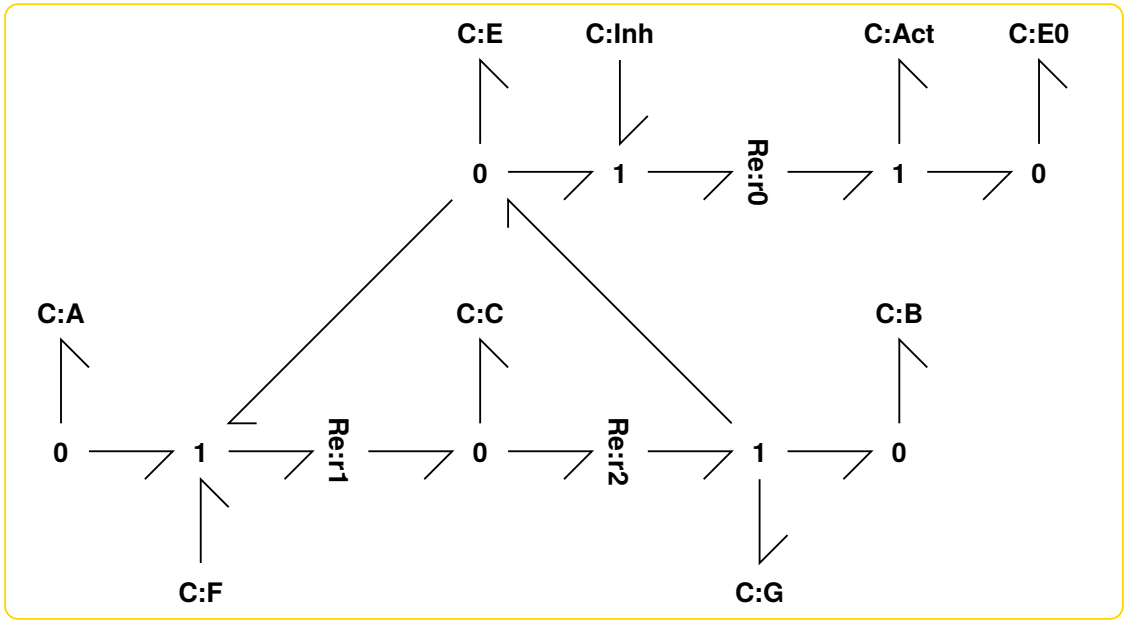
In [2]:
```
def convertBG(name,quiet=True,flatten=True):

    svg = name+'.svg'

    print('Processing', svg)

    ## Convert svg to BGtools and import
    sbg.model(svg,quiet=quiet)
    exec(f'import {name}')
    exec(f'imp.reload({name})')
    if flatten:
        print('    Flattening')
        ## Create stoichiometry
        ss = eval(f'st.stoich({name}.model(),quiet=quiet)')

        ## Create flattened BG
        stbg.model(ss,filename=name)
        exec(f'imp.reload({name})')

    ## Stoichiometry
    print('    Computing stoichiometry')

    s = eval(f'st.stoich({name}.model(),quiet=quiet)')

    return s



Sfilename = 'S.dat'
if Initialise_model:
    S = {} ## Stoichiometry of each system
    names = ['ecr','ECR',
             'CFM']
    TopLevel = []
    #TopLevel = ['mCoop','Pfb','PIfb','Pfb0','PIfb0','Pol','PIol','Pol0','PIol0']
    for name in names:
        flatten = not name in TopLevel
        s = convertBG(name+'_abg', flatten=flatten)
```

```
            S[name] = s

        Sfile = open(Sfilename, 'wb')
        pickle.dump(S, Sfile)
    else:
        Sfile = open(Sfilename, 'rb')
        S = pickle.load(Sfile)
```

2 Modulated Pumped Enzyme Catalysed Reaction

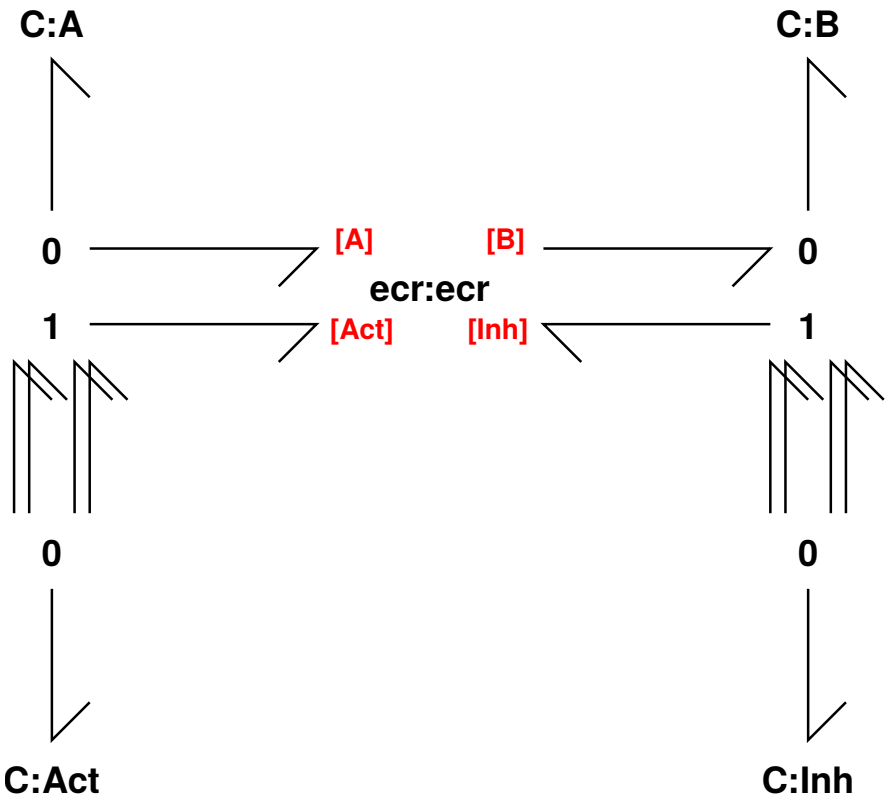
In [3]: `disp.SVG('ecr_abg.svg')`

Out[3]:



In [4]: `disp.SVG('ECR_abg.svg')`

Out[4]:



2.1 Steady-state analysis

The system represented by the bond graph is similar to that of Section 5(a) of (Gawthrop and Crampin, 2014) with the following differences:

- The chemostats F and G pump (or drive) the reaction from A to B; K_F is larger than K_G.
- The amount of enzyme E is modulated by reaction r0 and the activation and inhibition potentials.
- The activation and inhibition is via N=4 bonds corresponding to cooperative binding of N molecules.

The three reactions are:

$$\mathrm{NInh} + \mathrm{E} \overset{\mathrm{r0}}{\rightleftharpoons} \mathrm{NAct} + \mathrm{E}_0 \tag{1}$$

$$\mathrm{A} + \mathrm{E} + \mathrm{F} \overset{\mathrm{r_1}}{\rightleftharpoons} \mathrm{C} \tag{2}$$

$$\mathrm{C} \overset{\mathrm{r_2}}{\rightleftharpoons} \mathrm{B} + \mathrm{E} + \mathrm{G} \tag{3}$$

The three substance E0, E and C form a conserved moiety so that $x_{E0} + x_E + x_C = e_0$ where the constant e_0 is the total amount.

From (Gawthrop and Crampin, 2014), x_C the amount of C is gven by:

$$x_C = \frac{K_e}{K_c}\sigma_v x_e \tag{4}$$

$$\text{where } \sigma_v = \frac{\kappa_1 e^{\frac{\Phi^f}{V_N}} + \kappa_2 e^{\frac{\Phi^r}{V_N}}}{\kappa_1 + \kappa_2} \tag{5}$$

$$\text{and } \Phi^f = K_F K_A x_F x_A \; ; \Phi^r = K_G K_B x_G x_B \tag{6}$$

Using the equilibrium conditions for reaction R0:

$$x_{E0} = \left(K_{IA} x_{IA}\right)^N x_E \tag{7}$$

$$\text{where } x_{IA} = \frac{x_I}{x_A} \tag{8}$$

$$\text{and } K_{IA} = \frac{K_E K_I}{K_{E0} K_A} \tag{9}$$

Using the conserved moiety, it follows that:

$$x_E = \frac{e_0}{1 + \frac{K_e}{K_c}\sigma_v + \left(K_{IA} x_{IA}\right)^N} \tag{10}$$

Following the analysis of (Gawthrop and Crampin, 2014), the stready state reaction flow v associiated with r1 and r2 is:

$$v = \bar{\kappa} \frac{K_e e_0}{1 + \frac{K_e}{K_c}\sigma_v + \left(K_{IA} x_{IA}\right)^N} \Phi \tag{11}$$

$$\text{where } \Phi = \Phi_f - \Phi_r \text{ and } \bar{\kappa} = \frac{\kappa_1 \kappa_2}{\kappa_1 + \kappa_2} \tag{12}$$

The *incremental gain* $\frac{dv}{dx_{IA}}$ is:

$$\frac{dv}{dx_{IA}} = -N K_{IA}^N x_{IA}^{N-1} \bar{\kappa} \frac{K_e e_0}{\left(1 + \frac{K_e}{K_c}\sigma_v + \left(K_{IA} x_{IA}\right)^N\right)^2} \tag{13}$$

Noting that $\phi = \phi^\circ + V_N \ln \frac{x}{x^\circ}$ and so $\frac{d\phi}{dx} = \frac{V_N}{x}$, it follows that:

$$\frac{dv}{d\phi_{IA}} = -N \left(K_{IA} x_{IA}\right)^N \bar{\kappa} \frac{K_e e_0}{V_N \left(1 + \frac{K_e}{K_c}\sigma_v + \left(K_{IA} x_{IA}\right)^N\right)^2} \tag{14}$$

This can be reexpressed in terms of ϕ_{AI} and x_{IA} by noting that $\left(K_{IA} x_{IA}\right)^N = \left(K_{AI} x_{AI}\right)^{-N}$.

```
In [5]: ## Theoretical steady-state flow in modulated enzyme-catalysed reaction
        def mECR_flow(x_A,x_B,x_IA,e0=1,N=4,dphi=True,
                      K_A=1,K_B=1,K_C=1,K_E=1,K_IA=1,
```

```
                K_F=1,K_G=0.1,
                kappa_r1 = 1,kappa_r2=1):
    """Theoretical flows in modulated Enzyme-catalysed Reactions
    """

    kappa_bar = (kappa_r1*kappa_r2)/(kappa_r1+kappa_r2)
    delta = K_A*x_A*K_F - K_B*x_B*K_G
    sigma = (kappa_r1*K_A*x_A*K_F + kappa_r2*K_B*x_B*K_G)/(kappa_r1 + kappa_r2)
    K_m = K_C/K_E

    den = 1 + (sigma/K_m) + (K_IA*x_IA)**N
    v = kappa_bar*e0*K_E*delta/den
    dv = -N*(K_IA**N)*(x_IA**(N-1))*v/den
    if dphi: # Compute dv/dphi
        dv *= x_IA/st.V_N()

    return v,dv
```

2.2 Stoichiometry and reactions

In [6]:
```
#s = st.stoich(CFM_abg.model(),quiet=quiet)
s = S['ECR']
print(s['species'])
print(s['reaction'])
chemostats=['A','B','Act','Inh','ecr_F','ecr_G']
sc = st.statify(s,chemostats=chemostats)
disp.Latex(st.sprintrl(s,chemformula=True))
```

```
['A', 'Act', 'B', 'Inh', 'ecr_C', 'ecr_E', 'ecr_E0', 'ecr_F', 'ecr_G']
['ecr_r0', 'ecr_r1', 'ecr_r2']
```

Out[6]:

$$4\,\mathrm{Inh} + \mathrm{ecr_E} \overset{\mathrm{ecr_{r0}}}{\rightleftharpoons} 4\,\mathrm{Act} + \mathrm{ecr_{E0}} \tag{15}$$

$$\mathrm{A} + \mathrm{ecr_E} + \mathrm{ecr_F} \overset{\mathrm{ecr_{r1}}}{\rightleftharpoons} \mathrm{ecr_C} \tag{16}$$

$$\mathrm{ecr_C} \overset{\mathrm{ecr_{r2}}}{\rightleftharpoons} \mathrm{B} + \mathrm{ecr_E} + \mathrm{ecr_G} \tag{17}$$

In [7]:
```
disp.Latex(st.sprintml(sc,chemformula=False))
```

Out[7]:

$$\Leftrightarrow A \tag{18}$$
$$\Leftrightarrow Act \tag{19}$$
$$\Leftrightarrow B \tag{20}$$
$$\Leftrightarrow Inh \tag{21}$$
$$\Leftrightarrow ecr_C + ecr_E + ecr_E 0 \tag{22}$$
$$\Leftrightarrow ecr_F \tag{23}$$
$$\Leftrightarrow ecr_G \tag{24}$$

3 Cyclic Flow Modulation (CFM)

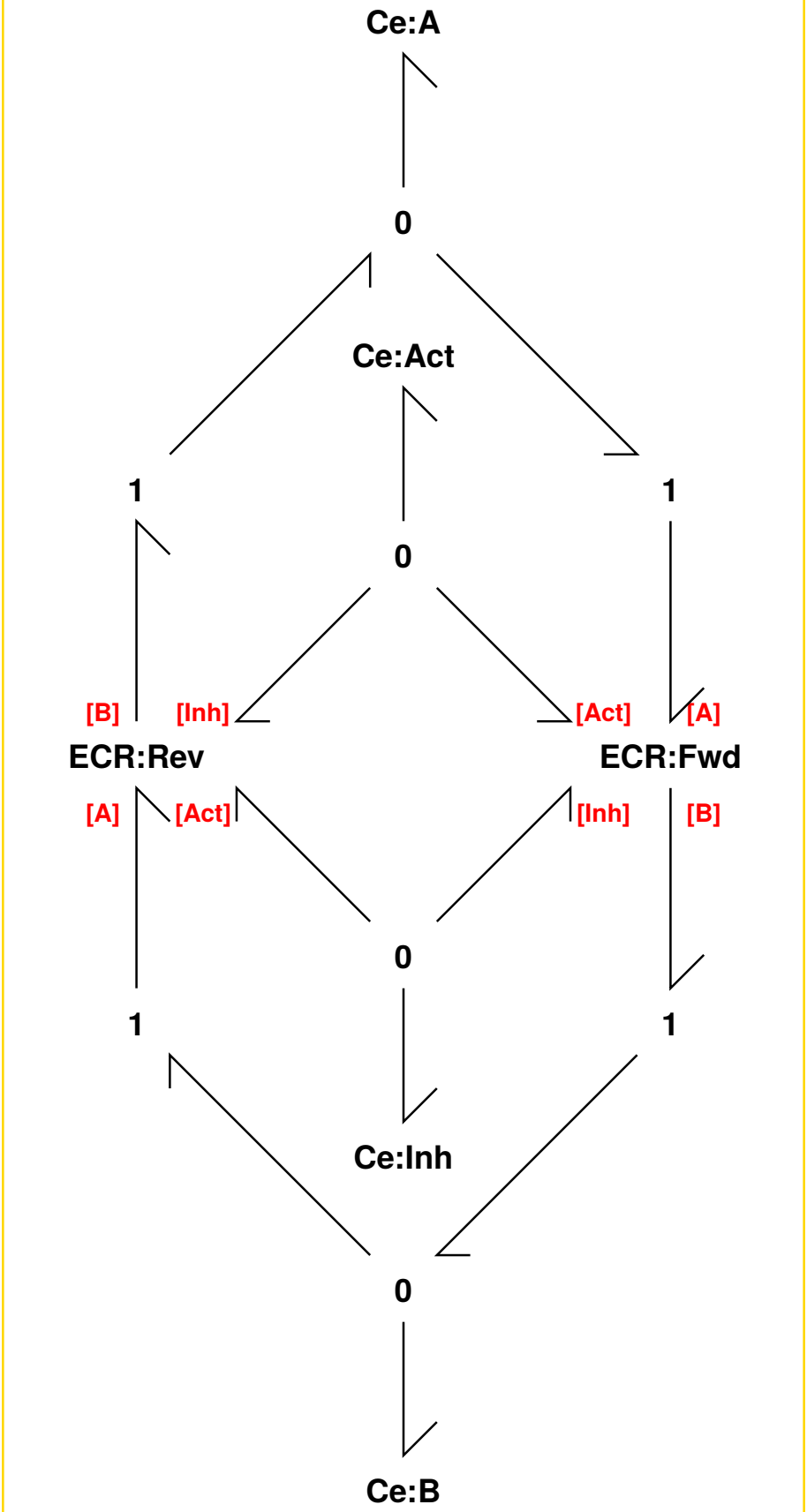
As discussed in the context of the PFK/FBP cycle in the introduction, CFM involves a cycle formed of two modulated enzyme-catalysed reactions. Such a cycle is shown in the following bond graph with the following components and interpretation:

- ECR:Fwd an instance of the ECR representing the forward reaction [PFK]
- ECR:Rev an instance of the ECR representing the reverse reaction [FBP]
- Ce:A The substrate species [F_6P]
- Ce:B The product species [$F_{16}P$]
- Ce:Act The activation species [AMP + $F_{26}P$]
- Ce:Inh The inhibition species [ATP + Cit]

Note that the activator Ce:Act activates ECR:Fwd and inhibits ECR:Rev and Ce:Inh inhibits ECR:Fwd and activates ECR:Rev.

```
In [8]: disp.SVG('CFM_abg.svg')
```

Out[8]:


Ce:A
0
Ce:Act
1
0
1
[B]
[Inh]
[Act]
[A]
ECR:Rev
ECR:Fwd
[A]
[Act]
[Inh]
[B]
0
1
1
Ce:Inh
0
Ce:B

3.1 Steady-state analysis

The net flow out of A in to B is the differtence of the flows in the two ECR components. As activation and inhinition are reversed in the ECR:rev, N is replaced by $-N$.

```
In [9]: ## Theoretical steady-state flow in Cyclic Flow Modulation
        ## Based on theoretical steady-state flow in modulated enzyme-catalysed reaction
        def CFM_flow(x_A,x_B,x_IA,e0=1,N=4,dphi=True,
                     K_A = 1,K_B=1,K_C=1,K_E=1,K_IA=1,
                     K_F=1,K_G=0.1,
                     kappa_r1 = 1,kappa_r2=1,
                     oneway = False,activate=False):
            """Theoretical flows in Cyclic Flow Modulation
            """

            v_F,dv_F = mECR_flow(x_A,x_B,x_IA,e0=e0,N=N,dphi=dphi,
                         K_A=K_A,K_B=K_B,K_C=K_C,K_E=K_E,K_IA=K_IA,
                         K_F=K_F,K_G=K_G,
                         kappa_r1 = kappa_r1,kappa_r2=kappa_r2)

            v_R,dv_R = mECR_flow(x_B,x_A,x_IA,e0=e0,N=-N,dphi=dphi,
                     K_A=K_B,K_B=K_A,K_C=K_C,K_E=K_E,K_IA=1/K_IA,
                     K_F=K_F,K_G=K_G,
                     kappa_r1=kappa_r1,kappa_r2=kappa_r2)

            if oneway:
                v = v_F
                dv = dv_F
            else:
                v = v_F - v_R
                dv = dv_F - dv_R

            return v,dv
```

3.2 Stoichiometry and reactions

```
In [10]: #s = st.stoich(CFM_abg.model(),quiet=quiet)
         s = S['CFM']
         print(s['species'])
         print(s['reaction'])
         chemostats=['A','B','Act','Inh']
         chemostats += ['Fwd_ecr_F','Fwd_ecr_G','Rev_ecr_F','Rev_ecr_G']
         sc = st.statify(s,chemostats=chemostats)
         disp.Latex(st.sprintrl(s,chemformula=True))
```

```
['A', 'Act', 'B', 'Inh', 'Fwd_ecr_C', 'Fwd_ecr_E', 'Fwd_ecr_E0', 'Fwd_ecr_F', 'Fwd_ecr_G', 'Rev_
['Fwd_ecr_r0', 'Fwd_ecr_r1', 'Fwd_ecr_r2', 'Rev_ecr_r0', 'Rev_ecr_r1', 'Rev_ecr_r2']
```

Out[10]:

$$4\,\mathrm{Inh} + \mathrm{Fwd_e cr_E} \rightleftharpoons 4\,\mathrm{Act} + \mathrm{Fwd_e cr_{E0}} \quad (25)$$

$$\mathrm{A} + \mathrm{Fwd_e cr_E} + \mathrm{Fwd_e cr_F} \rightleftharpoons \mathrm{Fwd_e cr_C} \quad (26)$$

$$\mathrm{Fwd_e cr_C} \rightleftharpoons \mathrm{B} + \mathrm{Fwd_e cr_E} + \mathrm{Fwd_e cr_G} \quad (27)$$

$$4\,\mathrm{Act} + \mathrm{Rev_e cr_E} \rightleftharpoons 4\,\mathrm{Inh} + \mathrm{Rev_e cr_{E0}} \quad (28)$$

$$\mathrm{B} + \mathrm{Rev_e cr_E} + \mathrm{Rev_e cr_F} \rightleftharpoons \mathrm{Rev_e cr_C} \quad (29)$$

$$\mathrm{Rev_e cr_C} \rightleftharpoons \mathrm{A} + \mathrm{Rev_e cr_E} + \mathrm{Rev_e cr_G} \quad (30)$$

4 Simulation of Steady-state properties

The steady state properties are investigated using dynamic simulation where slowly varing exogenous quantities are used to induce quasi-steady-state behaviour. In each case, the variable is at a constant value to start with followed by a slowly increasing ramp. The response after the initial reponse is plotted to remove artefacts due to the initial transient.

4.1 Set up some parameters for simulation

In [11]:
```
## Set up some parameters for simulation
def setParameter(oneway=False):
    ## Set up the non-unit parameters and states

    parameter = {}

    FwdRev = ['Fwd','Rev']

    ## Reactions
    I = ['0','1','2']
    for fr in FwdRev:
        for i in I:
            Kappa_i = 'kappa_'+fr+'_ecr_r'+i
            if oneway and (fr is 'Rev'):
                parameter[Kappa_i] = 0
            else:
                parameter[Kappa_i] = kappa

    ## Species
    for fr in FwdRev:
        K_i = 'K_'+fr+'_ecr'
```

```
            parameter[K_i+'_E'] = K_E
            parameter[K_i+'_F'] = K_F
            parameter[K_i+'_G'] = K_G
            parameter[K_i+'_C'] = K_C

        parameter['K_A'] = K_A
        parameter['K_B'] = K_B
        parameter['K_Act'] = K_Act
        parameter['K_Inh'] = K_Inh

        ## States
        X0 = np.ones(s['n_X'])
        species = s['species']
        E =  ['E0','E','C']
        for fr in FwdRev:
            for e in E:
                ee = fr+'_ecr_'+e
                i = species.index(ee)
                X0[i] = e0/len(E)
        X0[species.index('A')] = x0_A

        return parameter,X0

epsilon = 1e-2
K_A = 1
K_B = 1
K_F = 1
K_G = epsilon
K_C = 10
K_E = 1

K_Act = 1
K_Inh = 1

K_IA = K_Inh/K_Act

kappa = 1


e0 = 1
x0_A = 1

parameter,X0 = setParameter()
#print(parameter,X0)
```

4.2 Simulation code

The flow v is a dynamical function of substrate x_A, activation x_{Act}, inhibition x_{Inh} and cooperativity index N. An approximate steady-state is acjieved by varying one of the three concentrations slowly whilst fixing the other two. The following function does this by declaring the varying function species by the string sX, a fixed species with a number of discrete values as sX1 with values XX1 and the other species as sX2 with value X2. N can take on a range of values.

deriv=True gives a plot of the derivative of the flow with respect to ϕ.

```
In [12]: def label(sX1,sX2,X1,X2,Loop=False):

             if Loop:
                 return f'{sX1}={X1}(Loop flow)'
             else:
                 return f'{sX1}={X1}'

         def VaryX(sX='A',sX1='Act',sX2='Inh',Xrange=[1e-2,1e2],XX1=[1],X2=1,K_B=1e-6,
                   IntPar=False,deriv=False,power=False,oneway=False,
                   quiet=True,plotting=True):

             spec = s['species']
             reac = s['reaction']

             ## Time
             t_max = int(1e6)
         #     N_sim = int(1e4)
             N_sim = int(1e4)
             t = np.linspace(0,t_max,N_sim)
             t_0 = 1e-2*t_max
             t_1 = t_max-t_0
             i_max = len(t)
             i_0 = int(i_max*t_0/t_max)
             i_1 = i_max-i_0
             print(i_0,i_1)

             ## Set up the chemostats: vary X
             x_max = Xrange[1]
             x_min = Xrange[0]
             chemo = '{3} + ({0}-{3})*np.heaviside(t-{1},1)*((t-{1})/{2})'.format(x_max,t_0,t_1,
             X_chemo =  {sX:chemo}

             for X1 in XX1:

                 ## Non-unit parameters and states
                 parameter,X0 = setParameter(oneway=oneway)
                 X0[s['spec_index'][sX1]] = X1
                 X0[s['spec_index'][sX2]] = X2
```

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

    ## Extract flows at the chemostatted species
    VV = dat['V']
    dX  = dat['dX']
    dX_A = dX[:,spec.index('A')]
    dX_B = dX[:,spec.index('B')]
    V = dX_B

    V_F = VV[:,reac.index('Fwd_ecr_r2')]
    V_R = VV[:,reac.index('Rev_ecr_r2')]
    V_FR = V_F+V_R


    ## Extract the state being varied
    X = dat['X'][:,s['spec_index'][sX]]

    ## Extract potential being varied
    phi = dat['phi'][:,s['spec_index'][sX]]

    ## Extract power
    P_Re = dat['P_Re']
    p_Re = np.sum(P_Re,axis=1) ## Net dissipation


    lw = 2
    ls = None
    if deriv:
        slope = np.gradient(V[-i_1:],phi[-i_1:])
        plt.semilogx(X[-i_1:],slope,lw=lw,label=label(sX1,sX2,X1,X2),linestyle=ls)
        ylabel = '$dv/d \log_{10}{x}$'

    elif power:
        #plt.semilogx(X[-i_1:],P_Re[-i_1:],lw=lw)
        plt.semilogx(X[-i_1:],p_Re[-i_1:],lw=lw,label=label(sX1,sX2,X1,X2),linestyl
        ylabel = '$P_{Re}$'

    else:
        plt.plot(phi[-i_1:],V[-i_1:],lw=lw,label=label(sX1,sX2,X1,X2),linestyle=ls)
        ylabel = '$v$'



plt.xlabel('$\phi_{'+sX+'}$')
plt.ylabel(ylabel)
plt.legend()
plt.grid()
```

```
        #plt.title('N = '+str(N))


        if plotting:
            filename = f'V_{sX}_{sX1}'
            if deriv:
                filename = 'd'+filename
            if power:
                filename = filename+'_P'
            plt.savefig('Figs/'+filename+'.pdf')

        plt.show()

        return V[-i_1:],X[-i_1:],phi[-i_1:]
```

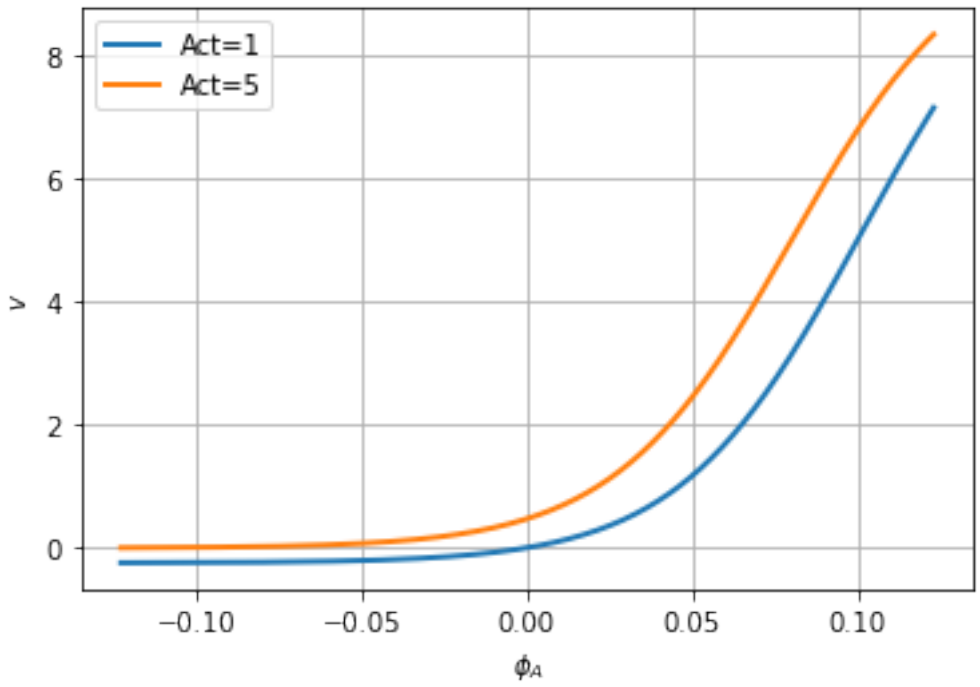
4.3 Vary the substrate concentration.

The substrate concentration x_A is varied for two values of activation x_{Act}.

- dotted lines give the cyclic flow.
- The derivative is also plotted.

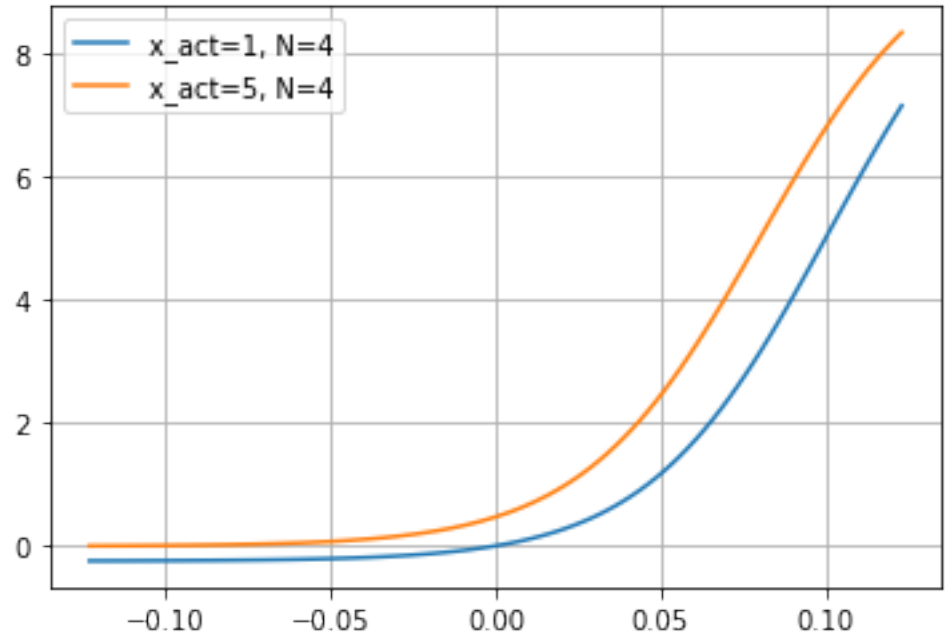
```
In [13]: #print(s['species'])
         Act = [1,5]
         v,x,phi = VaryX(sX='A',sX1='Act',sX2='Inh',XX1=Act,X2=1,oneway=False)
         #dat,x = VaryX(sX='A',sX1='Act',sX2='Inh',XX1=Act,X2=1,deriv=True)
         #dat,x = VaryX(NN=[N],sX='A',sX1='Act',sX2='Inh',XX1=Act,X2=1,power=True)
```

```
100 9900
```



4.3.1 Theory

```
In [14]: X_A = np.logspace(-2,2,100)
         phi_A = st.V_N()*np.log(X_A)
         X_B = 1
         for act in Act:
             for N in [4]:
                 X_IA = 1/act
                 v_theory,dv_theory = CFM_flow(X_A,X_B,X_IA,e0=e0,N=N,
                          K_A=K_A,K_B=K_B,K_C=K_C,K_E=K_E,K_IA=K_IA,
                           K_F=K_F,K_G=K_G,
                          kappa_r1=kappa,kappa_r2=kappa)

                 #slope = np.gradient(v_theory,np.log10(X_A))
                 plt.plot(phi_A,v_theory,label=f'x_act={act}, N={N}')

         plt.grid()
         plt.legend()
         plt.show()
```



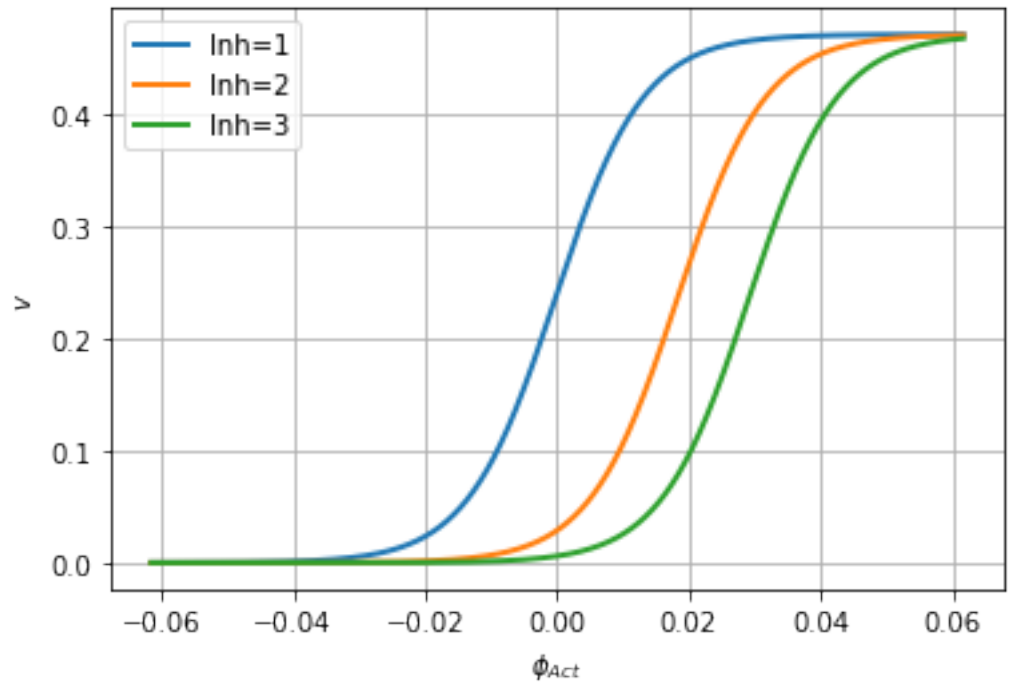
4.4 Vary the activation species concentration.

The activation species concentration x_{Act} is varied for three values of x_{Inh}.

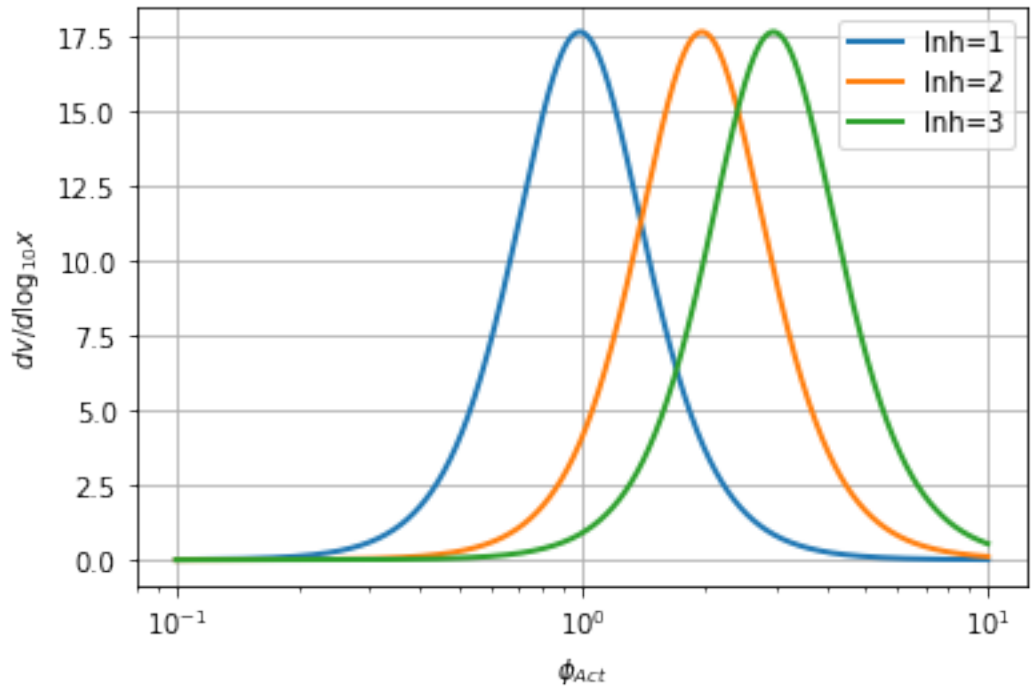
- dotted lines give the cyclic flow.
- The derivative is also plotted.

```
In [15]: Inh = [1,2,3]
         for oneway in [True,False]:
             for deriv in [False,True]:
                 v,x,phi = VaryX(sX='Act',sX1='Inh',sX2='A',XX1=Inh,X2=1,oneway=oneway,Xrange=[0
```

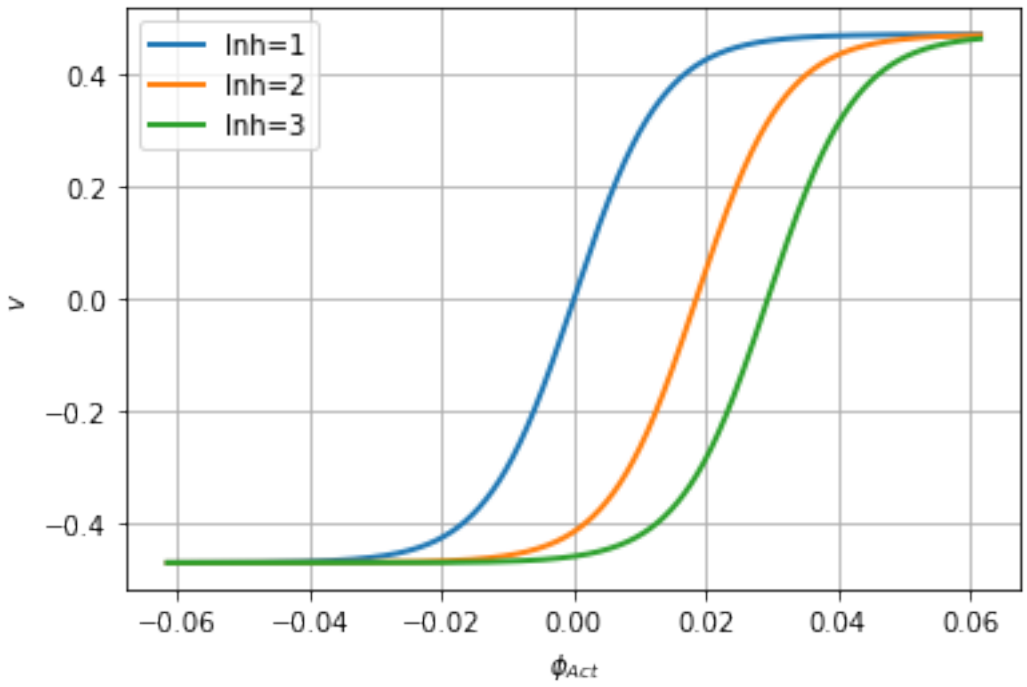
```
100 9900
```


Inh=1
Inh=2
Inh=3
0.4
0.3
0.2
0.1
0.0
v
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
ϕ_{Act}

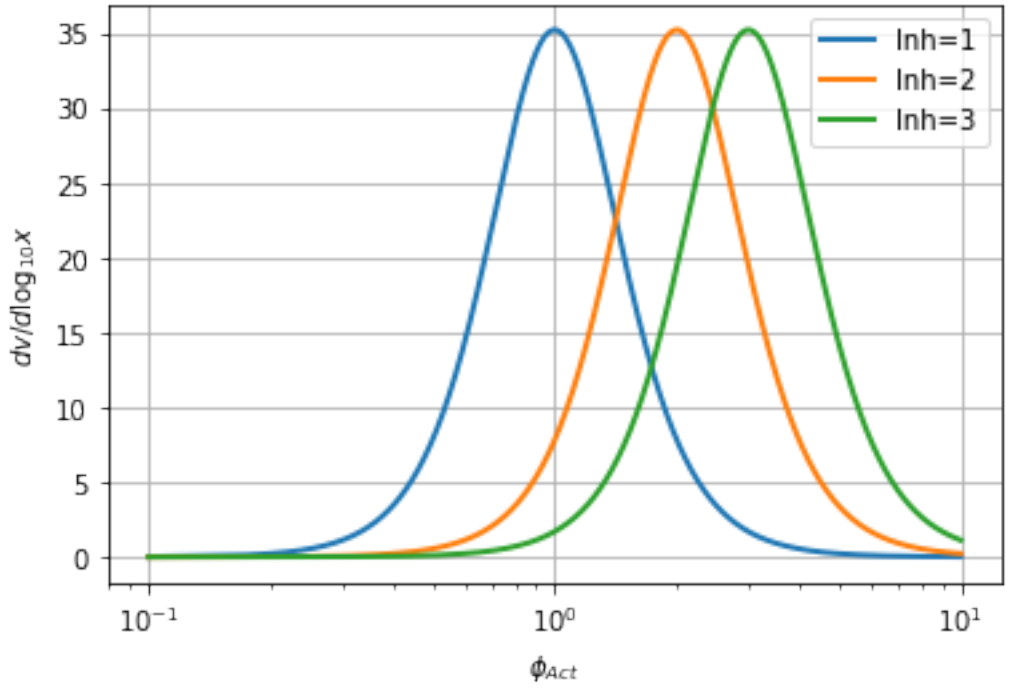
100 9900


Inh=1
Inh=2
Inh=3
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
$dv/d\log_{10}x$
10^{-1}
10^{0}
10^{1}
ϕ_{Act}

100 9900

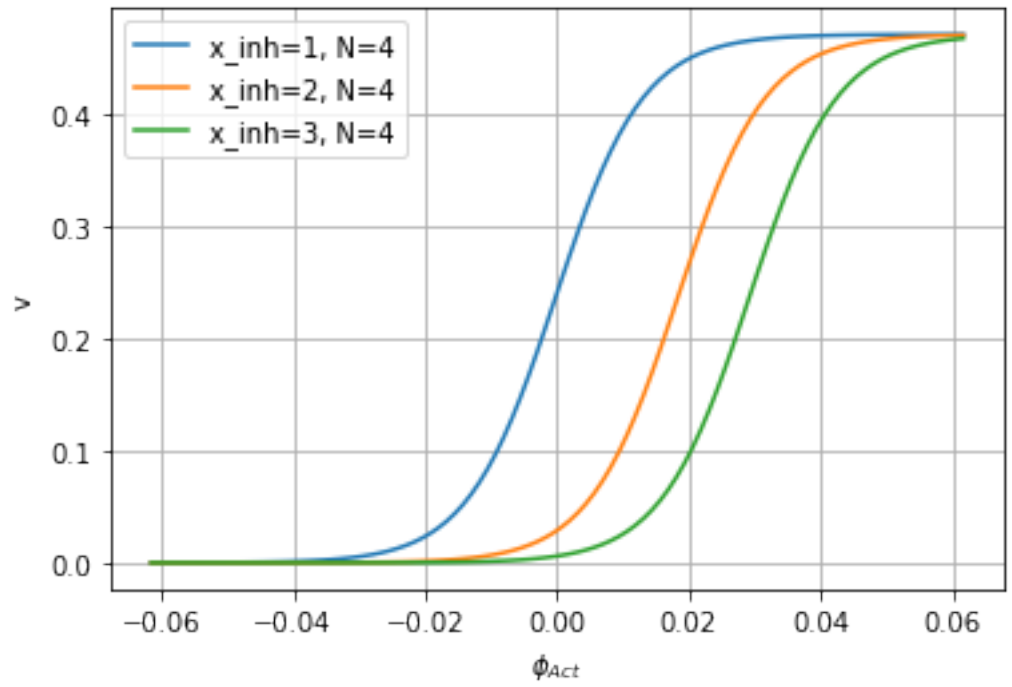


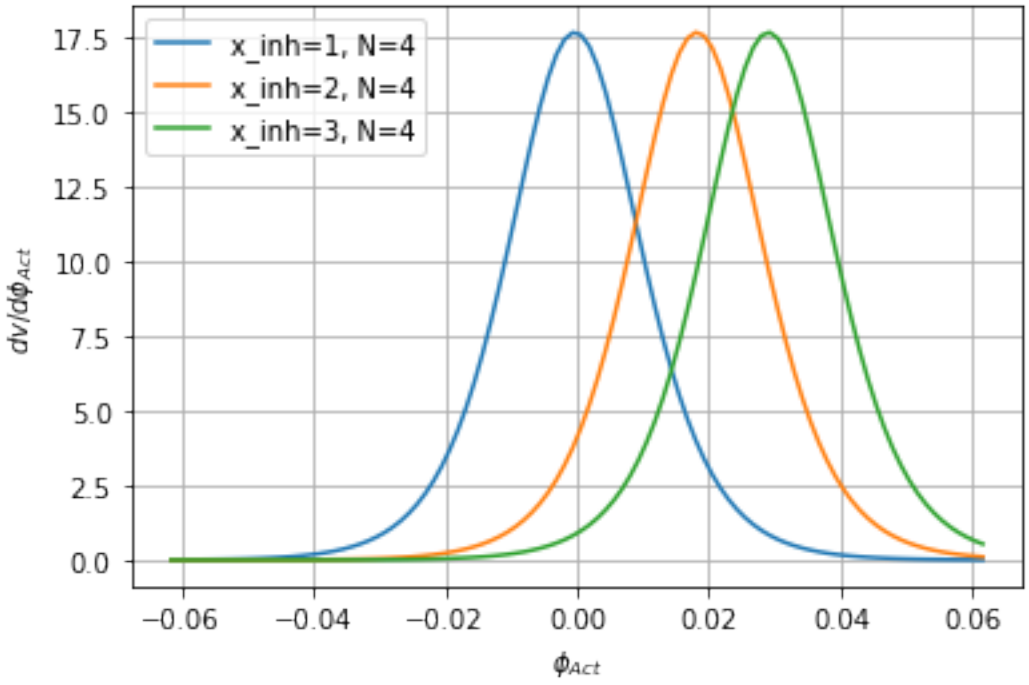
100 9900

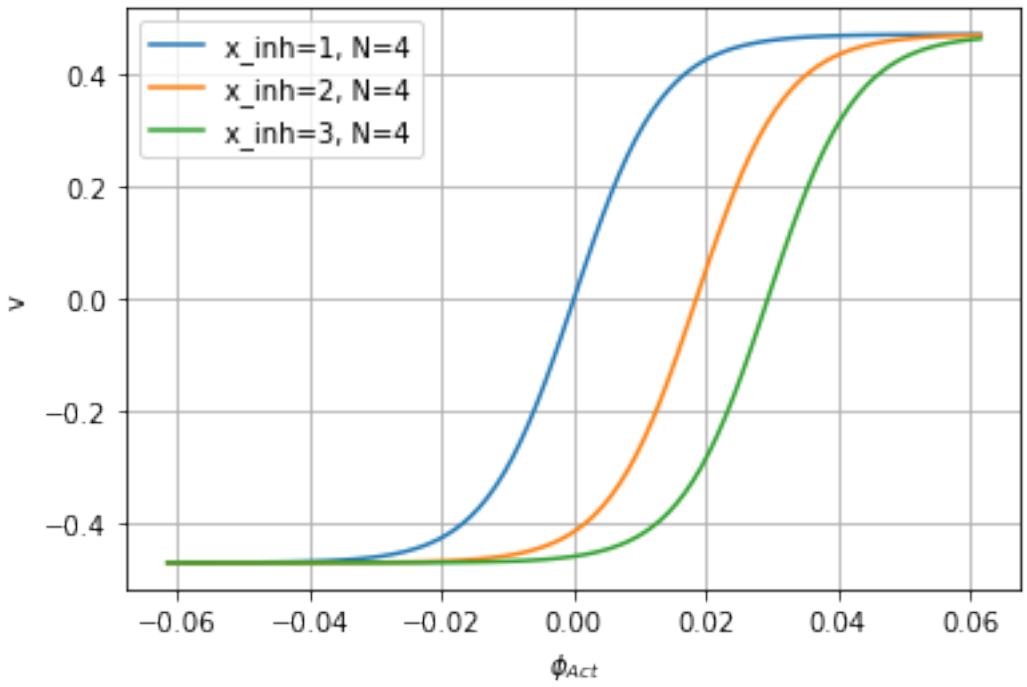


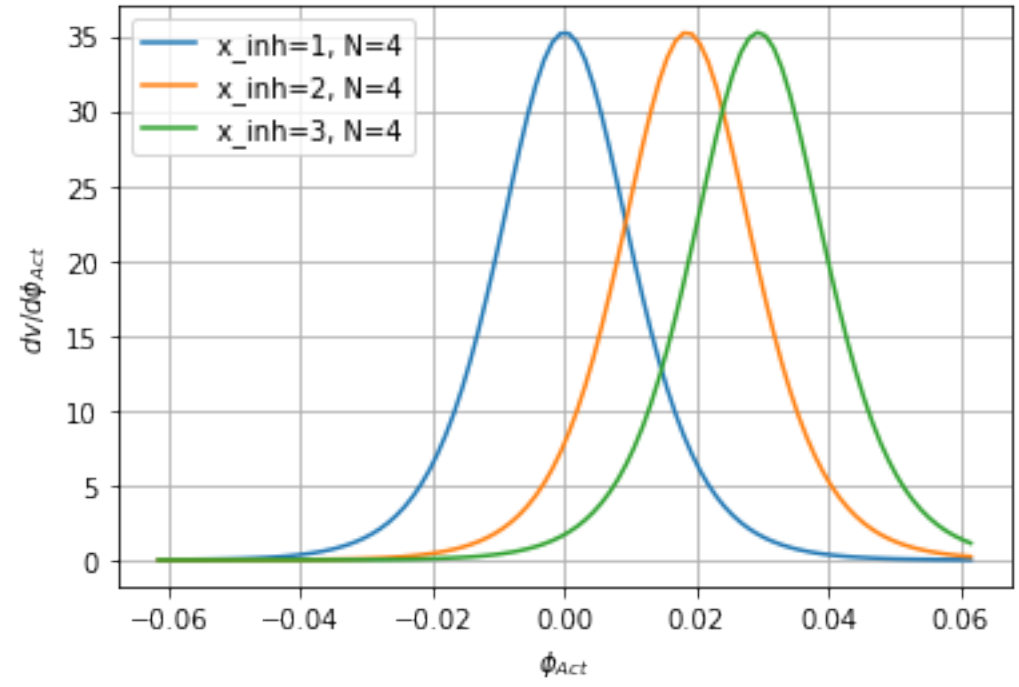
4.4.1 Theory

In [16]:
```
X_Act = np.logspace(-1,1,100)
phi_Act = st.V_N()*np.log(X_Act)

X_A = 1
X_B = 1


for oneway in [True,False]:
    for deriv in [False,True]:
        for inh in Inh:
            for N in [4]:
                X_IA = X_Act/inh
                v_theory,dv_theory = CFM_flow(X_A,X_B,X_IA,e0=e0,N=-N,
                          K_A=K_A,K_B=K_B,K_C=K_C,K_E=1,K_IA=1,
                           K_F=K_F,K_G=K_G,
                          kappa_r1 = 1,kappa_r2=1, oneway=oneway)

                if deriv:
                    slope = np.gradient(v_theory,phi_Act)
                    plt.plot(phi_Act,dv_theory,label=f'x_inh={inh}, N={N}')
                    #plt.plot(phi_Act,dv_theory*X_IA/st.V_N(),label=f'x_inh={inh}',ls='
                else:
                    plt.plot(phi_Act,v_theory,label=f'x_inh={inh}, N={N}')
            if deriv:
                ylabel = '$dv/d\phi_{Act}$'
            else:
                ylabel = 'v'
            plt.ylabel(ylabel)
            plt.xlabel('$\phi_{Act}$')
            plt.grid()
            plt.legend()
            plt.show()
```


x_inh=1, N=4
x_inh=2, N=4
x_inh=3, N=4
v
0.0
0.1
0.2
0.3
0.4
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
ϕ_{Act}


x_inh=1, N=4
x_inh=2, N=4
x_inh=3, N=4
$dv/d\phi_{Act}$
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
ϕ_{Act}


x_inh=1, N=4
x_inh=2, N=4
x_inh=3, N=4
v
0.4
0.2
0.0
−0.2
−0.4
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
ϕ_{Act}


x_inh=1, N=4
x_inh=2, N=4
x_inh=3, N=4
$dv/d\phi_{Act}$
35
30
25
20
15
10
5
0
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
ϕ_{Act}

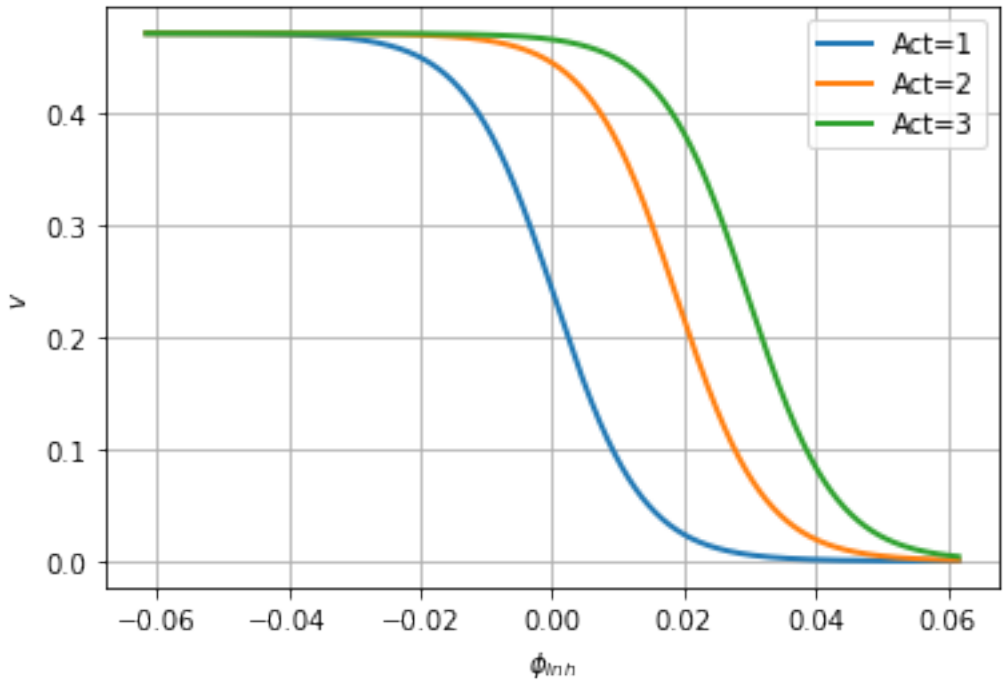
4.5 Vary the inhibition species concentration.

The activation species concentration x_{Inh} is varied for three values of x_{Act}.

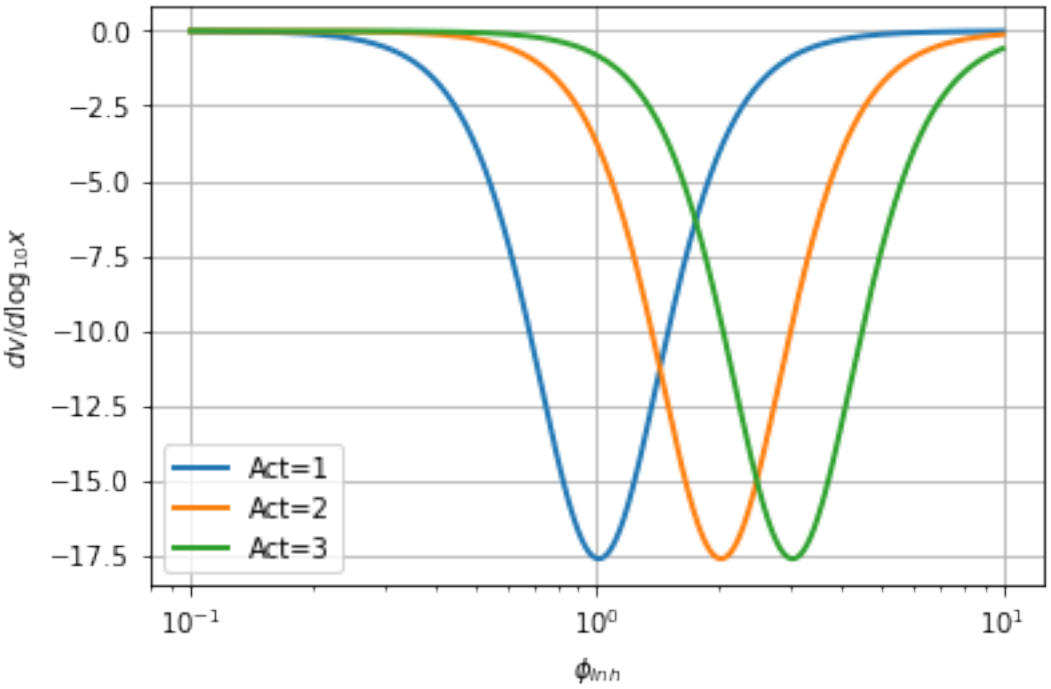
- dotted lines give the cyclic flow.
- The derivative is also plotted.

```
In [17]: Act = [1,2,3]
         for oneway in [True,False]:
             for deriv in [False,True]:
                 v,x,phi = VaryX(sX='Inh',sX1='Act',sX2='A',XX1=Inh,X2=1,oneway=oneway,Xrange=[0
         #dat,x = VaryX(NN=[N],sX='Act',sX1='Inh',sX2='A',XX1=Inh,X2=1,deriv=True)
         #dat,x = VaryX(NN=[N],sX='Act',sX1='Inh',sX2='A',XX1=Inh,X2=1,power=True)
```

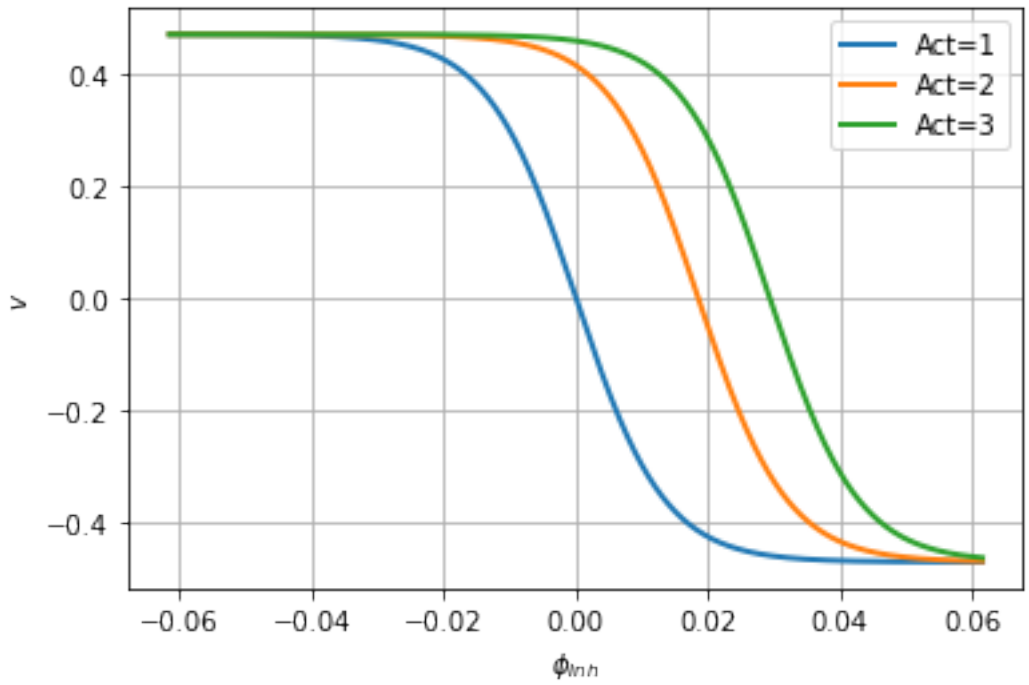
```
100 9900
```



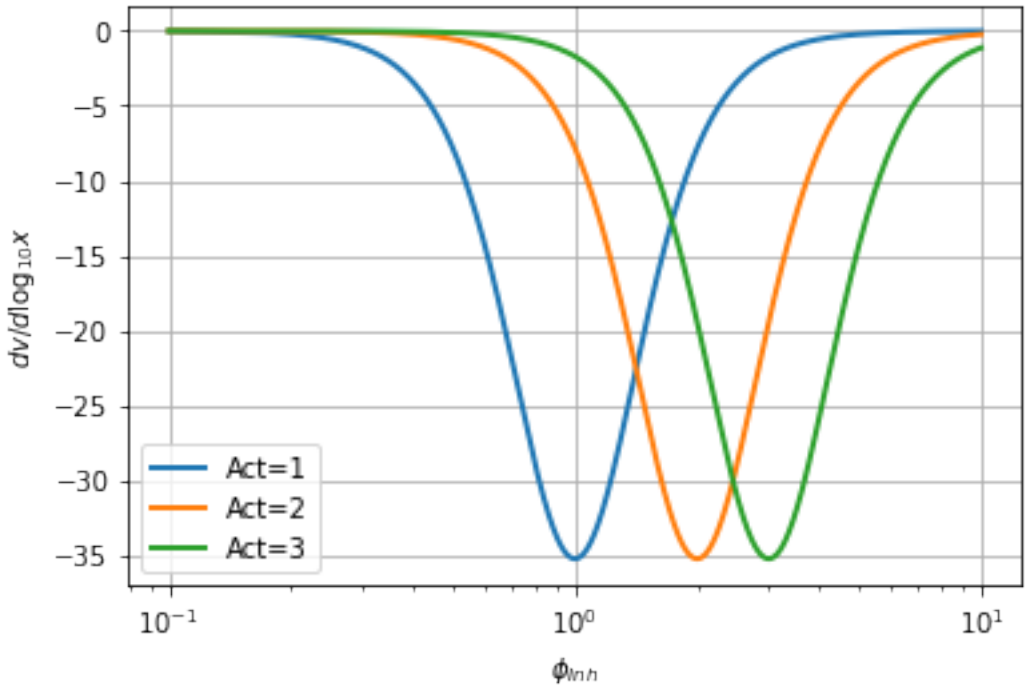

```
100 9900
```


0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−15.0
−17.5
$dv/d\log_{10}x$
Act=1
Act=2
Act=3
10^{-1}
10^{0}
10^{1}
ϕ_{NnH}

100 9900


0.4
0.2
0.0
−0.2
−0.4
v
Act=1
Act=2
Act=3
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
ϕ_{NnH}

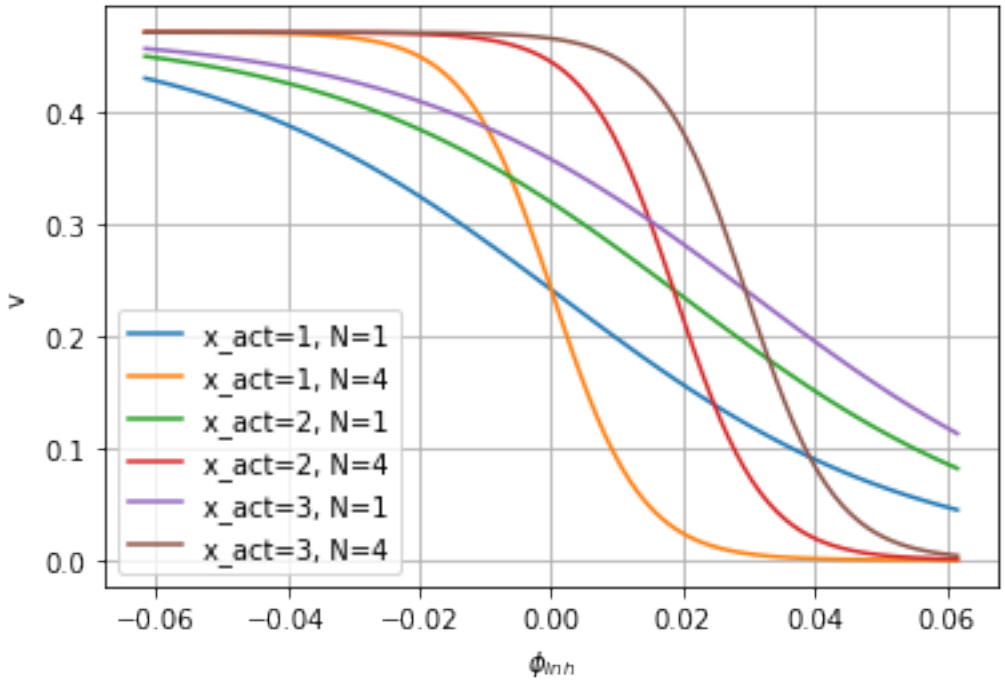
```
100 9900
```

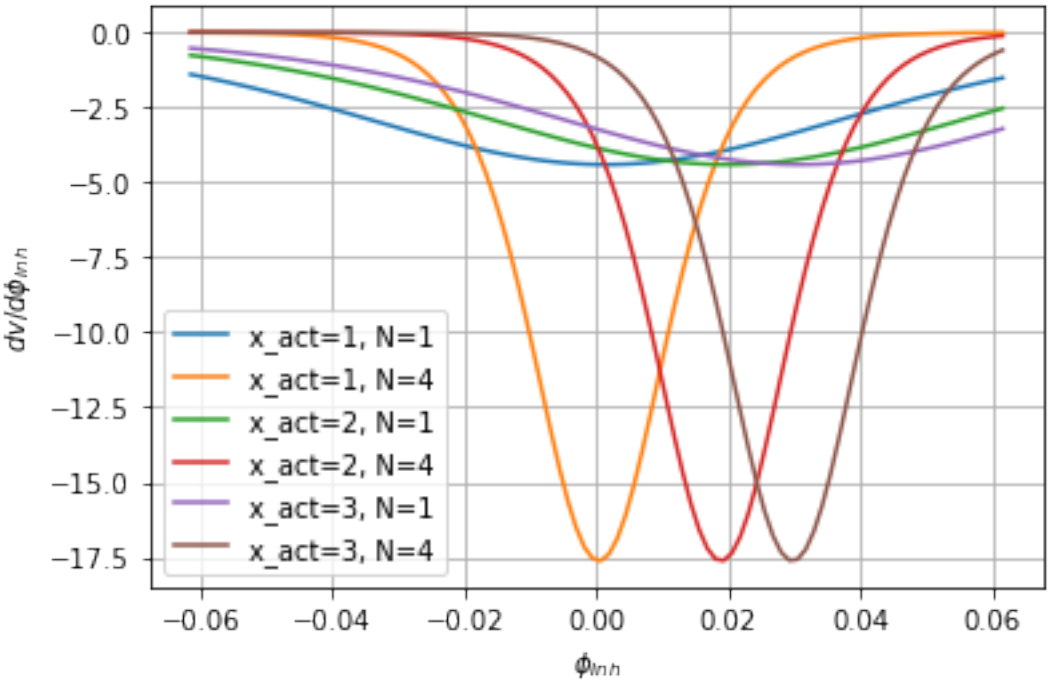


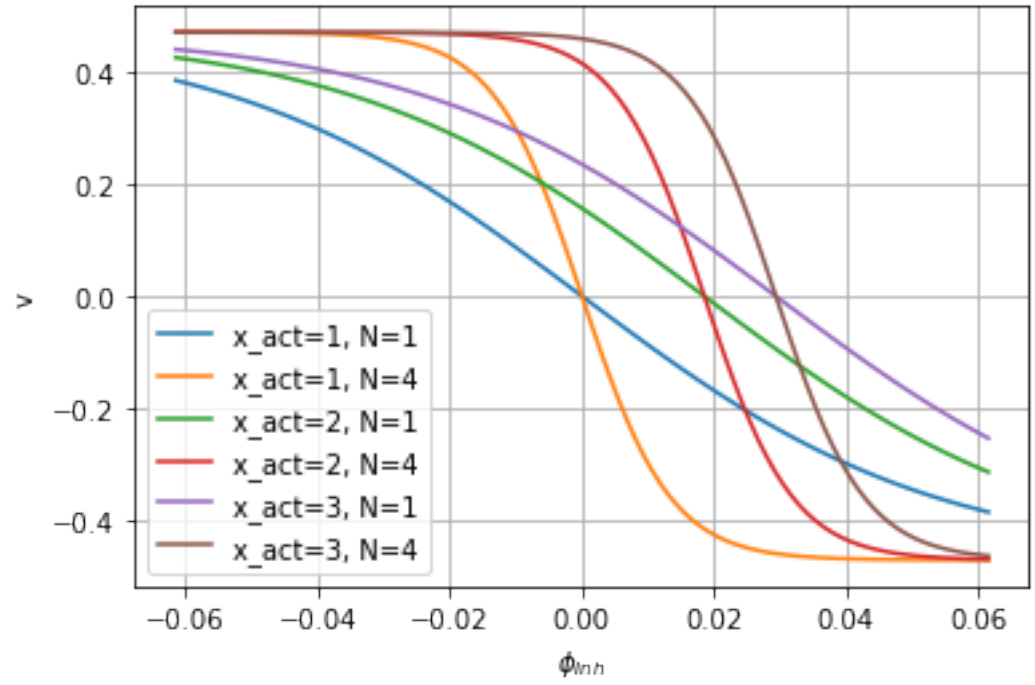
4.5.1 Theory

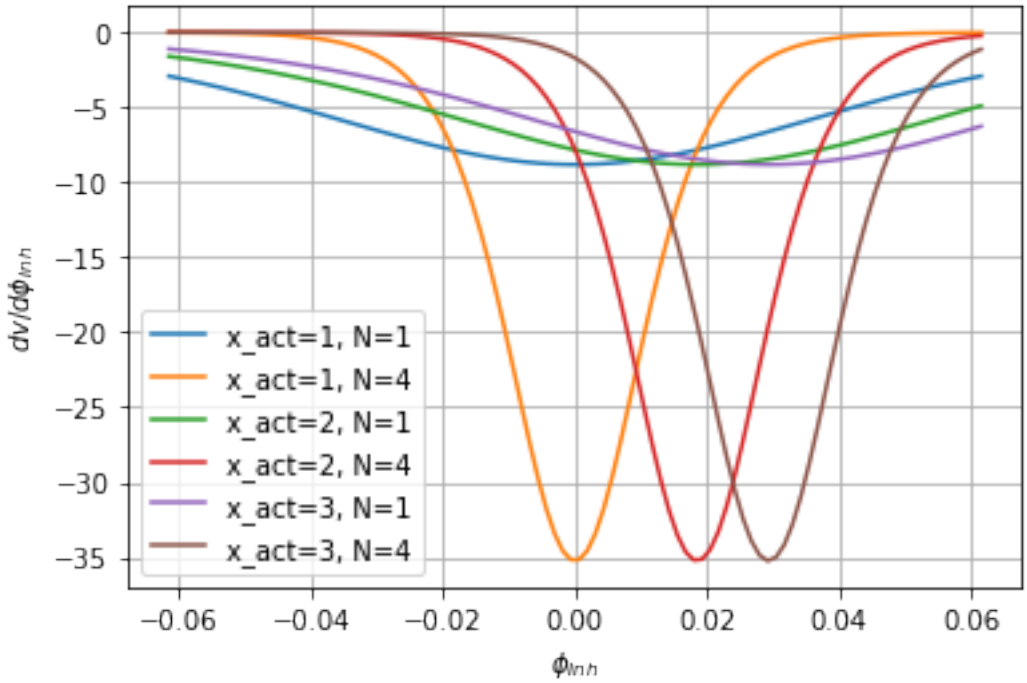
```
In [18]: X_Inh = np.logspace(-1,1,100)
         phi_Inh = st.V_N()*np.log(X_Inh)

         X_A = 1
         X_B = 1

         for oneway in [True,False]:
             for deriv in [False,True]:
                 for act in Act:
                     for N in [1,4]:
         #                if N is 4:
         #                    v,x,phi = VaryX(sX='Inh',sX1='Act',sX2='A',XX1=Inh,X2=1,oneway=on
         #                    if not deriv:
         #                        plt.plot(phi,v,ls='dashed',color = 'black')

                             X_IA = X_Inh/act
                             v_theory,dv_theory = CFM_flow(X_A,X_B,X_IA,e0=e0,N=N,
                                      K_A=K_A,K_B=K_B,K_C=K_C,K_E=1,K_IA=1,
                                      K_F=K_F,K_G=K_G,
```

```
                    kappa_r1 = 1,kappa_r2=1,oneway = oneway)

            if deriv:
                #slope = np.gradient(v_theory,phi_Inh)
                plt.plot(phi_Inh,dv_theory,label=f'x_act={act}, N={N}')
                #plt.plot(phi_Inh,dv_theory*X_IA/st.V_N(),ls='dashed')
            else:
                plt.plot(phi_Inh,v_theory,label=f'x_act={act}, N={N}')

if deriv:
    ylabel = '$dv/d\phi_{Inh}$'
else:
    ylabel = 'v'
plt.ylabel(ylabel)
plt.xlabel('$\phi_{Inh}$')
plt.grid()
plt.legend()
plt.show()
```



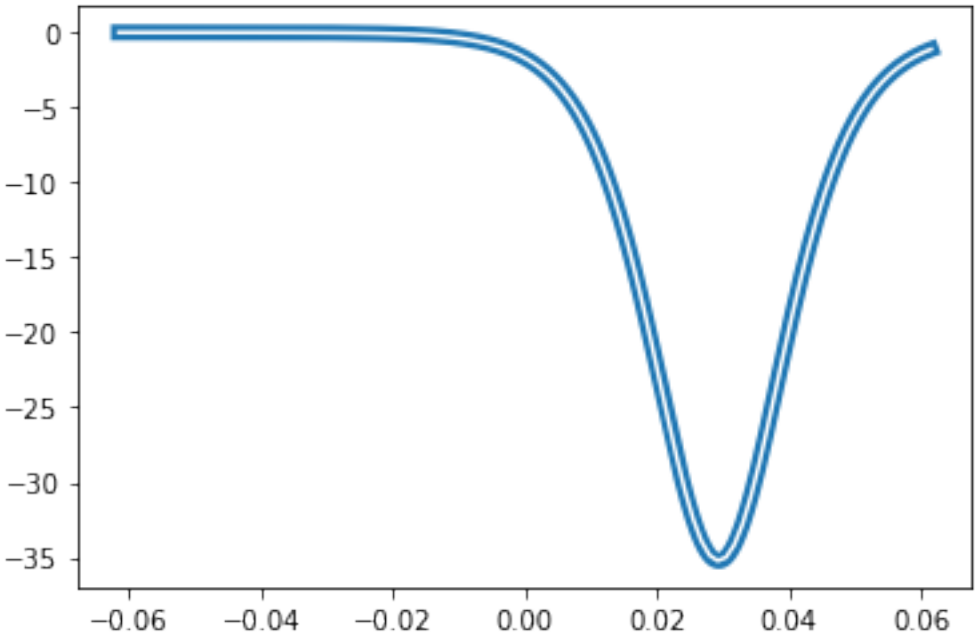

0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−15.0
−17.5
$dv/d\phi_{lnh}$
x_act=1, N=1
x_act=1, N=4
x_act=2, N=1
x_act=2, N=4
x_act=3, N=1
x_act=3, N=4
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
ϕ_{lnh}


0.4
0.2
0.0
−0.2
−0.4
v
x_act=1, N=1
x_act=1, N=4
x_act=2, N=1
x_act=2, N=4
x_act=3, N=1
x_act=3, N=4
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
ϕ_{lnh}



4.5.2 Compare simulation and theory (sanity check)

```
In [19]: ## Compare simulation and theory for last plot
         dv = np.gradient(v,phi)
         plt.plot(phi,dv,linewidth=6)
         plt.plot(phi_Inh,dv_theory,color='white')

Out[19]: [<matplotlib.lines.Line2D at 0x7ff67d1f78d0>]
```



4.6 Discussion

- Both positive and negative flow rates are possible
- The substrate concentration affects net flow but not loop flow.
- Loop flow is affected by the degree of activation/inhibition as well as the driving species AAf, BBf, AAr and BBr.
- Increasing activation increases flow - this corresponds to positive feedback with positive incremental gain given by the derivative plots.
- Increasing inhibition decreases flow - this corresponds to negative feedback with negative incremental gain given by the derivative plots.
- the behaviour is dependent on the parameters of the particular enzyme-catalysed reaction; those used here are for illustration.

5 Fructose-2,6-phosphate ($F_{26}P$)

The reaction F_6P + ATP $\overset{PFK2}{\rightleftharpoons}$ $F_{26}P$ + ADP is catalysed by the enzyme PFK2 where

- PFK_2 phosphofructokinase-2
- F_6P fructose-6-phosphate
- $F_{26}P$ fructose-2,6-biphosphate

As pointed out by (Garrett and Grisham, 2017) section 22.2a, the PFK2-catalysed reaction forms a cycle with the reaction: $F_{26}P + H_2O \overset{F26BP}{\rightleftharpoons} F_6P$ + Pi where:

- $F_{26}BP$ fructose-2,6-biphosphatase

- Pi inorganic phosphate

The species which activate PFK2 and inhibit F26BP include:

- AMP
- F_6P fructose-6-phosphate

Thus this pair of reactions is a further example of Cyclic Flow Modulation (CFM).
Moreover, the PFK and PFK2 CFMs stongly interact:

- The PFK CFM is positively modulated by the product of the PFK2 CFM: $F_{26}P$
- the PFK2 CFM is positively modulated by the substrate of the PKF (and PFK2) CFM: F_6P
- both are positively modulated by AMP.
- this has been sugested as a mechanism for **integral action** (Cloutier and Wellstead, 2010).

TIGAR (TP53-induced gycolysis and apoptosis regulator) mimics $F_{26}P$; this is related to oncogenesis (Garrett and Grisham, 2017)

5.1 Fructose-2,6-phosphate ($F_{26}P$) CFM as an integrator

(Cloutier and Wellstead, 2010) suggest that the reaction catalysed by PFK2 generating $F_{26}P$ can be used as an integrator based on the fact that $F_{26}P$ is a strong activator of PFK. Their model involves a single irreversible reaction $F_6P + ATP \xrightarrow{PFK2} F_{26}P + ADP$; the basic idea is that the the concentration $F_{26}P$ is the integral of the molar flow which is modulated by AMP.

Within the CFM context, a similar effect can be acheived by *not* setting species B to be a chemostat and its state will indeed be the integral of the net CFM flow. However, unlike a true integrator, this flow will depend on the amount of B x_B and thus the CFM in these circumstances will only approximate an integrator. This appproximation will depend on the parameters of the CFM itself.

The approximation will look like a high-gain low-pass filter rather than an integrator.

An alternative approach would have both species A and B not chemostats; they would them form a conserved moiety and the response would be *symmetrical*. Are there any actual biomolecular systems like this?

Both the *asymmetric* and *symmetric* cases are simulated below.

5.2 Simulation

The following simulation illustrates the basic properties of Both the *asymmetric* and *symmetric* cases for a particular set of parameter. The key changes are:

- Asymmetric
  - **Ce:B** is no longer a chemostat
- Symmetric
  - Neither **Ce:A** or **Ce:B** are chemostats
  - The initial state of x_A=999 - the total conserved moiety is thus 1000

The simulation starts from the steady-state and

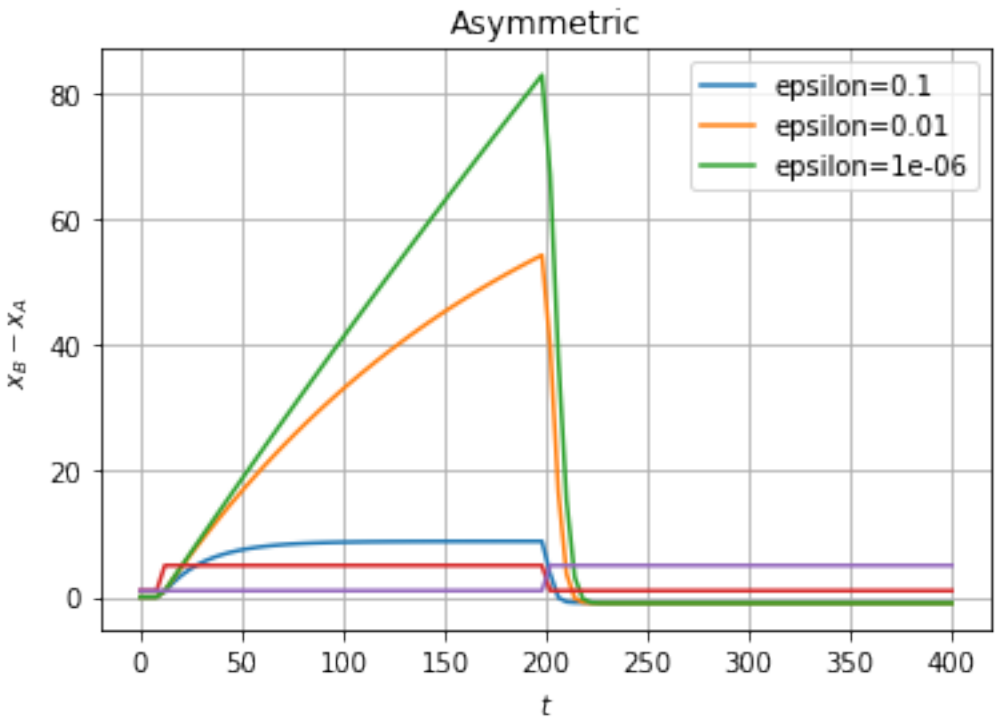
$$x_{Act} = \begin{cases} 5 & \text{for } 10 < t < 200 \\ 1 & \text{otherwise} \end{cases} \tag{31}$$

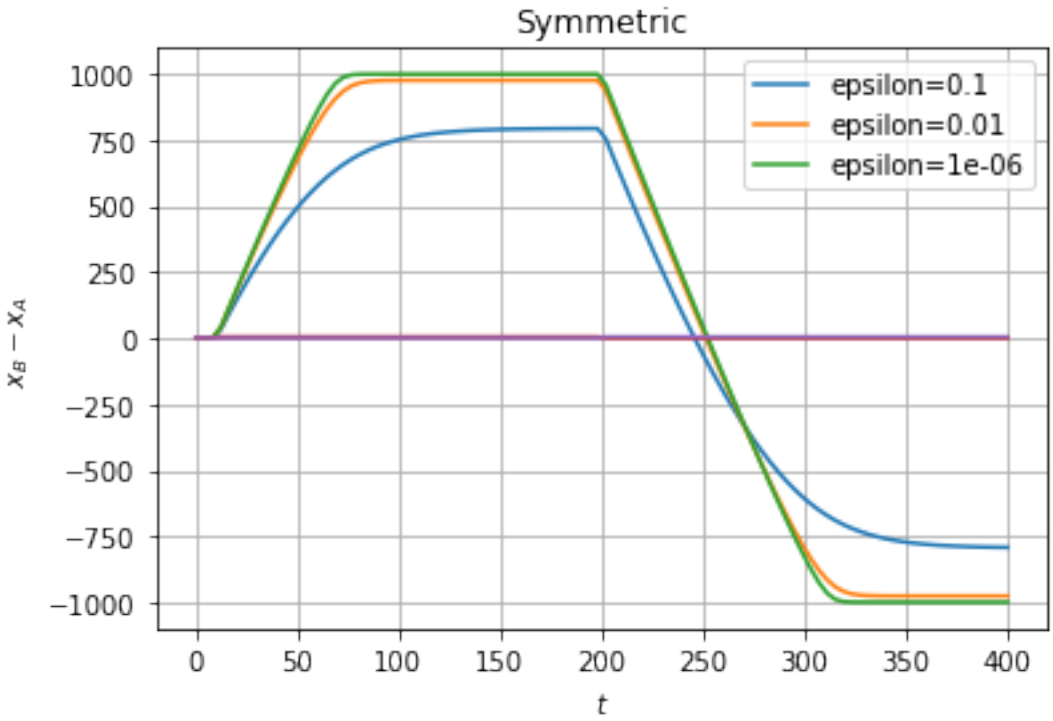
$$x_{Inh} = \begin{cases} 5 & \text{for } 200 < t < 400 \\ 1 & \text{otherwise} \end{cases} \tag{32}$$

```
In [20]: t_ss = np.linspace(0,10000)
         t = np.linspace(0,400,100)

         ## Indices of species
         for i,spec in enumerate(s['species']):
             exec(f'i_{spec} = {i}')

         # Activation and inhition
         amp_Act = 4
         amp_Inh = 4
         t0 = 10
         t1 = 200
         t2 = 200
         t3 = 600
         Act_chemo = f'(1+{amp_Act}*(np.heaviside(t-{t0},1) - np.heaviside(t-{t2},1)))'
         Inh_chemo = f'(1+{amp_Inh}*(np.heaviside(t-{t1},1) - np.heaviside(t-{t3},1)))'
         X_chemo =  {'Act':Act_chemo, 'Inh':Inh_chemo}
         #X_chemo = {'Act':'1+1'}
         print(X_chemo)
         #N=2
         e0 = 1

         for double in [False,True]:

             if double:
                 chemostats=['Act','Inh']
                 x0_A = 999
                 title = 'Symmetric'
             else:
                 chemostats=['A','Act','Inh']
                 x0_A = 1
                 title = 'Asymmetric'

             chemostats += ['Fwd_ecr_F','Fwd_ecr_G','Rev_ecr_F','Rev_ecr_G']
             scB = st.statify(s,chemostats=chemostats)
             #X_chemo=None


             Epsilon = [1e-1,1e-2,1e-6]
             for epsilon in Epsilon:
```

```
            K_G = epsilon
            parameter,X0 = setParameter(oneway=False)
        #     parameter['K_Rev_ecr_F'] = epsilon
        #     parameter['K_Rev_ecr_G'] = epsilon**2

            ## Get the steady-state
            dat = st.sim(s,sc=scB,t=t_ss,parameter=parameter,X0=X0,X_chemo=None,quiet=quiet
            X_ss = dat['X'][-1]


            ## Simulate from steady-state
            dat = st.sim(s,sc=scB,t=t,parameter=parameter,X0=X_ss,X_chemo=X_chemo,quiet=qui
            X = dat['X']
            x_A = X[:,i_A]
            x_B = X[:,i_B]
            x_Act = X[:,i_Act]
            x_Inh = X[:,i_Inh]
            plt.plot(t,x_B-x_A,label=f'epsilon={epsilon}')

        plt.plot(t,x_Act)
        plt.plot(t,x_Inh)
        plt.grid()
        plt.legend()
        plt.title(title)
        plt.xlabel('$t$')
        plt.ylabel('$x_B-x_A$')
        plt.show()
```

```
{'Act': '(1+4*(np.heaviside(t-10,1) - np.heaviside(t-200,1)))', 'Inh': '(1+4*(np.heaviside(t-200
```


Asymmetric
epsilon=0.1
epsilon=0.01
epsilon=1e-06
$x_B - x_A$
t
0
20
40
60
80
0
50
100
150
200
250
300
350
400


Symmetric
epsilon=0.1
epsilon=0.01
epsilon=1e-06
$x_B - x_A$
t
1000
750
500
250
0
−250
−500
−750
−1000
0
50
100
150
200
250
300
350
400

5.3 Discussion: asymmetric case

- In the context of the fructose-2,6-phosphate ($F_{26}P$) CFM, the activator Act is AMP and the product B is $F_{26}P$
- The step change in AMP activation at time t=10 gives rise to an increasing value of $F_{26}P$: this is similar to an integrator response.
- When the activation ceases, the amount of $F_{26}P$ decays.
- As $F_{26}P$ is an activator of PFK, the behaviour would give rise to a similar increase and then decrease of the flow through the PFK reaction.
- Thus PFK + PFK-2 act as a proportional + integral (PI) controller in the context of regulating energy levels (as measured by AMP) via metabolism.

5.4 Discussion: symmetric case

- This setup is speculative at the moment
- B would be used to activate, and A to inhibit, another CFM cycle.
- note that F_6P is the common precurser for both the $F_{16}P$ and $F_{26}P$ reactions.

References

Athel Cornish-Bowden. *Fundamentals of enzyme kinetics*. Wiley-Blackwell, London, 4th edition, 2013. ISBN 978-3-527-33074-4.

Reginald H. Garrett and Charles M. Grisham. *Biochemistry*. Cengage Learning, Boston, MA, 6th edition, 2017.

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 Cudmore, Peter J. Gawthrop, Michael Pan, and Edmund J. Crampin. Computer-aided modelling of complex physical systems with BondGraphTools. Available at arXiv:1906.10799, Jun 2019.

Peter J Gawthrop. Energy-based Feedback Control of Biomolecular Systems with Cyclic Flow Modulation. Available at arXiv:2007.14762, July 2020.

Mathieu Cloutier and Peter Wellstead. The control systems structures of energy metabolism. *Journal of The Royal Society Interface*, 7(45):651–665, 2010. doi:10.1098/rsif.2009.0371.