Cooperative Enzyme-catalysed Reactions

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

March 23, 2020

Contents

Note: this is the Cooperative.ipynb notebook. The PDF version "Cooperative Enzyme-catalysed Reactions" is available here.

1 Introduction

"For many enzymes, the reaction velocity is not a simple hyperbolic curve, as predicted by the Michaelis–Menten model, but often has a sigmoidal character. This can result from cooperative effects, in which the enzyme can bind more than one substrate molecule but the binding of one substrate molecule affects the binding of subsequent ones" (Keener and Sneyd, 2009), Section 1.4.4.

This note gives a bond graph (Gawthrop and Crampin, 2014) interpretation of such cooperativity and uses the iterative properties of BondGraphTools (Cudmore et al., 2019) to build high-order cooperative systems. These systems are simulated to give steady-state behavior as the order of cooperativity increases.

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
import IPython.display as disp

## Stoichiometric analysis
import stoich as st

## SVG bg representation conversion
import svgBondGraph as sbg

## Modular bond graphs
import modularBondGraph as mbg

## Data structure copy
import copy

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

2 Enzyme-catalysed reaction

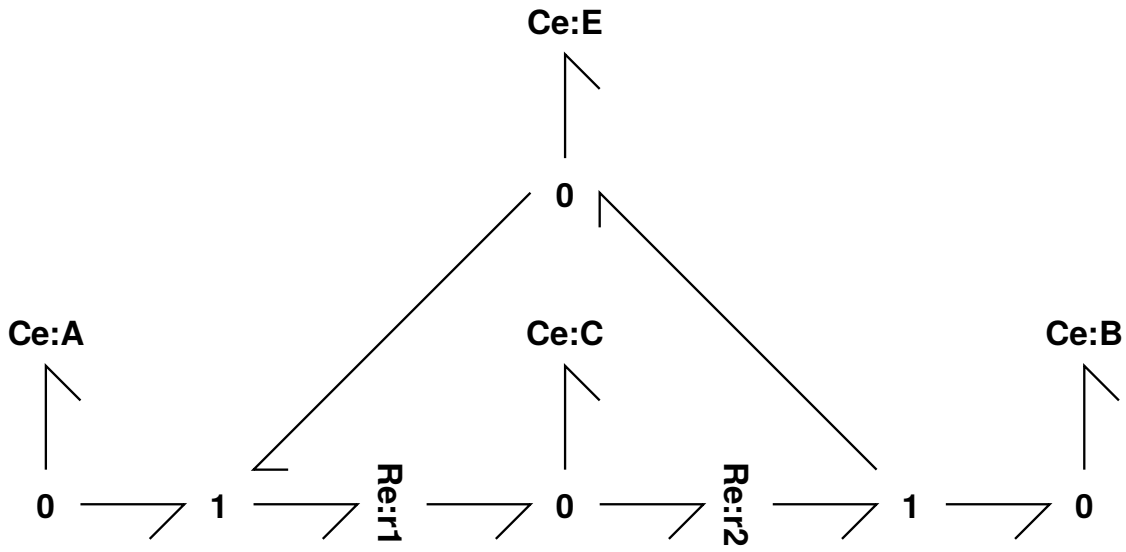
The basic enzyme-catalysed reaction is given in this section. It is the basic building block of cooperative enzyme-catalysed reactions More details are given by (Gawthrop and Crampin, 2014).

In [2]:
```
bgt.version
```

```
Out[2]: '0.3.7'
```

```
In [3]: ## Enzyme-catalysed reaction
        sbg.model('RE_abg.svg')
        import RE_abg
        disp.SVG('RE_abg.svg')
```

Out[3]:




```
In [4]: s = st.stoich(RE_abg.model(),quiet=quiet)
        disp.Latex(st.sprintrl(s,chemformula=True))
```

Out[4]:

$$A + E \underset{}{\overset{r_1}{\rightleftharpoons}} C \tag{1}$$

$$C \underset{}{\overset{r_2}{\rightleftharpoons}} B + E \tag{2}$$

3 Cooperative enzyme-catalysed reaction

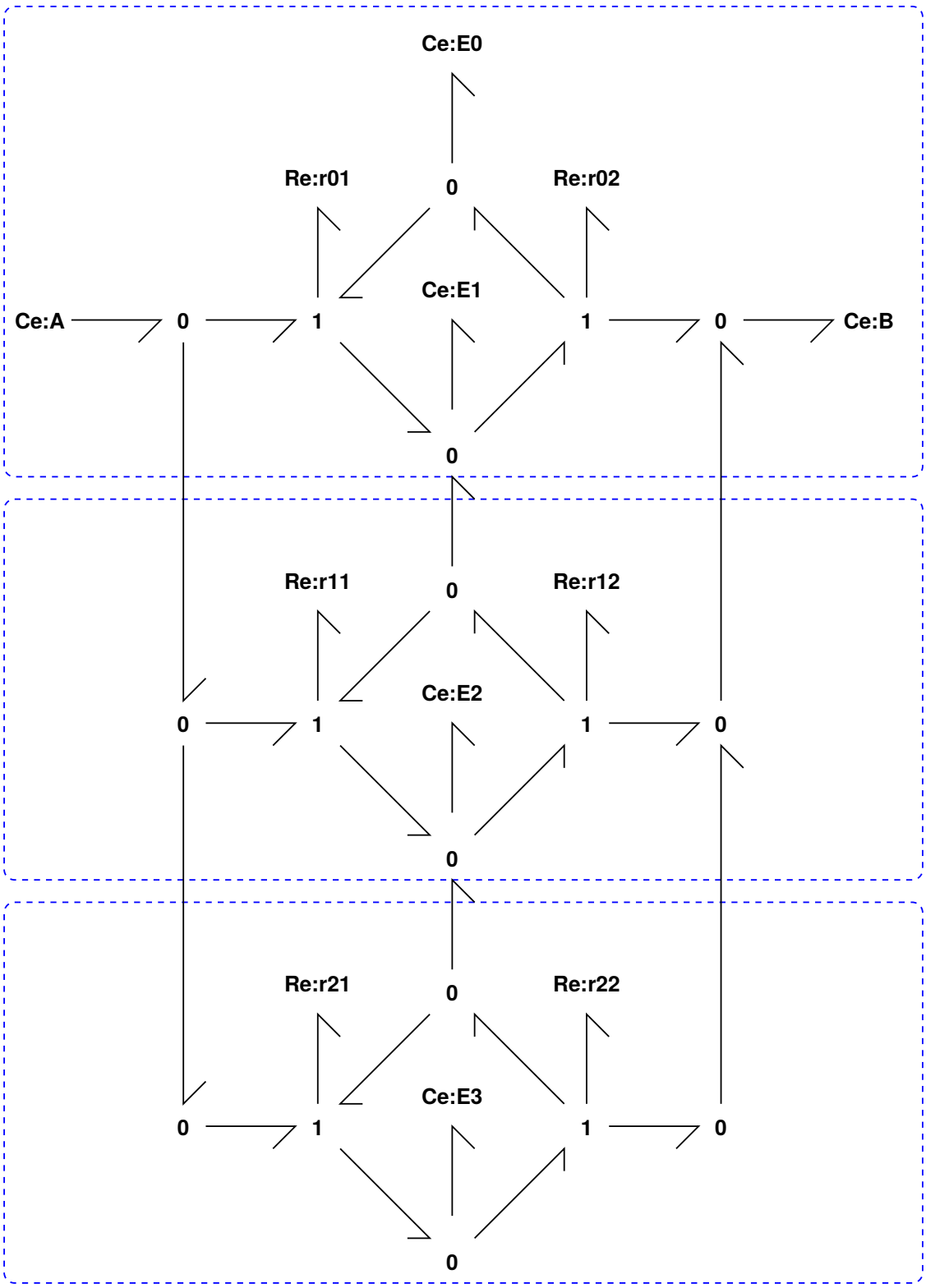
(Keener and Sneyd, 2009), Section 1.4.4, discusses cooperativity. This section gives a bond graph interpretation. This is done in two ways:

1. As a graphical representation of a two-stage cooperative enzyme-catalysed reaction.
2. As a generic representation of an N-stage cooperative enzyme-catalysed reaction using bond-graph tools

3.1 Two-stage cooperative enzyme-catalysed reaction (N=2)

In [5]:
```
## Two-stage cooperative enzyme-catalysed reaction (N=2)
sbg.model('Coop_abg.svg',quiet=quiet)
import Coop_abg
disp.SVG('Coop_abg.svg')
```

Out[5]:




```
In [6]: ss = st.stoich(Coop_abg.model(),quiet=quiet)
```

```
ssc = st.statify(ss,chemostats=['A','B'])
disp.Latex(st.sprintrl(ss,chemformula=True))
```

Out[6]:

$$A + E_0 \rightleftharpoons^{r_{01}} E_1 \tag{3}$$

$$E_1 \rightleftharpoons^{r_{02}} B + E_0 \tag{4}$$

$$A + E_1 \rightleftharpoons^{r_{11}} E_2 \tag{5}$$

$$E_2 \rightleftharpoons^{r_{12}} B + E_1 \tag{6}$$

$$A + E_2 \rightleftharpoons^{r_{21}} E_3 \tag{7}$$

$$E_3 \rightleftharpoons^{r_{22}} B + E_2 \tag{8}$$

3.2 Create cooperative enzyme-catalysed reaction of any degree N

The following code builds an N-stage cooperative enzyme-catalysed reaction using bond-graph tools.

1. N+1 instances of the basic enzyme-catalysed reaction are created and the enzyme and complex renamed.
2. The substrate A, product B and enzymes E1-EN are unified.

```
In [7]: ## Create cooperative enzyme-catalysed reaction of any degree N
        ## Optionally append a simple reaction
        ## Optionally use feedback inhibition
        def makeCoop(N=3,quiet=True):
            Coop = bgt.new(name='Coop')
            for i in range(N+1):
                RE = RE_abg.model()
                RE.name = 'RE'+str(i)
                mbg.rename(RE,{
                            'E':'E'+str(i),
                            'C':'E'+str(i+1),
                            'r1':'r'+str(i)+'1',
                            'r2':'r'+str(i)+'2'
                            },
                           quiet=quiet)
                Coop.add(RE)

            ## Unify common components
            unified = ['A','B']
            for i in range(N):
                Ei = 'E'+str(i+1)
                unified.append(Ei)
            print('unified =',unified)
            mbg.unify(Coop,unified,quiet=quiet)
```

```
    ## Stoichiometry
    chemostats = ['A','B']
    s = st.stoich(Coop,quiet=quiet)
    sc = st.statify(s,chemostats=chemostats)
    if not quiet:
        print(st.sprint(sc,'species'))
        print(st.sprint(sc,'reaction'))
    return s,sc,Coop
```

3.2.1 Generate equations for $N = 2$

Note that these equations are identical to those of the explicit bondgraph.

```
In [8]: ### Generate equations for N=2
        s,sc,Coop = makeCoop(N=2,quiet=quiet)
        disp.Latex(st.sprintrl(s,chemformula=True))

unified = ['A', 'B', 'E1', 'E2']
```

Out[8]:

$$\mathrm{E_0} + \mathrm{A} \underset{}{\overset{\mathrm{r_{01}}}{\rightleftharpoons}} \mathrm{E_1} \tag{9}$$

$$\mathrm{E_1} \overset{\mathrm{r_{02}}}{\rightleftharpoons} \mathrm{E_0} + \mathrm{B} \tag{10}$$

$$\mathrm{A} + \mathrm{E_1} \overset{\mathrm{r_{11}}}{\rightleftharpoons} \mathrm{E_2} \tag{11}$$

$$\mathrm{E_2} \overset{\mathrm{r_{12}}}{\rightleftharpoons} \mathrm{B} + \mathrm{E_1} \tag{12}$$

$$\mathrm{A} + \mathrm{E_2} \overset{\mathrm{r_{21}}}{\rightleftharpoons} \mathrm{E_3} \tag{13}$$

$$\mathrm{E_3} \overset{\mathrm{r_{22}}}{\rightleftharpoons} \mathrm{B} + \mathrm{E_2} \tag{14}$$

3.2.2 Generate pathway equations for $N = 2$

Pathways are generated using the approach of (Gawthrop and Crampin, 2017).

```
In [9]: sp = st.path(s,sc)
        print(st.sprintp(sc))
        disp.Latex(st.sprintrl(sp))

3 pathways
0:  + r01 + r02
1:  + r11 + r12
2:  + r21 + r22
```

Out[9]:

$$A \Leftrightarrow B \tag{15}$$
$$A \Leftrightarrow B \tag{16}$$
$$A \Leftrightarrow B \tag{17}$$

3.3 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.

All parameters are unity except for $K_B = 0.01$ (to approximate an irreversible reaction) and initial states are chosen so that the total enzyme is $e_0 = 1$.

3.3.1 Set up some parameters for simulation

In [10]:
```
## Set up some parameters for simulation
def setParameter(s,N,e0):
    ## Set up the non-unit parameters and states

    K_E0 = 1
    K_EN = 1/K_E0
    K_m = K_EN/K_E0
    parameter = {}

    ## Set product constant to a small value
    ## to make the ECR approximately irreversible
    K_B = 0.01
    parameter['K_B'] = K_B

    ## Set up enzyme parameters and reaction constants
    parameter['K_E0'] = K_E0
    parameter['K_E'+str(N+1)] = K_EN

    ## States
    ## Set total enzyme to e0
    X0 = np.ones(s['n_X'])
    for i in range(N+2):
        Ei = 'E'+str(i)
        X0[s['spec_index'][Ei]] = (e0/(N+2))

    return parameter,X0,K_EN,K_m
```

In [11]:
```
## Compute the total enzyme
def totalE(N,dat,Feedback=False):
    E_spec = []
```

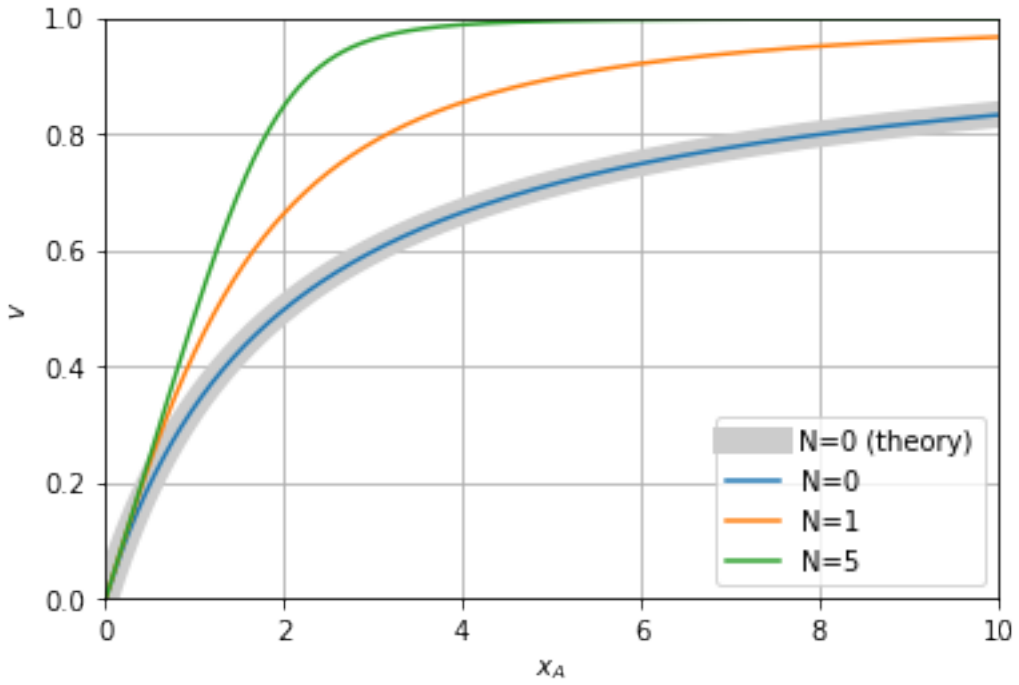
```
        for i in range(N+2):
            Ei = 'E'+str(i)
            E_spec.append(Ei)

        for spec in E_spec:
            x = copy.copy(dat['X'][:,s['spec_index'][spec]])
            if spec is E_spec[0]:
                X = x
            else:
                X += x

        return X
```

```
In [12]: ## Compute the total flow
         def totalFlow(s,N,dat):
             for i in range(N+1):
                 r = 'r'+str(i)+'2'
                 Vi = copy.copy(dat['V'][:,s['reac_index'][r]])
                 if i is 0:
                     V = Vi
                 else:
                     V += Vi
             return V
```

3.3.2 Vary substrate concentration x_A

```
In [13]: ## Simulation
         ## Vary x_A

         ##Time
         quiet = True
         t_max = int(1e4)
         t = np.linspace(0,t_max,10000)
         t_0 = 100
         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

         NN = [0,1,5]
         for N in NN:
             ## Create system stoichiometry
             s,sc,Coop = makeCoop(N=N,quiet=quiet)

             ## Non-unit parameters and states
             e0 = 1 # Total enzyme
             parameter,X0,K_EN,K_m = setParameter(s,N,e0)
```

```
            K_B = parameter['K_B']

            ## Chemostats: vary x_A
            x_max = 100
            x_A_max = 10
            x_min = K_B

            A_chemo = '{3} + ({0}-{3})*np.heaviside(t-{1},1)*((t-{1})/{2})'.format(x_max,t_0,t_
            X_chemo =  {'A':A_chemo}

            ## Simulate
            dat = st.sim(s,sc=sc,t=t,parameter=parameter,X0=X0,X_chemo=X_chemo,quiet=quiet)
            V = totalFlow(s,N,dat)
            x_A = dat['X'][:,s['spec_index']['A']]

            ## Compute approx ECR flow (assumes K_B is small)
            V_ECR = e0*(x_A-K_B)/((2*K_m) + x_A)

            if N is NN[0]:
                VV = V
                VV_ECR = V_ECR
            else:
                VV = np.vstack((VV,V))


        ## Plot flow v. x_A
        grey = '0.8'
        plt.clf()
        plt.plot(x_A,VV_ECR,color=grey,lw=10)
        if len(NN) is 1:
            plt.plot(x_A[-i_1:],VV[-i_1:])
        else:
            plt.plot(x_A[-i_1:],VV[:,-i_1:].T)
        plt.grid()
        plt.ylim((0,e0))
        plt.xlim((0,x_A_max))
        plt.legend(['N=0 (theory)']+['N='+str(i) for i in NN])
        plt.xlabel('$x_A$')
        plt.ylabel('$v$')
        plt.show()

unified = ['A', 'B']
unified = ['A', 'B', 'E1']
unified = ['A', 'B', 'E1', 'E2', 'E3', 'E4', 'E5']
```

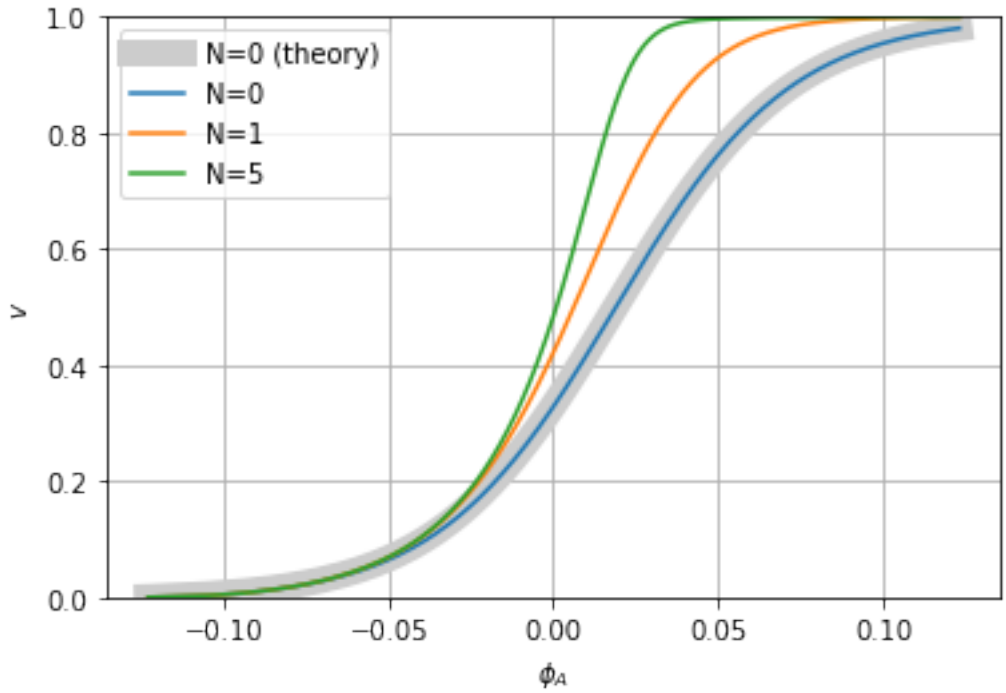


The effect of cooperativity is to give an approximation to a linear rise in v as x_A increases with a cut off at $v = e0 = 1$.

3.4 Replot against ϕ_A

It is common to plot v against the logarithm of x_A. Here we use the potential $\phi_A = \ln K_A x_A$.

```
In [14]: phi_A = dat['phi'][:,s['spec_index']['A']]
         plt.clf()
         plt.plot(phi_A,VV_ECR,color=grey,lw=10)
         if len(NN) is 1:
             plt.plot(phi_A[-i_1:],VV[-i_1:])
         else:
             plt.plot(phi_A[-i_1:],VV[:,-i_1:].T)
         plt.grid()
         plt.ylim((0,e0))
         #plt.xlim((0,x_max))
         plt.legend(['N=0 (theory)']+['N='+str(i) for i in NN])
         plt.xlabel('$\phi_A$')
         plt.ylabel('$v$')
         plt.show()
```

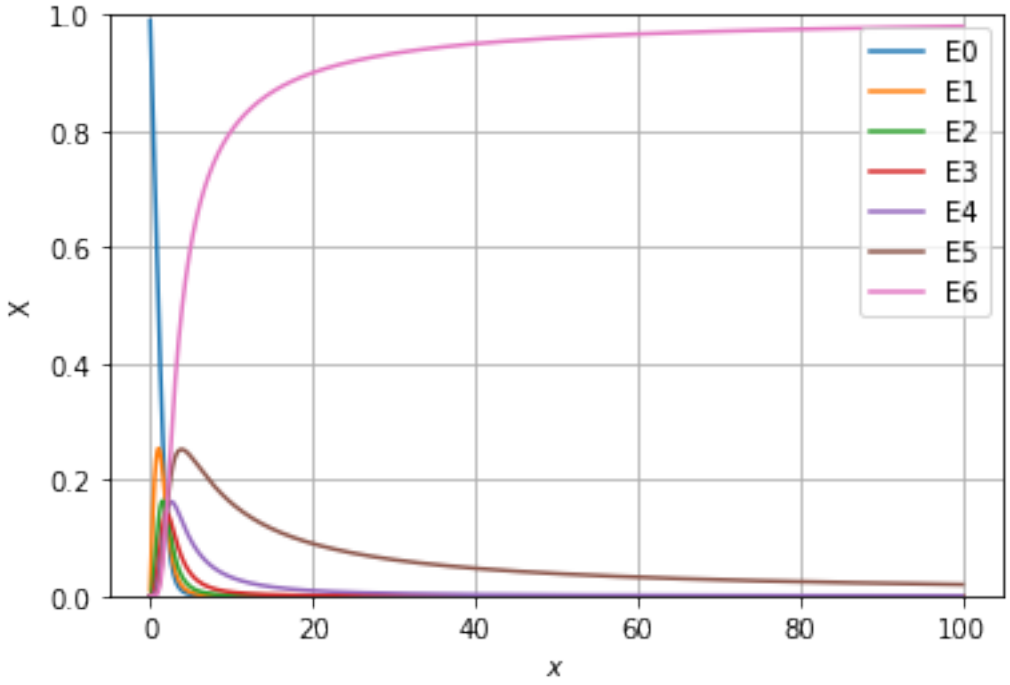


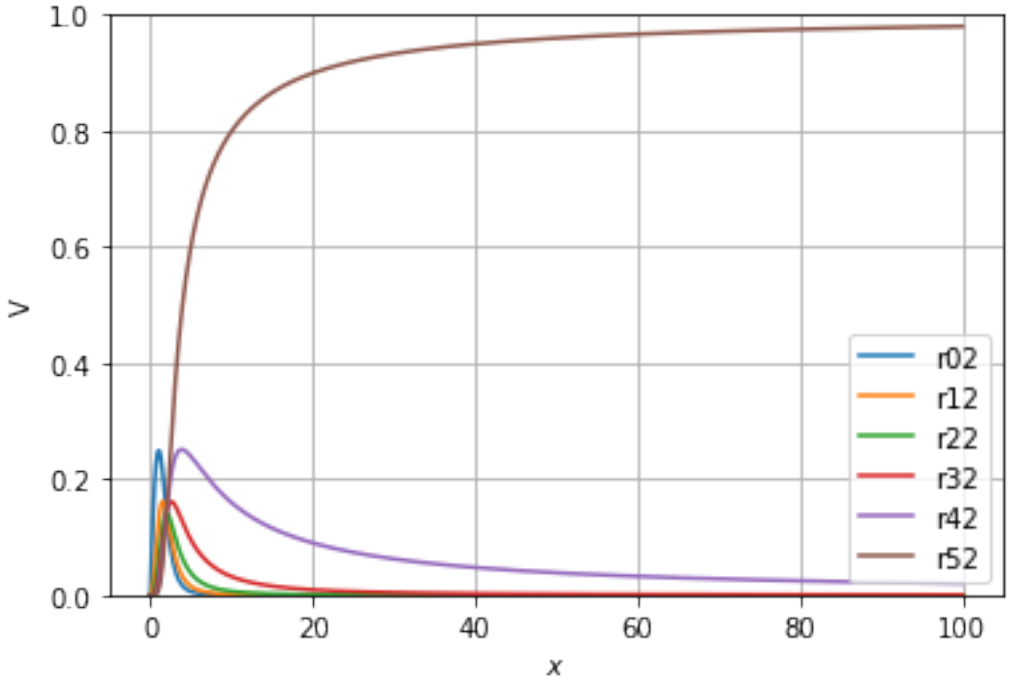
3.5 A closer look

The following graphs look more closely at the behaviour of the system for N=5.

1. The first graph shows the $N+1=6$ enzyme/complex states plotted against x_A. These states form a conserved moiety and the sum is e_0.
2. The second graph shows the $N=5$ reaction flows plotted against x_A. In this particular case, $K_B \approx 0$ and so the flow though reaction $ri2$ is $v_i \approx \kappa_{ri2} K_{Ei+1} x_{Ei+1} = x_{Ei+1}$.

```
In [15]: st.plot(s,dat,species=['E'+str(i) for i in range(N+2)],reaction=[],ylim=(0,1),x=x_A,i0=
         st.plot(s,dat,species=[],reaction=['r'+str(i)+'2' for i in range(N+1)],ylim=(0,1),x=x_A
```


1.0
0.8
0.6
0.4
0.2
0.0
X
0
20
40
60
80
100
x
E0
E1
E2
E3
E4
E5
E6


1.0
0.8
0.6
0.4
0.2
0.0
V
0
20
40
60
80
100
x
r02
r12
r22
r32
r42
r52

References

James P Keener and James Sneyd. *Mathematical Physiology: I: Cellular Physiology*, volume 1. Springer, New York, 2nd edition, 2009.

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 and Edmund J. Crampin. Energy-based analysis of biomolecular pathways. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 473 (2202), 2017. ISSN 1364-5021. doi:10.1098/rspa.2016.0825. Available at arXiv:1611.02332.