Linearisation of Biomolecular Systems

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

May 21, 2024

Contents

Note: this is the Linearisation.ipynb notebook. The PDF version "Linearisation of Biomolecular Systems" is available here.

1 Introduction

As discussed by Gawthrop and Crampin (2016):

"The bond graph approach gives the set of *nonlinear* ordinary differential equations describing the biomolecular system being modelled. Linearisation of non-linear systems is a standard technique in control engineering: as discussed by Goodwin et al. (2001),"The incentive to try to approximate a nonlinear system by a linear model is that the science and art of linear control is vastly more complete and simpler than they are for the nonlinear case.". Nevertheless, it is important to realise that conclusions drawn from linearisation can only be verified using the full *nonlinear* equations."

This notebook tutorial examines the linearisation of bond graph models in the context of biomolecular systems using two simple examples:

1. An enzyme-catalysed reaction
2. An enzyme-catalysed reaction with product removal

As will be seen, the first example is actually linear given the particular choce of chemostats; the second is non-linear the deviation from linearity is examined.

Linearisation of a dynamic system $\dot{x} = f(x, v)$ is with reference to a steady state defined by constant states $x = x_{ss}$ and constant flows $v = v_{ss}$ such that $f(x_{ss}, v_{ss}) = 0$. In general, determination of steady-states is a difficult problem and can only be determined numerically. Some theoretical results are given by Gawthrop (2018). In this tutorial, steady-states can be determined theoretically.

1.1 Import some python code

The bond graph analysis uses a number of Python modules:

[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

## Control systems package
import control as con

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

## Set slycot=True if slycot is installed (see control module)
```

```
slycot=True
```

2 Example system: enzyme-catalysed reaction

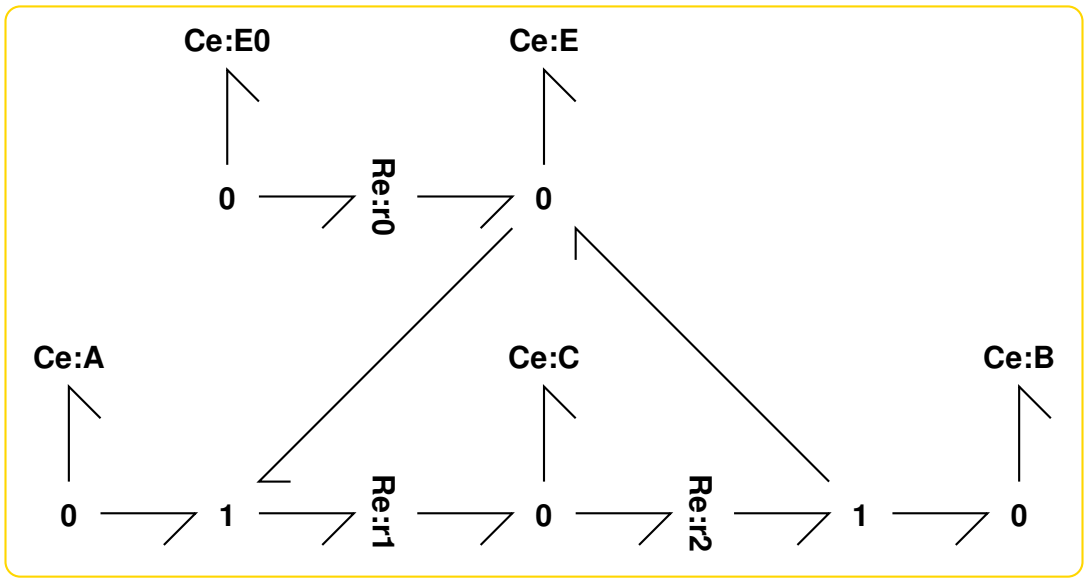
The bond graph representation of the (reversible) enzyme-catalysed reaction is given by Gawthrop and Crampin (2014) and is discussed in the tutorial ECR.

The additional species $E0$ represents a reservoir of enzyme coupled to the ECR via the reaction $r0$. $E0$ is used as a chemostat to adjust the total amount of enzyme associated with the ECR.

[2]:
```
sbg.model('eRE_abg.svg')
import eRE_abg
disp.SVG('eRE_abg.svg')
```

```
{}
```

[2]:



2.1 Stoichiometry & reactions

[3]:
```
s = st.stoich(eRE_abg.model(),quiet=quiet)
disp.Latex(st.sprintrl(s))
```

[3]:

$$E0 \Leftrightarrow E \tag{1}$$
$$A + E \Leftrightarrow C \tag{2}$$
$$C \Leftrightarrow B + E \tag{3}$$

2.2 Chemostats and pathways

[4]:
```
chemostats=['A','B','E0']
sc = st.statify(s,chemostats=chemostats)
sp =st.path(s,sc)
```

```
disp.Latex(st.sprintrl(sp))
```

[4]:

$$A \Leftrightarrow B \tag{4}$$

2.3 Steady state.

In this particular case, the system steady state can be found theoretically assuming constant amounts of A and B Gawthrop and Crampin (2014). Function ssECR does this:

[5]:
```
def ssECR(x_A,x_B,e0=1,
              K_A = 1,K_B=1,K_C=1,K_E=1, K_E0 = 1e-3,
              kappa_r1 = 1,kappa_r2=1):
    """Theoretical steady state of Enzyme-catalysed Reactions
    """

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

    v = kappa_bar*e0*K_C*delta/(K_m+sigma)
    x_E = e0/(1+(sigma/K_m))
    x_C = e0 - x_E
    x_E0 = (K_E/K_E0)*x_E
    x = np.array([x_A,x_B,x_C,x_E,x_E0])
    #x = [x_A,x_B,x_C,x_E,x_E0]
    return x,v

x_A = 2
x_B = 1
K_E0 = 1e-4
K_C = 1
x_ss,v_ss = ssECR(x_A,x_B,K_C=K_C,K_E0=K_E0)
print('x_ss =',x_ss)
print('v_ss =',v_ss)
```

```
x_ss = [2.e+00 1.e+00 6.e-01 4.e-01 4.e+03]
v_ss = 0.2
```

2.4 Parameters

[6]:
```
## Parameters
x_E0 = x_ss[4]

parameter = {}
parameter['K_E0'] = K_E0
parameter['K_C'] = K_C
```

2.5 Linearisation

The function lin provides the linear transfer function sys in control toolbox format and puts the corresponding state-space matrices a, b, c and d into the data structure sc where $\dot{x} = ax + bu$ and $y = cx + du$ The system is the *reduced* form Gawthrop and Crampin (2016). The output y corresponds to the flows V, the input u to the chemostats X_{chemo} and x to the reduced state. In this case

$$y = V = \begin{pmatrix} V_{r0} \\ V_{r1} \\ V_{r2} \end{pmatrix} \tag{5}$$

$$u = X_{chemo} = \begin{pmatrix} X_A \\ X_B \\ X_{E0} \end{pmatrix} \tag{6}$$

$$x = \begin{pmatrix} X_C \\ X_E \end{pmatrix} \tag{7}$$

It also gives the symbolic matrix dv/dx relating incremental flows to incremental states where the states X are:

$$X = \begin{pmatrix} X_A \\ X_B \\ X_C \\ X_E \\ X_{E0} \end{pmatrix} \tag{8}$$

[7]:
```
## imp.reload(st)
sys = st.lin(s,sc,x_ss=x_ss,parameter=parameter,quiet=quiet)
```

[8]:
```
disp.Latex(st.sprintl(sc,'dvdx'))
```

[8]:

$$dvdx = \begin{pmatrix} 0 & 0 & 0 & -K_E\kappa_{r0} & K_{E0}\kappa_{r0} \\ K_AK_E\kappa_{r1}x_E & 0 & -K_C\kappa_{r1} & K_AK_E\kappa_{r1}x_A & 0 \\ 0 & -K_BK_E\kappa_{r2}x_E & K_C\kappa_{r2} & -K_BK_E\kappa_{r2}x_B & 0 \end{pmatrix} \tag{9}$$

[9]:
```
disp.Latex(st.sprintl(sc,'a'))
```

[9]:

$$a = \begin{pmatrix} -2.0 & 3.0 \\ 2.0 & -4.0 \end{pmatrix} \tag{10}$$

[10]:
```
disp.Latex(st.sprintl(sc,'b'))
```

[10]:

$$b = \begin{pmatrix} 0.4 & 0.4 & 0 \\ -0.4 & -0.4 & 0.0001 \end{pmatrix} \tag{11}$$

[11]:
```
disp.Latex(st.sprintl(sc,'c'))
```

[11]:

$$c = \begin{pmatrix} 0 & -1.0 \\ -1.0 & 2.0 \\ 1.0 & -1.0 \end{pmatrix} \tag{12}$$

[12]:
```
disp.Latex(st.sprintl(sc,'d'))
```

[12]:

$$d = \begin{pmatrix} 0 & 0 & 0.0001 \\ 0.4 & 0 & 0 \\ 0 & -0.4 & 0 \end{pmatrix} \tag{13}$$

2.6 Pole/zero analysis

One advantage of dealing with linear systems is the possibility of using standard control system methods Goodwin et al. (2001). One such method is examining the poles and zeros of the system transfer function relating inputs and outputs of interest. In this case, the transfer function relating input X_{E0} to output V_{r2} is examined.

NB con.zero requires slycot to be installed; set slycot=False if slycot is not installed.

Note that the positive zero ($s = 1$) corresponds to the initial negative response of the flow - this is an non-minimum phase system

[13]:
```
i_E0 = chemostats.index('E0')
reaction = sc['reaction']
i_r = reaction.index('r2')
np.set_printoptions(precision=2)
aa = sc['a']
bb = np.array([sc['b'][:,i_E0]]).T
cc = sc['c'][i_r,:]
dd = sc['d'][i_r,i_E0]
siso = con.ss(aa,bb,cc,dd)
np.set_printoptions(precision=2)
gain = x_E0*con.dcgain(siso)
print('System gain = {0:.2f}'.format(gain))
print('System poles = ',con.pole(siso))
if slycot:
    ## This needs slycot
    print('System zeros = ',np.real(con.zero(siso)))
```

```
System gain = 0.20
System poles =  [-0.35+0.j -5.65+0.j]
System zeros =  [1.]
```

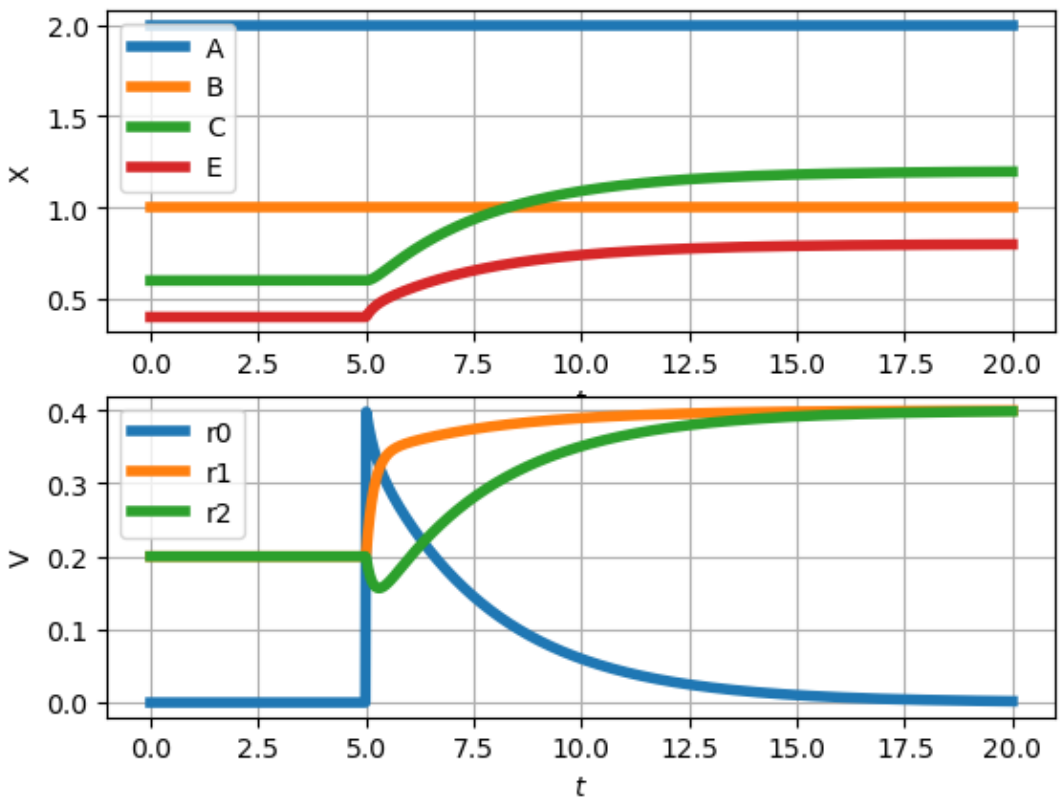
2.7 Simulation

1. As simulation starts at a steady-state value, states and flows remain constant until the chemostat E0 changes at time $t_0 = 5$.

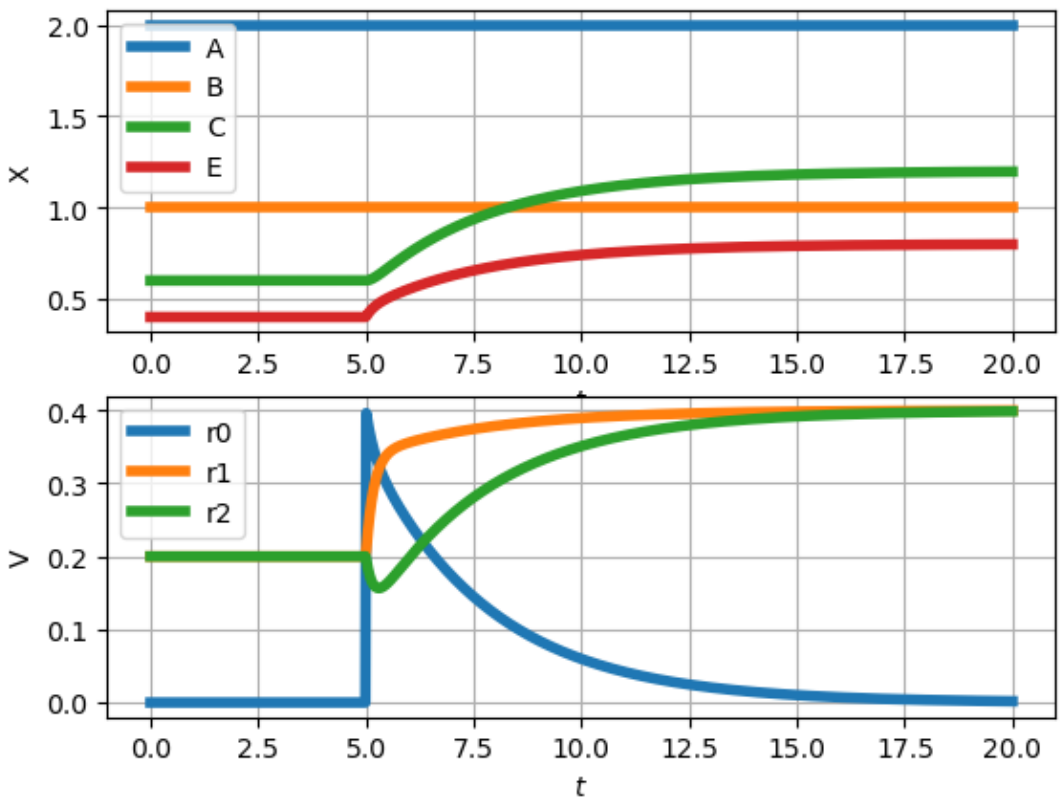
2. For each fixed chemostat value, the non-linear system is actually linear (but with parameters dependent on the steady-state).
3. The flow in reaction r2 is the same for both linear and non-linear simulation.
4. The flow in reaction r2 increases by the gain=0.2.

[14]:
```
## imp.reload(st)
##Time
t_max = int(20)
t = np.linspace(0,t_max,1000)
t_0 = 5

## Chemostat
x_chemo =     '{0}*(1+np.heaviside(t-{1},1))'.format(str(x_E0),str(t_0))
x_chemo_lin = '{0}*(np.heaviside(t-{1},1))'.format(str(x_E0),str(t_0))

## Simulate
print('NON-LINEAR SIMULATION')
#X0 = x_ss
X_chemo = {'E0':x_chemo}
ndat = st.
 ↪sim(s,sc=sc,t=t,parameter=parameter,X0=x_ss,X_chemo=X_chemo,quiet=False)
st.plot(s,ndat,species=['A','B','C','E'])

print('LINEAR SIMULATION')
#X0 = x_ss
X_chemo = {'E0':x_chemo_lin}
V0 = [0,v_ss,v_ss] # Steady-state flows
ldat = st.
 ↪sim(s,sc=sc,t=t,linear=True,V0=V0,parameter=parameter,X0=x_ss,X_chemo=X_chemo,quiet=False)
st.plot(s,ldat,species=['A','B','C','E'])
#st.plot(s,ldat,species=[])
```

```
NON-LINEAR SIMULATION
Setting K_C to 1
Setting K_E0 to 0.0001
```

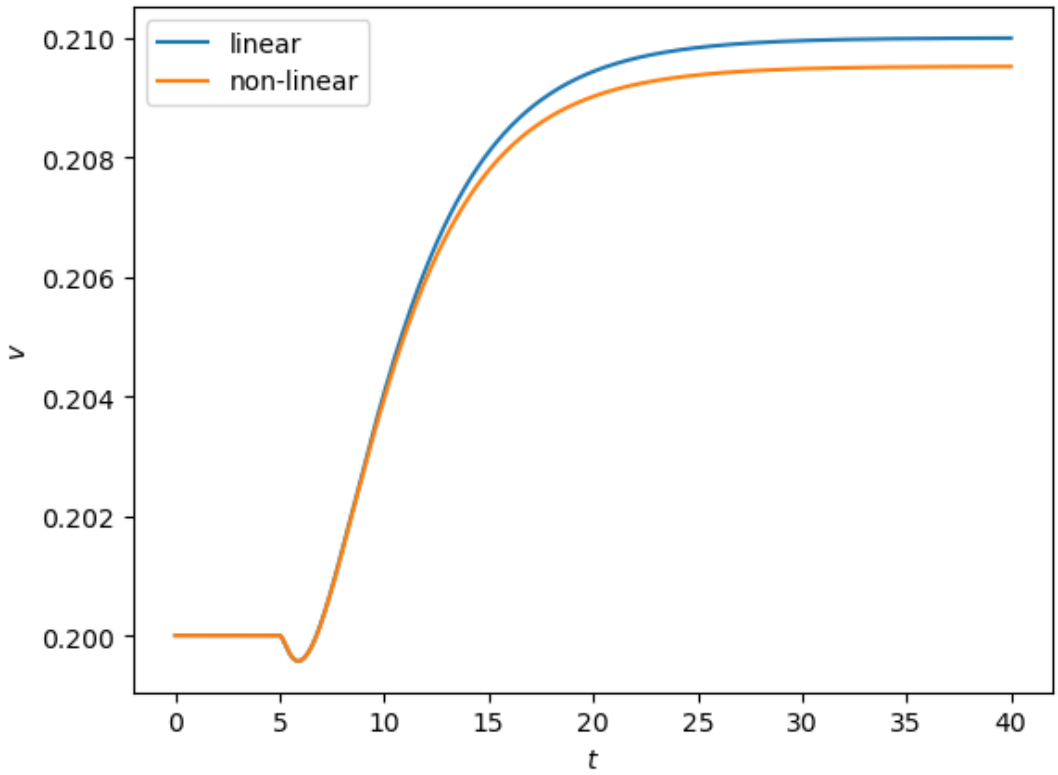



```
LINEAR SIMULATION
Setting K_C to 1
Setting K_E0 to 0.0001
Setting K_C to 1
Setting K_E0 to 0.0001
/home/peterg/WORK/Research/SystemsBiology/lib/python/stoich.py:140:
RuntimeWarning: divide by zero encountered in log
  lx_i = np.log(xx_i)
/home/peterg/WORK/Research/SystemsBiology/lib/python/stoich.py:1932:
RuntimeWarning: invalid value encountered in matmul
  Phi = -phi@N
/home/peterg/WORK/Research/SystemsBiology/lib/python/stoich.py:1964:
RuntimeWarning: invalid value encountered in multiply
  P_Re = Phi*V
/home/peterg/WORK/Research/SystemsBiology/lib/python/stoich.py:1967:
RuntimeWarning: invalid value encountered in multiply
  P_C = phi*dX
```



2.7.1 Compare linear and nonlinear

[28]:
```
def compare(ndat,ldat,v_ss,gain,i):

    v_1_l = ldat['V'][:,i]
    v_1_n = ndat['V'][:,i]

    # ax = plt.gca()  # gca stands for 'get current axis'
    plt.plot(t,v_1_l,label='linear')
    plt.plot(t,v_1_n,label='non-linear')

    plt.xlabel('$t$')
    plt.ylabel('$v$')
    plt.legend()
    plt.grid
    #plt.show()

compare(ndat,ldat,v_ss,gain,i_r)
# compare(ndat,ldat,v_ss,gain,i_r)
```

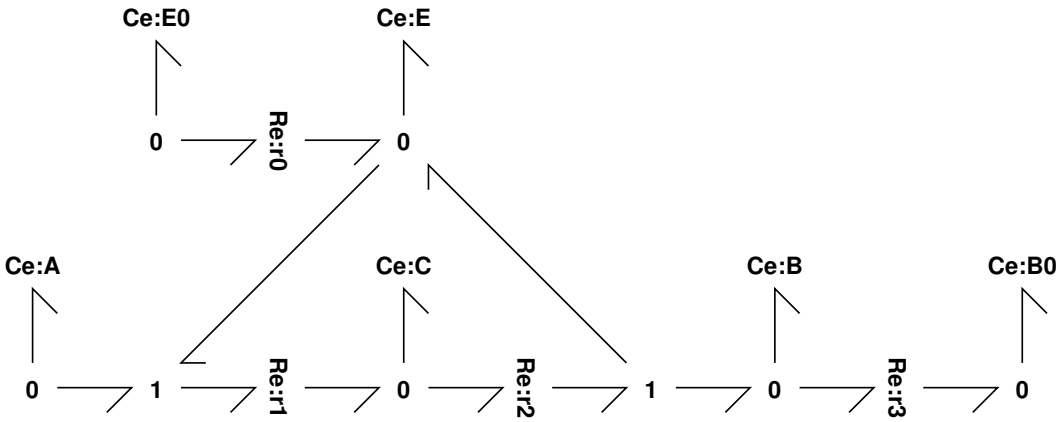


3 Example system: enzyme-catalysed reaction with product removal

[16]:
```
## imp.reload(st)
sbg.model('eRErabg.svg')
import eRErabg
disp.SVG('eRErabg.svg')
```

```
{}
```

[16]:



3.1 Stoichiometry

[17]:
```
s = st.stoich(eREr_abg.model(),quiet=quiet)
print(s['species'])
chemostats=['A','B0','E0']
disp.Latex(st.sprintrl(s))
sc = st.statify(s,chemostats=chemostats)
```

```
['A', 'B', 'B0', 'C', 'E', 'E0']
```

3.2 Parameters

[18]:
```
## Parameters
# x_E0 = x_ss[4]
# x_A = 2
# x_B = 1

parameter = {}
parameter['K_E0'] = K_E0
parameter['K_C'] = K_C
```

3.3 Steady state.

As a special case, the amount of B0 is taken to be very small and the parameter $\kappa_3 = v_{ss}$. This gives the same steady states a the previous section.

[19]:
```
## Steady states for ECR
x_ss,v_ss = ssECR(x_A,x_B,K_C=K_C,K_E0=K_E0)

## Steady states for this examples
X_ss = np.array([x_ss[0],x_ss[1],1e-10,x_ss[2],x_ss[3],x_ss[4]])
#print(X_ss)
V_ss = np.array([0,v_ss,v_ss,v_ss])
#print(V_ss)

## Set appropriate  kappa3
kappa_r3 = v_ss
parameter['kappa_r3'] = kappa_r3
```

3.4 Linearisation

The function lin provides the linear transfer function sys in control toolbox format and puts the corresponding state-space matrices a, b, c and d into the data structure sc where $\dot{x} = ax + bu$ and $y = cx + du$ The system is the *reduced* form Gawthrop and Crampin (2016). The output y corresponds to the flows V, the input u to the chemostats X_{chemo} and x to the reduced state.

In this case

$$y = V = \begin{pmatrix} V_{r0} \\ V_{r1} \\ V_{r2} \\ V_{r3} \end{pmatrix} \tag{14}$$

$$U = X_{chemo} = \begin{pmatrix} X_A \\ X_{B0} \\ X_{E0} \end{pmatrix} \tag{15}$$

$$x = \begin{pmatrix} X_B \\ X_C \\ X_E \end{pmatrix} \tag{16}$$

It also gives the symbolic matrix dv/dx relating incremental flows to incremental states where the states X are:

$$X = \begin{pmatrix} X_A \\ X_B \\ X_{B0} \\ X_C \\ X_E \\ X_{E0} \end{pmatrix} \tag{17}$$

[20]:
```
## imp.reload(st)
sys = st.lin(s,sc,x_ss=X_ss,parameter=parameter)
disp.Latex(st.sprintl(sc,'dvdx'))
```

```
Setting K_C to 1
Setting K_E0 to 0.0001
Setting kappa_r3 to 0.2
```

[20]:

$$dvdx = \begin{pmatrix} 0 & 0 & 0 & 0 & -K_E\kappa_{r0} & K_{E0}\kappa_{r0} \\ K_AK_E\kappa_{r1}x_E & 0 & 0 & -K_C\kappa_{r1} & K_AK_E\kappa_{r1}x_A & 0 \\ 0 & -K_BK_E\kappa_{r2}x_E & 0 & K_C\kappa_{r2} & -K_BK_E\kappa_{r2}x_B & 0 \\ 0 & K_B\kappa_{r3} & -K_{B0}\kappa_{r3} & 0 & 0 & 0 \end{pmatrix} \tag{18}$$

[21]:
```
disp.Latex(st.sprintl(sc,'a'))
```

[21]:

$$a = \begin{pmatrix} -0.6 & 1.0 & -1.0 \\ 0.4 & -2.0 & 3.0 \\ -0.4 & 2.0 & -4.0 \end{pmatrix} \tag{19}$$

[22]:
```
disp.Latex(st.sprintl(sc,'b'))
```

[22]:

$$b = \begin{pmatrix} 0 & 0.2 & 0 \\ 0.4 & 0 & 0 \\ -0.4 & 0 & 0.0001 \end{pmatrix} \tag{20}$$

[23]:
```
disp.Latex(st.sprintl(sc,'c'))
```

[23]:

$$c = \begin{pmatrix} 0 & 0 & -1.0 \\ 0 & -1.0 & 2.0 \\ -0.4 & 1.0 & -1.0 \\ 0.2 & 0 & 0 \end{pmatrix} \tag{21}$$

[24]:
```
disp.Latex(st.sprintl(sc,'d'))
```

[24]:

$$d = \begin{pmatrix} 0 & 0 & 0.0001 \\ 0.4 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & -0.2 & 0 \end{pmatrix} \tag{22}$$

3.5 Pole/zero analysis

One advantage of dealing with linear systems is the possibility of using standard control system methods Goodwin et al. (2001). One such method is examining the poles and zeros of the system transfer function relating inputs and outputs of interest. In this case, the transfer function relating input X_{E0} to output V_{r2} is examined.

NB con.zero requires slycot to be installed; set slycot=False if slycot is not installed.

Note that the positive zero ($s = 1$) corresponds to the initial negative response of the flow - this is an non-minimum phase system.

[25]:
```
i_E0 = chemostats.index('E0')
reaction = sc['reaction']
i_r = reaction.index('r3')
np.set_printoptions(precision=2)
aa = sc['a']
bb = np.array([sc['b'][:,i_E0]]).T
cc = sc['c'][i_r,:]
dd = sc['d'][i_r,i_E0]
siso = con.ss(aa,bb,cc,dd)
np.set_printoptions(precision=2)
gain = x_E0*con.dcgain(siso)
print('System gain = {0:.2f}'.format(gain))
print('System poles = ',con.pole(siso))
if slycot:
    ## This needs slycot
    print('System zeros = ',np.real(con.zero(siso)))
```

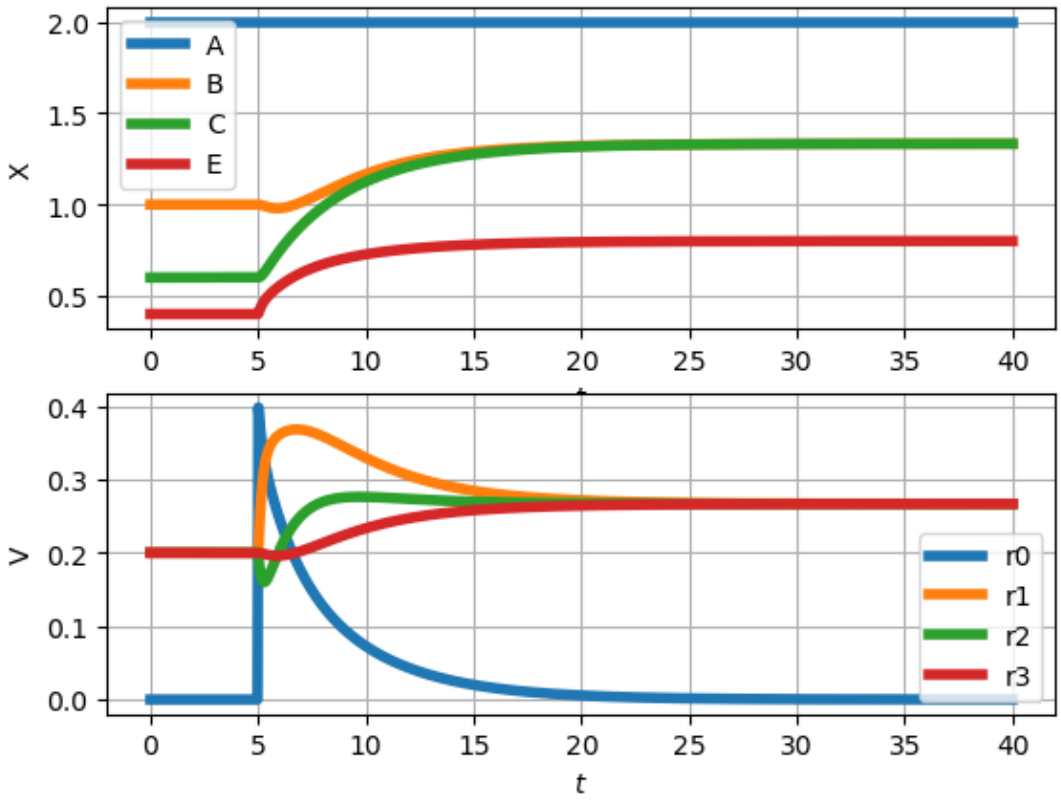
```
System gain = 0.10
System poles =  [-5.8 +0.j -0.56+0.j -0.25+0.j]
System zeros =  [1.]
```

3.6 Simulation

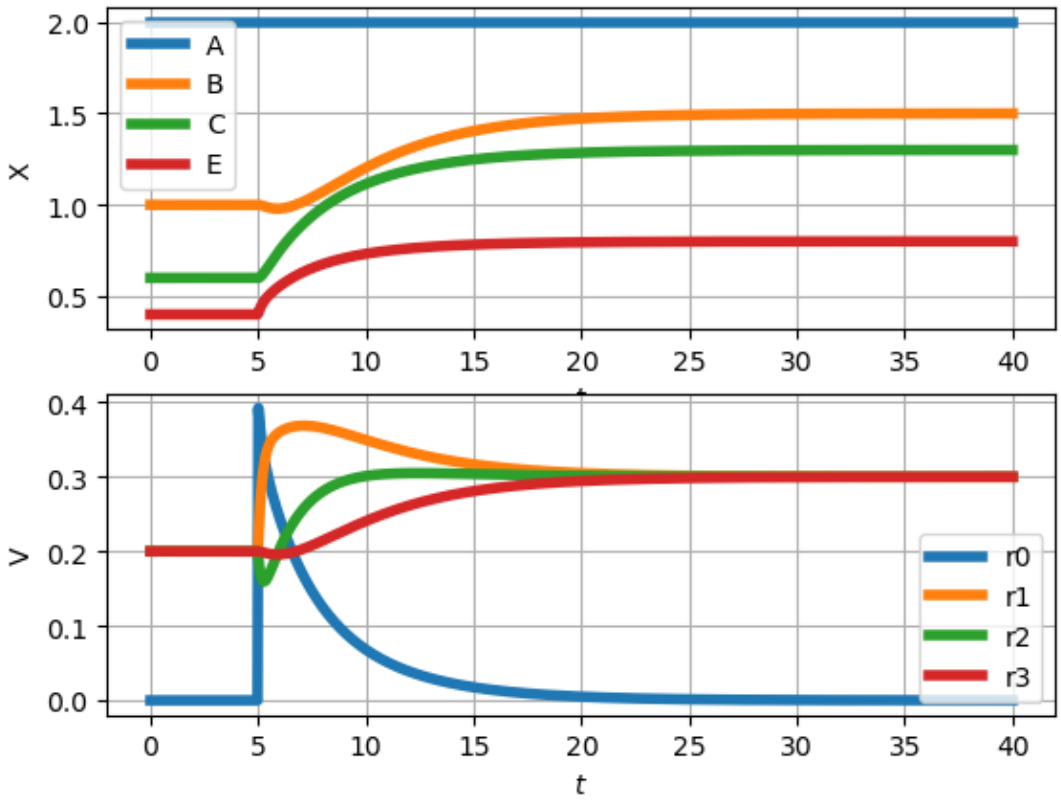
1. As simulation starts at a steady-state value, states and flows remain constant until the chemostat E0 changes at time $t_0 = 5$.
2. For each fixed chemostat value, the non-linear system leads to an approximate linear system (but with parameters dependent on the steady-state).
3. The flow in reaction r2 is the different for linear and non-linear simulation, but the difference is smaller for smaller step changes in x_{E0}.
4. In the linear case, the flow in reaction r2 increases by gain=0.1 multiplied by the step.
5. In both linear and non-linear cases, the flow in reaction r1 is zero in the steady-state. Thus flow from the input source E0 is transient; in other words *retroactivity* is zero in the steady state.

[26]:
```
## imp.reload(st)

##Time
t_max = int(40)
t = np.linspace(0,t_max,1000)
t_0 = 5

for step in [1,0.1]:

    ## Chemostat
    #step = 1
    x_chemo =     '{0}*(1+{2}*np.heaviside(t-{1},1))'.
↪format(str(x_E0),str(t_0),str(step))
    x_chemo_lin = '{0}*({2}*np.heaviside(t-{1},1))'.
↪format(str(x_E0),str(t_0),str(step))

    ## Simulate
    print('NON-LINEAR SIMULATION: Step = '+str(step))
    #X0 = x_ss
    X_chemo = {'E0':x_chemo}
    ndat = st.
↪sim(s,sc=sc,t=t,parameter=parameter,X0=X_ss,X_chemo=X_chemo,quiet=quiet)
    st.plot(s,ndat,species=['A','B','C','E'])

    print('LINEAR SIMULATION: Step = '+str(step))
    #X0 = x_ss
    X_chemo = {'E0':x_chemo_lin}
    V0 = [0,v_ss,v_ss] # Steady-state flows
    ldat = st.
↪sim(s,sc=sc,t=t,linear=True,V0=V_ss,parameter=parameter,X0=X_ss,X_chemo=X_chemo,quiet=quie
    st.plot(s,ldat,species=['A','B','C','E'])

    print('COMPARE LINEAR & NONLINEAR: Step = '+str(step))
    compare(ndat,ldat,v_ss,step*gain,i_r)
```

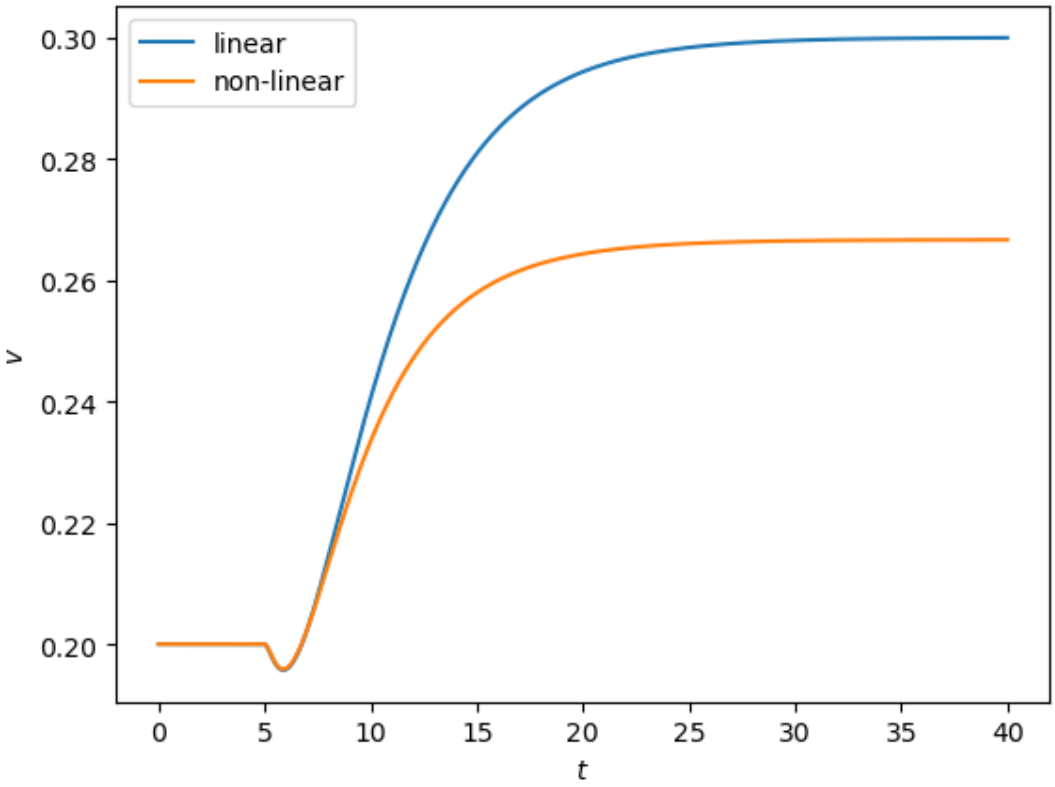
```
NON-LINEAR SIMULATION: Step = 1
```



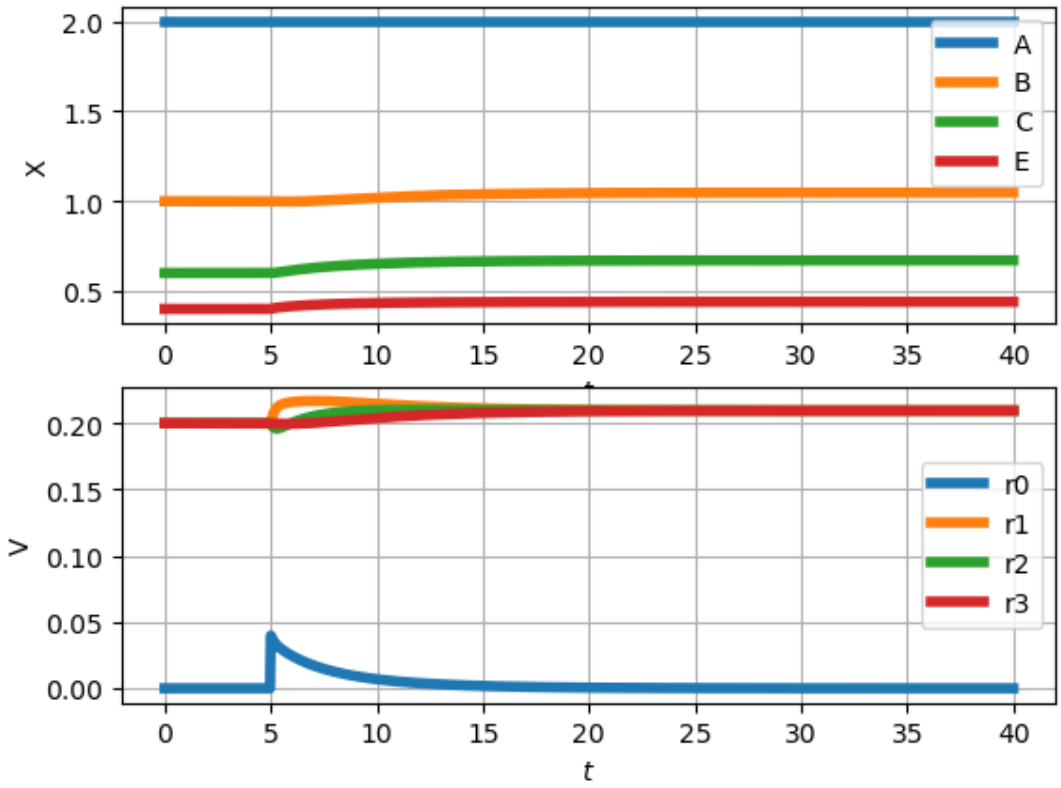
LINEAR SIMULATION: Step = 1



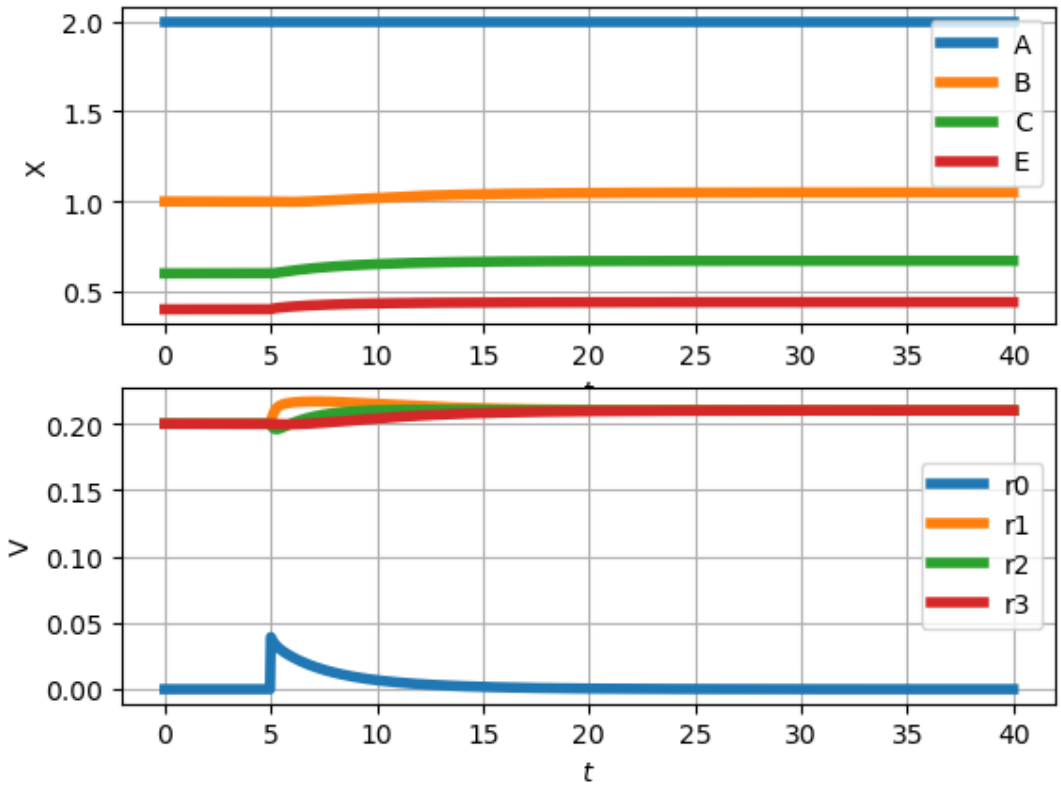
COMPARE LINEAR & NONLINEAR: Step = 1



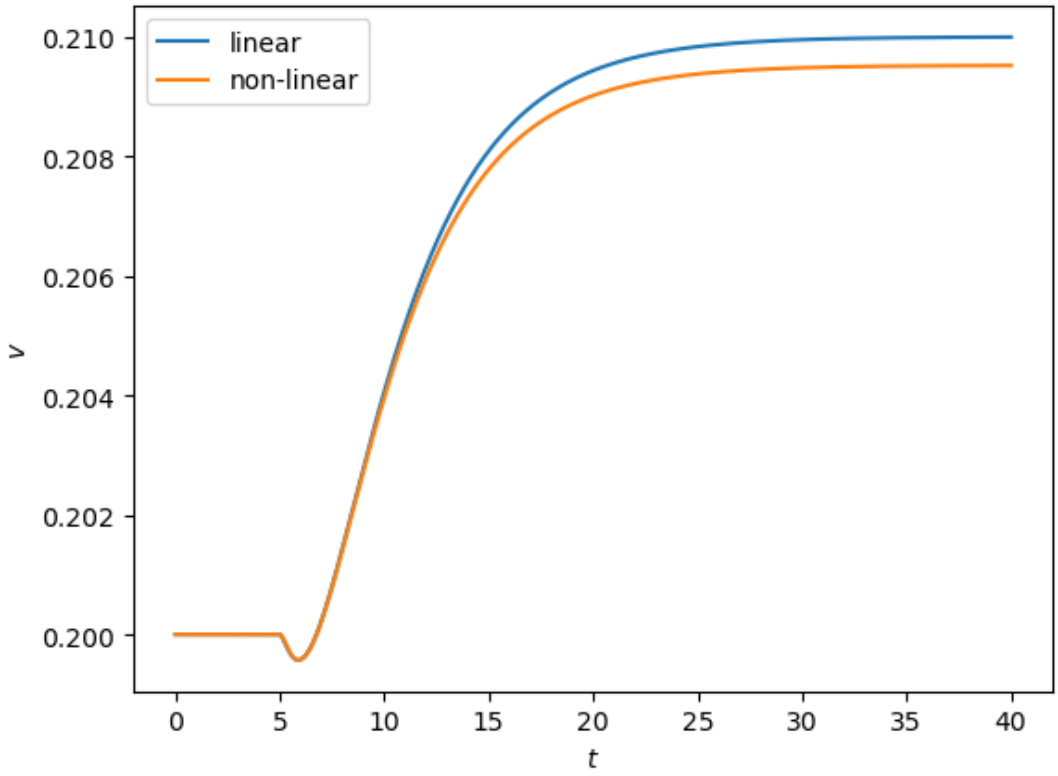
NON-LINEAR SIMULATION: Step = 0.1



LINEAR SIMULATION: Step = 0.1



```
COMPARE LINEAR & NONLINEAR: Step = 0.1
```



References

P. Gawthrop. Computing biomolecular system steady-states. *IEEE Transactions on NanoBioscience*, 17(1):36–43, March 2018. ISSN 1536-1241. doi: 10.1109/TNB.2017.2787486. Published online 25th December 2017.

P. J. Gawthrop and E. J. Crampin. Modular bond-graph modelling and analysis of biomolecular systems. *IET Systems Biology*, 10(5):187–201, October 2016. ISSN 1751-8849. doi: 10.1049/iet-syb.2015.0083. Available at arXiv:1511.06482.

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.

G.C. Goodwin, S.F. Graebe, and M.E. Salgado. *Control System Design*. Prentice Hall, Englewood Cliffs, New Jersey, 2001.