Energy-based Advection Modelling Using Bond Graphs: Additional Material

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

August 25, 2022

Contents

1 Introduction

This notebook generates the figures for the paper: *Energy-based Advection Modelling Using Bond Graphs*. This is a tidied and extended version of the 2021 version.

1.1 Import some python code

The bond graph analysis uses a number of Python modules:

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import IPython.display as disp

## Control toolbox
```

```
import control as con
## Bond graph tools
import BondGraphTools as bgt
print('BondGraphTools version', bgt.version)
## Stoichiometric analysis
import stoich as st
## SVG bg representation conversion
import svgBondGraph as sbg
## Modular bond graphs
import modularBondGraph as mbg
## Stoichiometric conversion
import stoichBondGraph as stbg
## Allow reloading of modules
import importlib as imp

## CSV reader etc
import csv
from os.path import exists

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

## Plot parameters
lw = 5

## Optional plotting
Plotting = True
def Savefig(plotname):
    if Plotting:
        plt.savefig(plotname)
```

```
BondGraphTools version 0.3.7
```

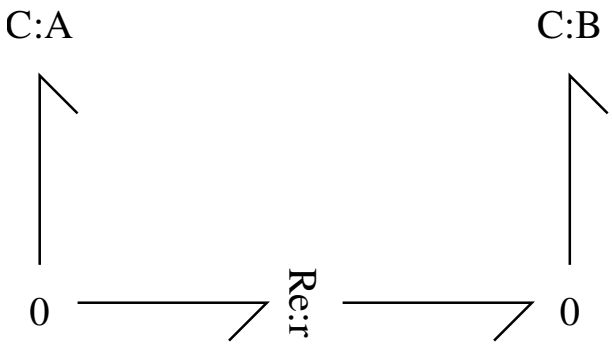
2 Pipe

2.1 Chain of compartments

This is the lumped model of advection considered in Section 3.1. of the paper. Using the modular extension to BGT, a chain of simple A $\rightleftharpoons$ B reactions of length N is created.

[2]:
```
## Simple A <==> B reaction
sbg.model('AB_abg.svg')
import AB_abg
disp.SVG('AB_abg.svg')
```

[2]:



[3]:
```
s0 = st.stoich(AB_abg.model(),quiet=quiet)
disp.Latex(st.sprintrl(s0))
```

[3]:

$$A \Leftrightarrow B \tag{1}$$

[4]:
```
chain = mbg.chain(AB_abg.model(),inport = 'A',outport = 'B',
                      N=1,rename_components=False,quiet=False)
ss0 = st.stoich(chain,quiet=quiet)
disp.Latex(st.sprintrl(ss0))
```

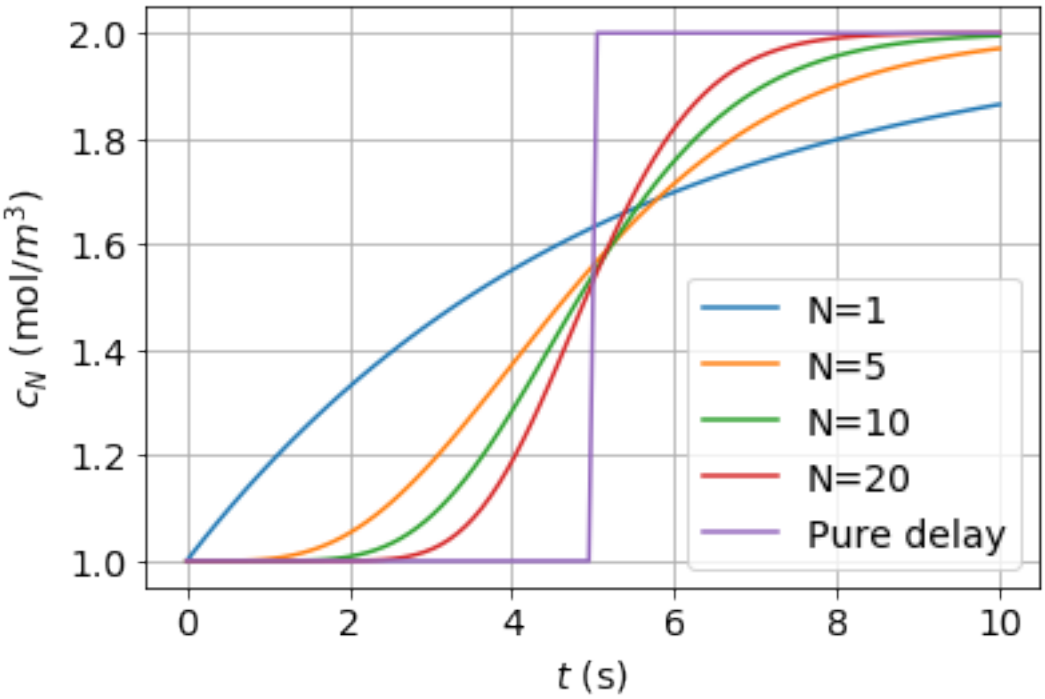
```
Exposing A and B
Creating ABChain
Creating AB0 from AB within ABChain
Creating A from A within ABChain
Creating jA from 013 within ABChain
Creating B from A within ABChain
Creating jB from 013 within ABChain
```

[4]:

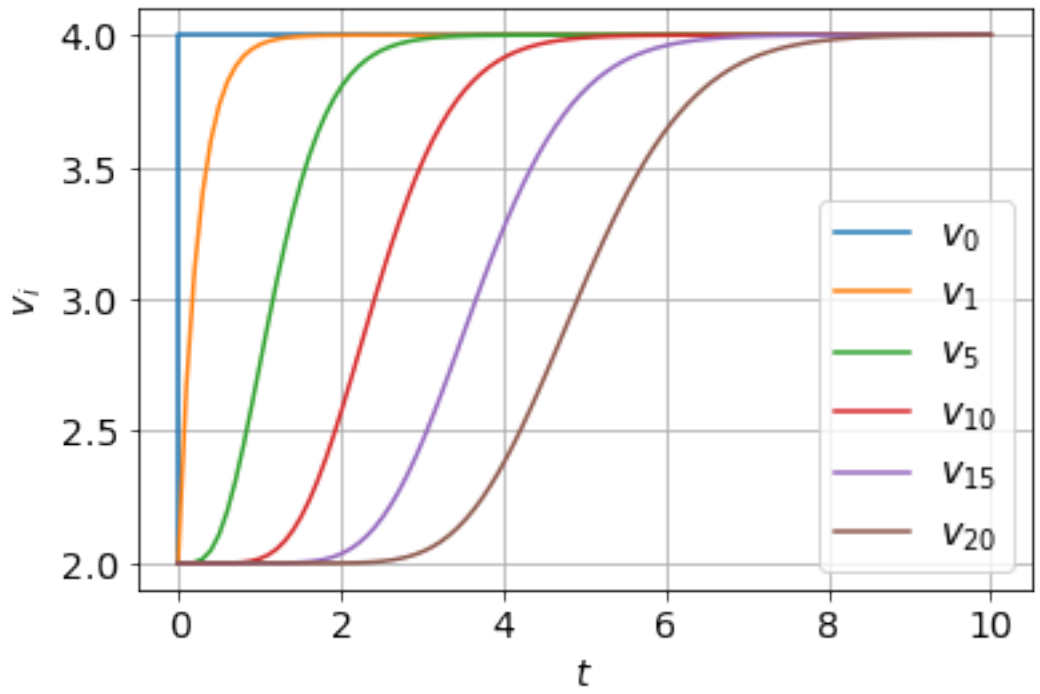
$$A \Leftrightarrow B \tag{2}$$

[5]:
```
## Vary number of lumps
N_lumps = [1,5,10,20]
#N_lumps = [1]
Q0 = 2
V_p = 10
delay = V_p/Q0

## Time
t = np.linspace(0,2*delay,100)

plt.rcParams.update({'font.size': 14})

## Create pipe model Pipe for various numbers of lumps
for N_lump in N_lumps:
```

```
    Pipe = mbg.chain(AB_abg.model(),inport = 'A',outport = 'B',
                     ␣
↪N=N_lump+1,rename_components=True,Comps=['R'],quiet=quiet)
    V_i = V_p/N_lump # Volume of lump
    parameter = {}
    UniDir = []
    for i in range(N_lump+1):
        reac = f'r_{i}'
        UniDir.append(reac)
        kap = f'kappa_{reac}'
        parameter[kap] = Q0

    s = st.stoich(Pipe,UniDir=UniDir, quiet=quiet)
    sc = st.statify(s,chemostats=['A','B'])
    species = s['species']
    disp.Latex(st.sprintrl(s))
    X0 = np.ones(s['n_X'])
    X0[species.index('A')] = 2
    for i in range(N_lump):
        spec = f'IS{i+1}'
        K = f'K_{spec}'
        parameter[K] = 1/V_i
        X0[species.index(spec)] = V_i

    dat = st.sim(s,sc=sc,t=t,parameter=parameter,X0=X0,quiet=quiet)
    r_N = f'r_{N_lump}'
    i_N = species.index(f'IS{N_lump}')
    x_N = dat['X'][:,i_N]


    plt.plot(t,x_N/V_i,label=f'N={N_lump}')

## Pure delay
y_del = 1+np.heaviside(t-delay,1)
plt.plot(t,y_del,label='Pure delay')
plt.grid()
plt.legend()
plt.xlabel('$t$ (s)')
plt.ylabel('$c_N$ (mol/$m^3$)')
Savefig('Figs/Pipe.pdf')
```



[6]:
```
V = dat['V']
reaction = s['reaction']
I = [0,1,5,10,15,20]
for i,reac in enumerate(reaction):
    if i in I:
        v = V[:,i]
        if i == 0:
            vv = np.hstack((2,v))
            tt = np.hstack((0,t))
        else:
            tt = t
            vv = v
        vi = 'v_{'+f'{i}'+'}'
        plt.plot(tt,vv,label=f'${vi}$')

plt.legend()
plt.grid()
plt.xlabel('$t$')
plt.ylabel('$v_i$')
plt.show()
Savefig('Figs/Pipe_v.pdf')
```

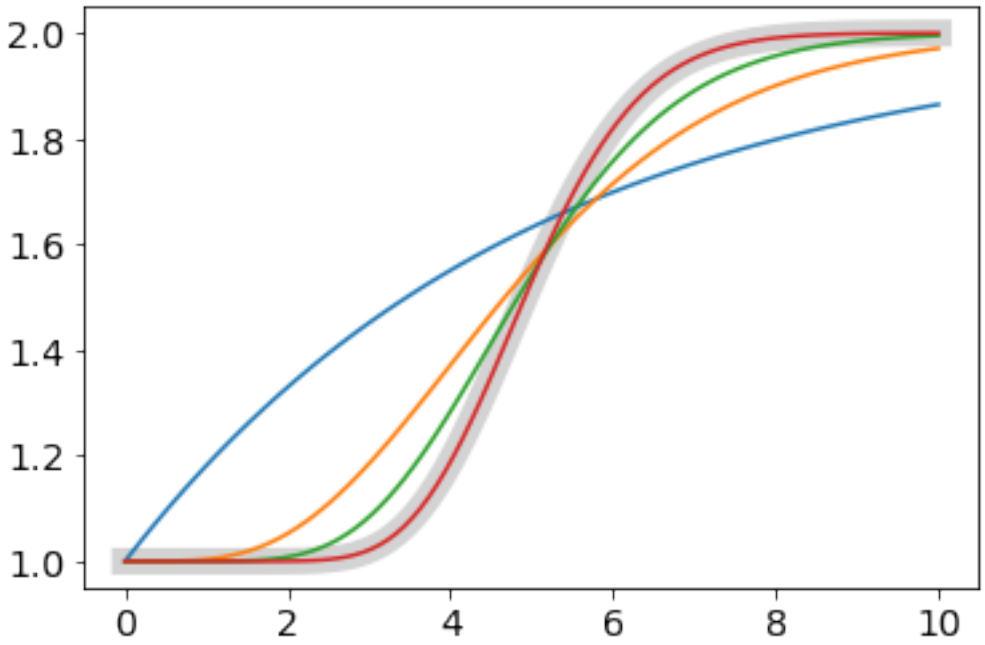



```
<Figure size 432x288 with 0 Axes>
```

[7]:
```
## Compare with a constructed transfer function 1/(s+sT/N)^N
plt.plot(t,x_N/V_i,lw=10,color='lightgrey')
for N_lump in N_lumps:
    lump = con.tf(1,[delay/N_lump,1])
    lumps = con.tf(1,1)
    for i in range(N_lump):
        lumps = con.series(lumps,lump)
#     print(lumps)
    t,y = con.step_response(lumps,T=t)
    plt.plot(t,y+1)
```

```
␣
↪---------------------------------------------------------------------------

RuntimeError                              Traceback (most recent call␣
↪last)

RuntimeError: module compiled against API version 0xe but this version␣
↪of numpy is 0xd


␣
↪---------------------------------------------------------------------------
```

```
       RuntimeError                              Traceback (most recent call␣
↪last)

       RuntimeError: module compiled against API version 0xe but this version␣
↪of numpy is 0xd



      ␣
↪---------------------------------------------------------------------------

       RuntimeError                              Traceback (most recent call␣
↪last)

       RuntimeError: module compiled against API version 0xe but this version␣
↪of numpy is 0xd



      ␣
↪---------------------------------------------------------------------------

       RuntimeError                              Traceback (most recent call␣
↪last)

       RuntimeError: module compiled against API version 0xe but this version␣
↪of numpy is 0xd

/home/peterg/.local/lib/python3.8/site-
packages/scipy/sparse/linalg/matfuncs.py:710: LinAlgWarning: Ill-conditioned
matrix (rcond=9.31351e-22): result may not be accurate.
  return solve(Q, P)
```



[8]:
```
## Redo for particular case
N_lump = 10
UniDir = []
for i in range(N_lump+1):
    reac = f'r_{i}'
Pipe = mbg.chain(AB_abg.model(),inport = 'A',outport = 'B',
                 N=N_lump+1,rename_components=True,Comps=['R'],quiet=quiet)
s = st.stoich(Pipe,UniDir=None, quiet=quiet)
s['name'] = 'Pipe_abg'
stbg.model(s)
disp.Latex(st.sprintrl(s,chemformula=True))
```

[8]:

$$\mathrm{A} \underset{}{\overset{r_0}{\rightleftharpoons}} \mathrm{IS}_1 \tag{3}$$
$$\mathrm{IS}_1 \overset{r_1}{\rightleftharpoons} \mathrm{IS}_2 \tag{4}$$
$$\mathrm{IS}_2 \overset{r_2}{\rightleftharpoons} \mathrm{IS}_3 \tag{5}$$
$$\mathrm{IS}_3 \overset{r_3}{\rightleftharpoons} \mathrm{IS}_4 \tag{6}$$
$$\mathrm{IS}_4 \overset{r_4}{\rightleftharpoons} \mathrm{IS}_5 \tag{7}$$
$$\mathrm{IS}_5 \overset{r_5}{\rightleftharpoons} \mathrm{IS}_6 \tag{8}$$
$$\mathrm{IS}_6 \overset{r_6}{\rightleftharpoons} \mathrm{IS}_7 \tag{9}$$
$$\mathrm{IS}_7 \overset{r_7}{\rightleftharpoons} \mathrm{IS}_8 \tag{10}$$
$$\mathrm{IS}_8 \overset{r_8}{\rightleftharpoons} \mathrm{IS}_9 \tag{11}$$
$$\mathrm{IS}_9 \overset{r_9}{\rightleftharpoons} \mathrm{IS}_{10} \tag{12}$$

$$\mathrm{IS}_{10} \xrightleftharpoons{\mathrm{r}_{10}} \mathrm{B} \tag{13}$$

2.2 Transfer-function analysis

This chain of reactions has liner dynamics which are extracted to give a model in the Python Control Toolbox format. The effective timeconstant is computed from:

$$\tau = \sum_{i=1}^{N} 1/p_i \tag{14}$$

where p_i is the ith pole of the tranfer function relating the input and output of the chain.

[9]:
```
imp.reload(st)
UniDir=[]
for i in range(N_lump+1):
    UniDir.append(f'r_{i}')
print(UniDir)
s = st.stoich(Pipe,UniDir=UniDir, quiet=quiet)
disp.Latex(st.sprintvl(s))
```

```
['r_0', 'r_1', 'r_2', 'r_3', 'r_4', 'r_5', 'r_6', 'r_7', 'r_8', 'r_9', 'r_10']
```

[9]:

$$v_{r0} = K_A \kappa_{r0} x_A \tag{15}$$
$$v_{r1} = K_{IS1} \kappa_{r1} x_{IS1} \tag{16}$$
$$v_{r2} = K_{IS2} \kappa_{r2} x_{IS2} \tag{17}$$
$$v_{r3} = K_{IS3} \kappa_{r3} x_{IS3} \tag{18}$$
$$v_{r4} = K_{IS4} \kappa_{r4} x_{IS4} \tag{19}$$
$$v_{r5} = K_{IS5} \kappa_{r5} x_{IS5} \tag{20}$$
$$v_{r6} = K_{IS6} \kappa_{r6} x_{IS6} \tag{21}$$
$$v_{r7} = K_{IS7} \kappa_{r7} x_{IS7} \tag{22}$$
$$v_{r8} = K_{IS8} \kappa_{r8} x_{IS8} \tag{23}$$
$$v_{r9} = K_{IS9} \kappa_{r9} x_{IS9} \tag{24}$$
$$v_{r10} = K_{IS10} \kappa_{r10} x_{IS10} \tag{25}$$

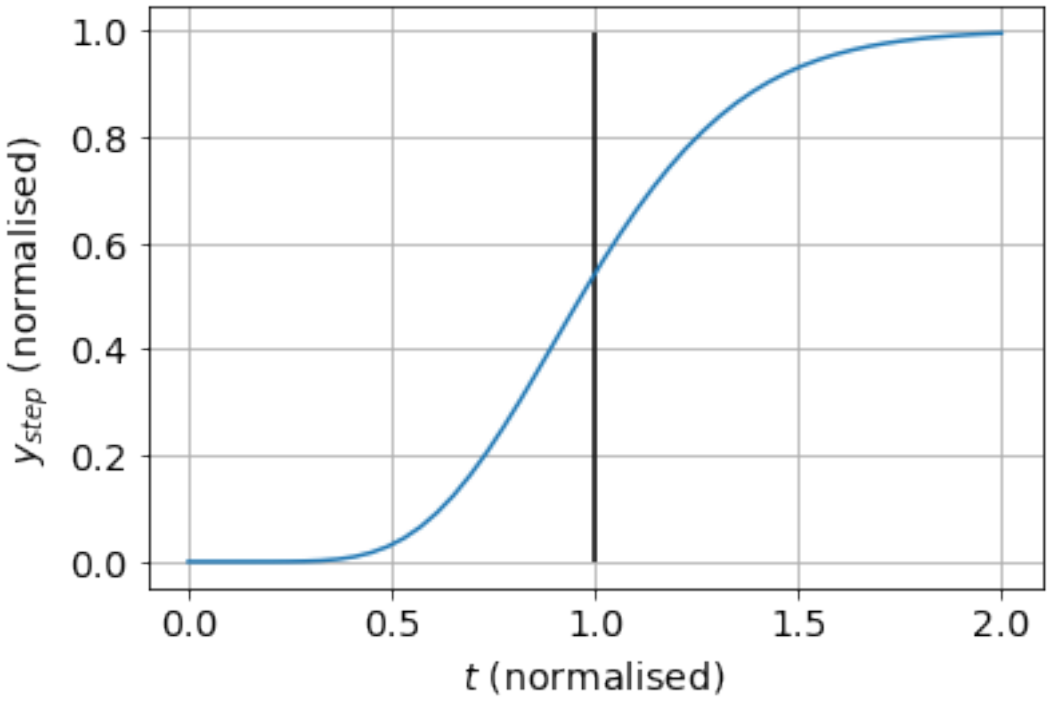
[10]:
```
sc = st.statify(s,chemostats=['A','B'])
sys = st.lin(s,sc)
# print(sys)
A,B,C,D = con.ssdata(sys)
sys12 = con.ss(A,B[:,0],C[N_lump,:],0)
print(con.tf(sys12))
print(con.pole(sys12))
tc = np.sum(-1/con.pole(sys12))
gain = con.dcgain(sys12)
print('Time constant =',tc)
t_step,y_step = con.step_response(sys12,T=np.linspace(0,2*tc))
plt.plot(t_step/tc,y_step.T/gain)
```

```
plt.vlines(1,0,max(y_step)/gain)
plt.grid()
plt.xlabel('$t$ (normalised)')
plt.ylabel('$y_{step}$ (normalised)')
```

```
␣
↪---------------------------------------------------------------------------

RuntimeError                              Traceback (most recent call␣
↪last)

RuntimeError: module compiled against API version 0xe but this version␣
↪of numpy is 0xd
```

```
-3.553e-15 s^9 - 7.105e-15 s^8 - 9.948e-14 s^7 - 2.842e-13 s^6 - 3.979e-13 s^5 -
4.547e-13 s^4 - 2.7e-13 s^3 - 1.137e-13 s^2 - 2.132e-14 s + 1
--------------------------------------------------------------------------------
--------------------------------------------------------------
                         s^10 + 10 s^9 + 45 s^8 + 120 s^7 + 210 s^6 + 252 s^5 +
210 s^4 + 120 s^3 + 45 s^2 + 10 s + 1

[-1. -1. -1. -1. -1. -1. -1. -1. -1. -1.]
Time constant = 10.0
```

[10]: `Text(0,0.5,'$y_{step}$ (normalised)')`



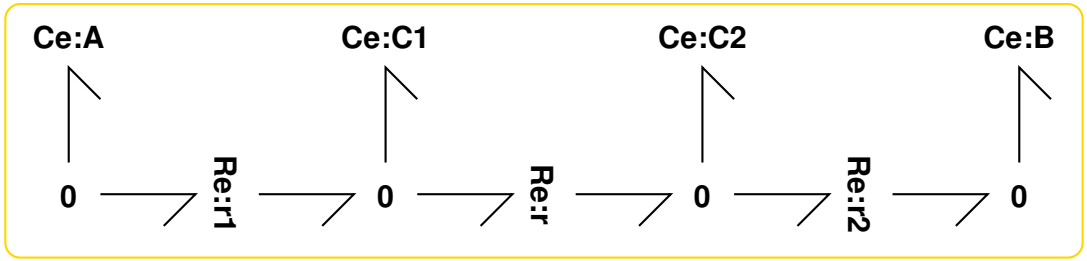
3 Coupled Advection and Transformation

This corresponds to Section 4 of the paper.

3.1 Orifice connection

[11]:
```
##
sbg.model('splitReos_abg.svg')
import splitReos_abg
disp.SVG('splitReos_abg.svg')
```

[11]:



[12]:
```
##Stoichiometry
s = st.stoich(splitReos_abg.model(),quiet=quiet)
chemostats = ['A','B']
sc = st.statify(s,chemostats=chemostats)
disp.Latex(st.sprintrl(s,chemformula=True))
```

[12]:

$$C_1 \underset{}{\overset{r}{\rightleftharpoons}} C_2 \tag{26}$$

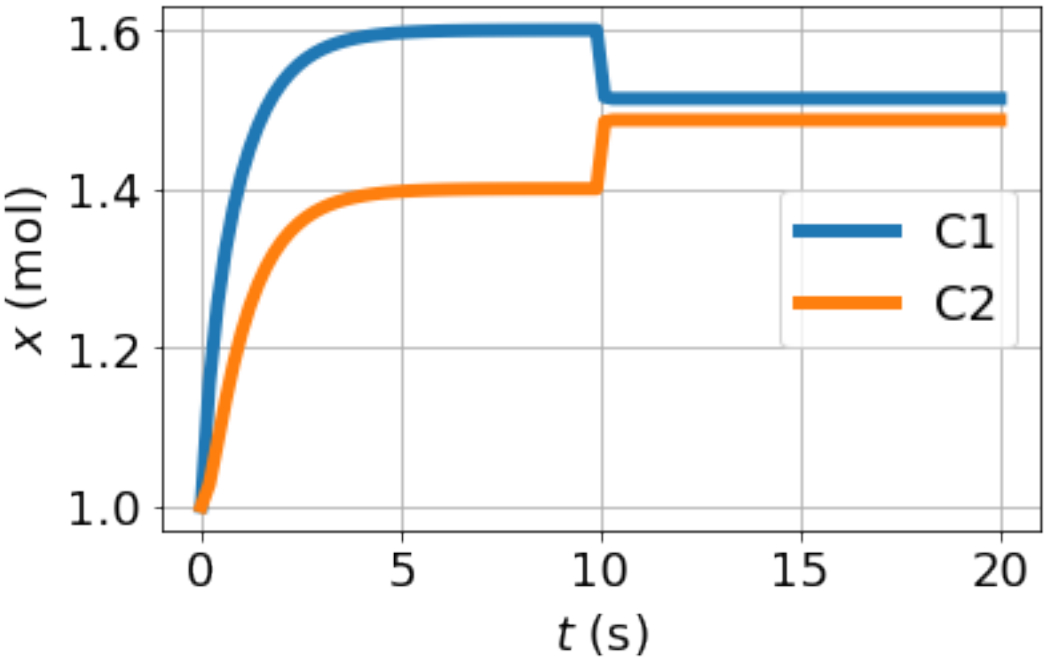
$$A \overset{r_1}{\rightleftharpoons} C_1 \tag{27}$$

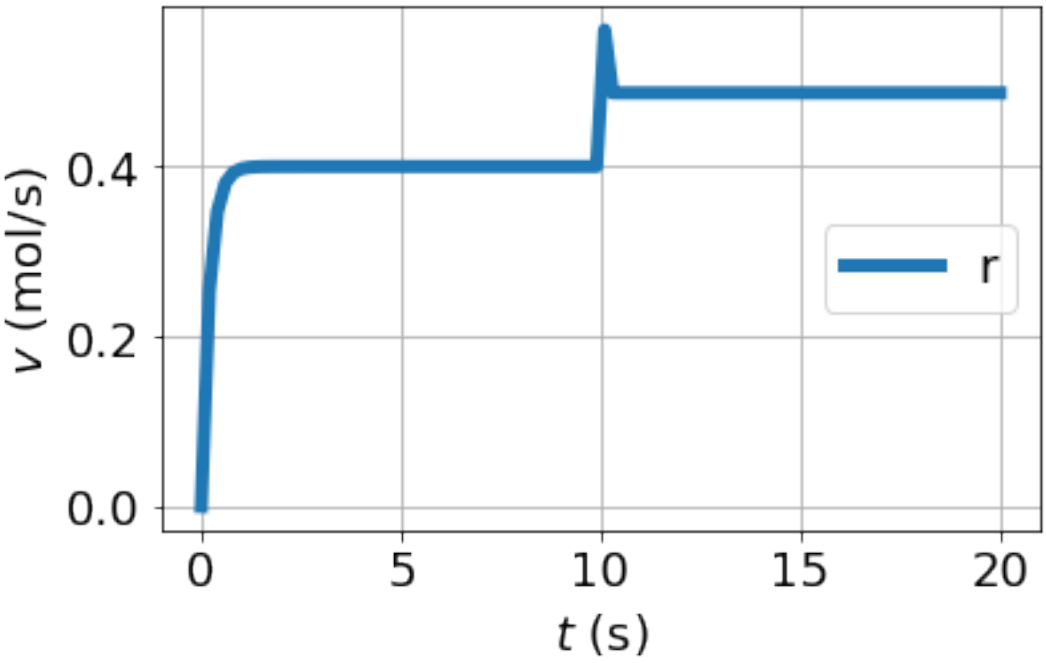
$$C_2 \overset{r_2}{\rightleftharpoons} B \tag{28}$$

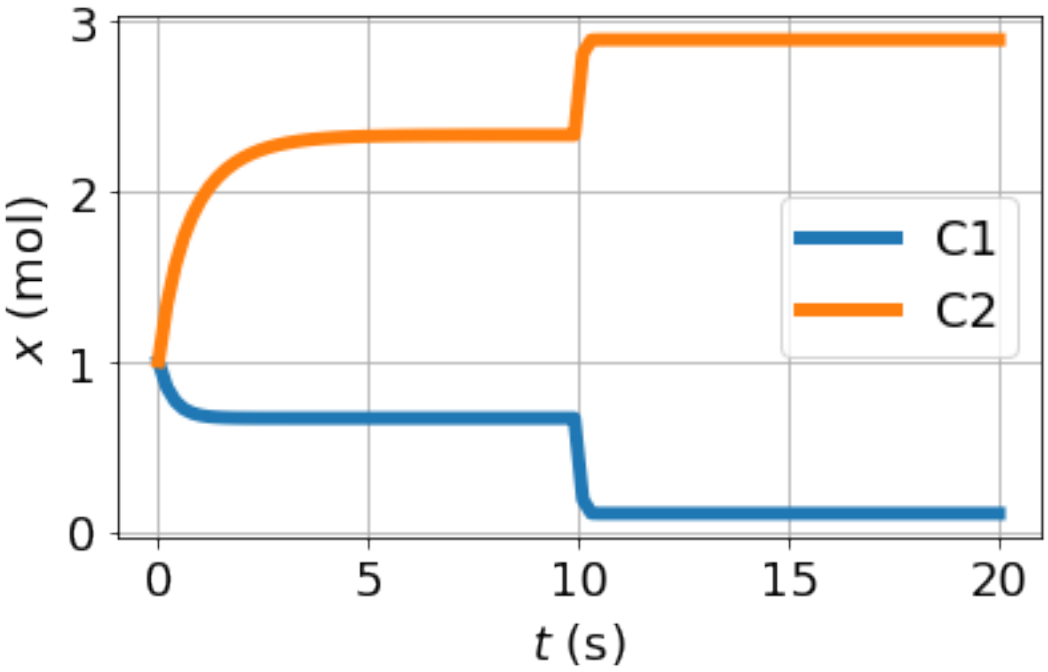
[13]:
```
Q0 = 2
t1 = 10
f1 = 10-Q0
t = np.linspace(0,20,100)
Flow = f'{Q0}*(1+{f1}*np.heaviside(t-{t1},0.5))'
#Flow = f'{Q0}'
print(Flow)
## Simulation
species = s['species']
reaction = s['reaction']

X0 = np.ones(s['n_X'])
X0[species.index('A')] = 2
parameter = {}
Kappa = {}
for unidir in [False,True]:
    if unidir:
```

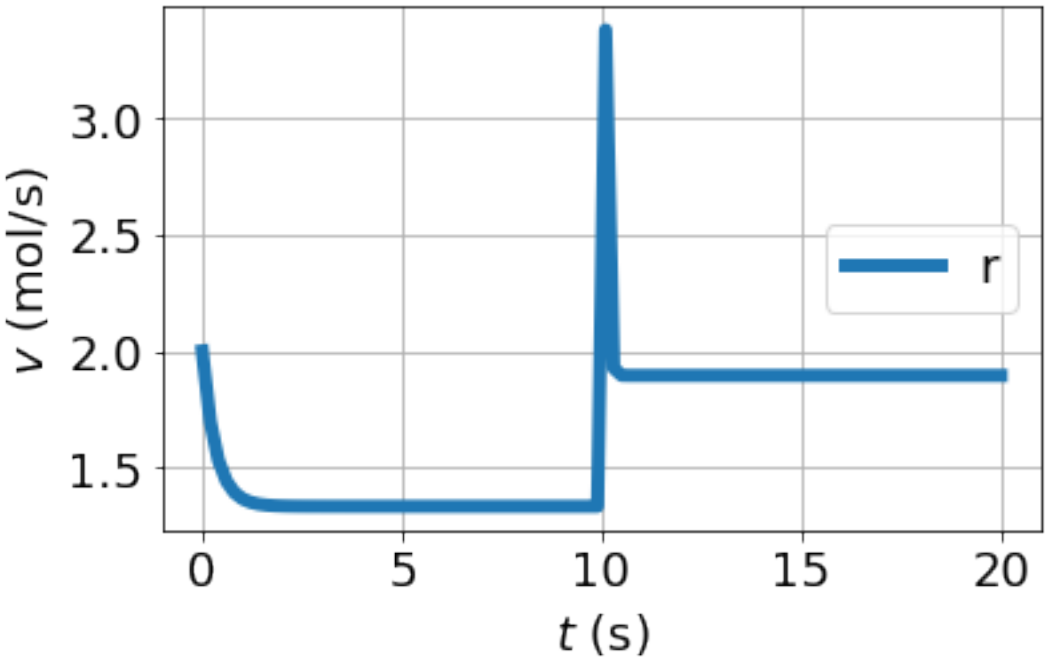
```
        UniDir=['r']
    else:
        UniDir=[]

    parameter['kappa_r1'] = parameter['kappa_r2'] = 1
    parameter['kappa_r'] = Q0
    Kappa['r'] = Flow

    s = st.stoich(splitReos_abg.model(),UniDir=UniDir, quiet=quiet)
    sc = st.statify(s,chemostats=chemostats)
    dat = st.
↪sim(s,sc=sc,t=t,parameter=parameter,Kappa=Kappa,X0=X0,quiet=quiet)

    X_old = {}
    specPlot = ['C1','C2']
    plt.rcParams.update({'font.size': 18})
    for spec in specPlot:
        x = dat['X'][:,species.index(spec)]
        plt.plot(t,x,label=spec,lw=lw)
        X_old[spec] = x
    plt.grid()
    plt.legend(loc='center right')
    plt.xlabel('$t$ (s)')
    plt.ylabel('$x$ (mol)')
    plt.tight_layout()
    name = f'Figs/splitReos_{unidir}'
    Savefig(name+'.pdf')
    plt.show()

    V_old = {}
    reacPlot = ['r']
    for reac in reacPlot:
        v = dat['V'][:,reaction.index(reac)]
        plt.plot(t,v,label=reac,lw=lw)
        V_old[reac] = v
    plt.grid()
    plt.legend(loc='center right')
    plt.xlabel('$t$ (s)')
    plt.ylabel('$v$ (mol/s)')
    plt.tight_layout()

    name = f'Figs/splitReos_{unidir}_v'
    Savefig(name+'.pdf')
    plt.show()

x_B_0 = dat['X'][:,species.index('B')]
#     st.plot(s,dat,species=['A','B','C1','C2'])
#     st.plot(s,dat,species=[],filename=f'{name}_v.pdf')
```

```
2*(1+8*np.heaviside(t-10,0.5))
```


1.6
1.4
1.2
1.0
x (mol)
0
5
10
15
20
t (s)
C1
C2


0.4
0.2
0.0
v (mol/s)
0
5
10
15
20
t (s)
r





3.2 Pipe connection

[14]:
```
## Create Pipe component
N_lump = 5
V_p = 10
```

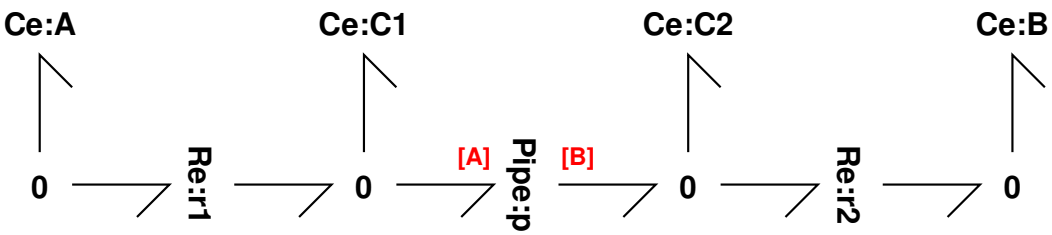
```
Pipe = mbg.chain(AB_abg.model(),inport = 'A',outport = 'B',
                 N=N_lump+1,rename_components=True,Comps=['R'],quiet=quiet)
s = st.stoich(Pipe,UniDir=None, quiet=quiet)
s['name'] = 'Pipe_abg'
stbg.model(s)
import Pipe_abg
imp.reload(Pipe_abg)
```

[14]:
```
<module 'Pipe_abg' from
'/home/peterg/WORK/Research/SystemsBiology/Notes/2022/Advection/Pipe_abg.py'>
```

[15]:
```
## Split enzyme catalysed Reaction
sbg.model('splitRep_abg.svg')
import splitRep_abg
disp.SVG('splitRep_abg.svg')
```

```
Creating subsystem: Pipe:p
```

[15]:



[16]:
```
##Stoichiometry+parameters
parameter = {}
Kappa = {}
UniDir = []
for i in range(N_lump+1):
    reac = f'p_r_{i}'
    UniDir.append(reac)
    kap = f'kappa_{reac}'
    parameter[kap] = Q0
    Kappa[reac] = Flow

s = st.stoich(splitRep_abg.model(),UniDir=UniDir,quiet=quiet)
species = s['species']
sc = st.statify(s,chemostats=chemostats)

## Initial condition
X0 = np.ones(s['n_X'])
X0[species.index('A')] = 2


## Pipe parameters
small = 1e-6
if N_lump>0:
```

```
    V_i = V_p/N_lump # Volume of lump
    for i in range(N_lump):
        spec = f'p_IS{i+1}'
        K = f'K_{spec}'
        parameter[K] = 1/V_i
        X0[species.index(spec)] = small

# print(parameter)
# print(X0)
# print(parameter)
# print(UniDir)
print(Kappa)

disp.Latex(st.sprintrl(s,chemformula=True))
```

```
{'p_r_0': '2*(1+8*np.heaviside(t-10,0.5))', 'p_r_1':
'2*(1+8*np.heaviside(t-10,0.5))', 'p_r_2': '2*(1+8*np.heaviside(t-10,0.5))',
'p_r_3': '2*(1+8*np.heaviside(t-10,0.5))', 'p_r_4':
'2*(1+8*np.heaviside(t-10,0.5))', 'p_r_5': '2*(1+8*np.heaviside(t-10,0.5))'}
```

[16]:

$$C_1 \xrightarrow{p_{r0}} p_I S_1 \tag{29}$$

$$p_I S_1 \xrightarrow{p_{r1}} p_I S_2 \tag{30}$$

$$p_I S_2 \xrightarrow{p_{r2}} p_I S_3 \tag{31}$$

$$p_I S_3 \xrightarrow{p_{r3}} p_I S_4 \tag{32}$$

$$p_I S_4 \xrightarrow{p_{r4}} p_I S_5 \tag{33}$$

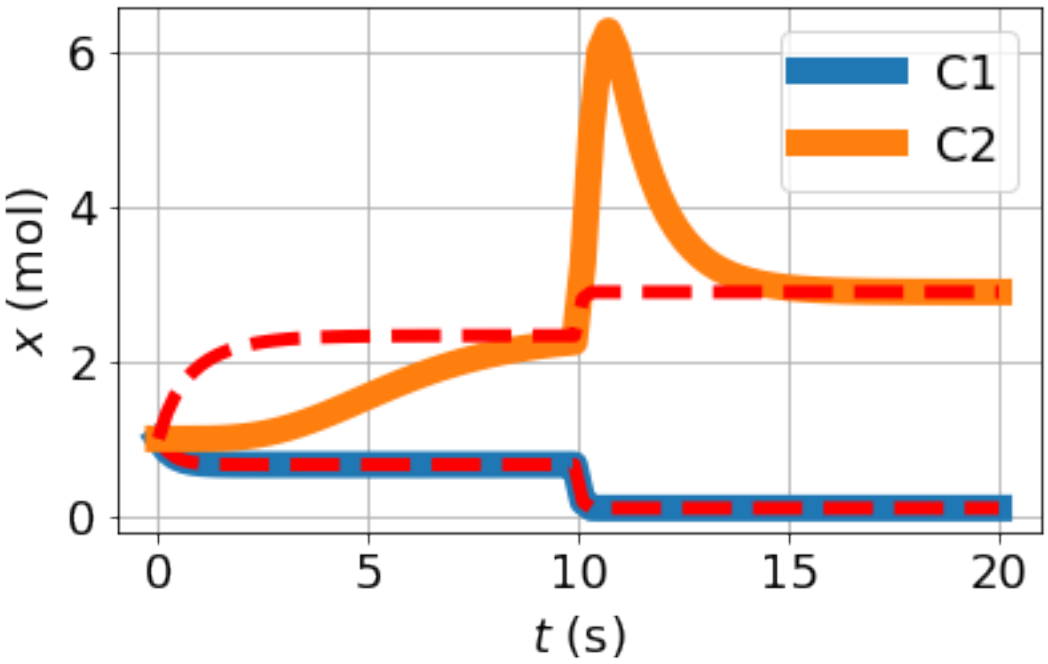
$$p_I S_5 \xrightarrow{p_{r5}} C_2 \tag{34}$$

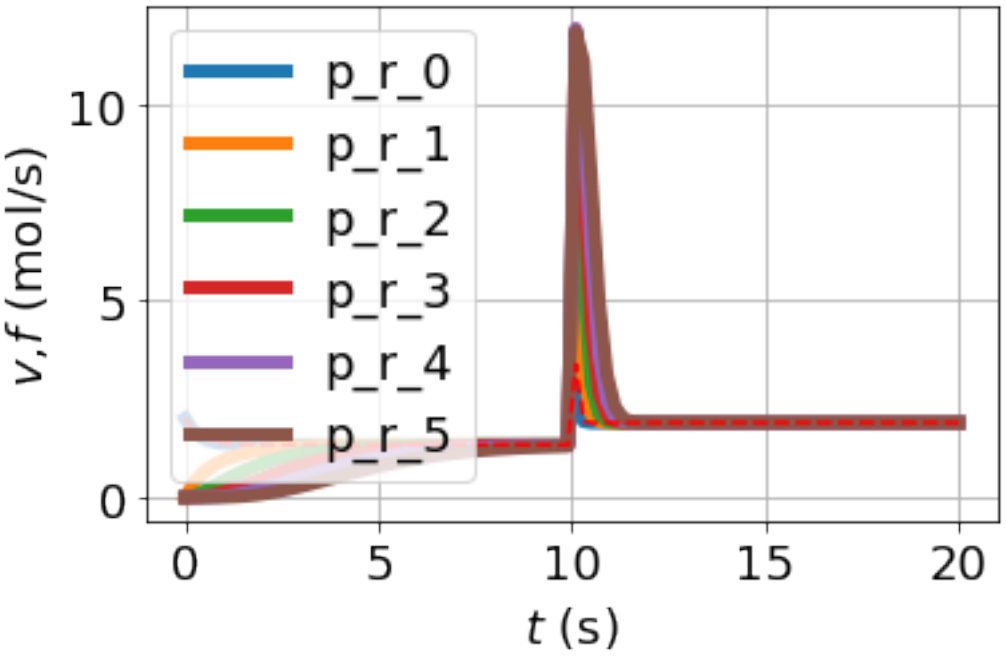
$$A \underset{}{\overset{r_1}{\rightleftharpoons}} C_1 \tag{35}$$

$$C_2 \underset{}{\overset{r_2}{\rightleftharpoons}} B \tag{36}$$

[17]:
```
## Simulate
dat = st.sim(s,sc=sc,t=t,parameter=parameter,X0=X0,Kappa=Kappa,quiet=quiet)

plt.rcParams.update({'font.size': 18})
for spec in specPlot:
    x = dat['X'][:,species.index(spec)]
    plt.plot(t,x,label=spec,lw=10)
    plt.plot(t,X_old[spec],color='red',ls='dashed',lw=5)
plt.grid()
plt.legend()
plt.xlabel('$t$ (s)')
plt.ylabel('$x$ (mol)')
plt.tight_layout()
Savefig('Figs/splitRep.pdf')
plt.show()

reaction = s['reaction']
```

```
for i in range(N_lump+1):
    reac = f'p_r_{i}'
    v = dat['V'][:,reaction.index(reac)]
    plt.plot(t,v,label=reac,lw=lw)
plt.plot(t,V_old['r'],color='red',ls='dashed')
plt.grid()
plt.legend()
plt.xlabel('$t$ (s)')
plt.ylabel('$v$,$f$ (mol/s)')
plt.tight_layout()
name = f'Figs/splitRep_v'
Savefig(name+'.pdf')
plt.show()
```



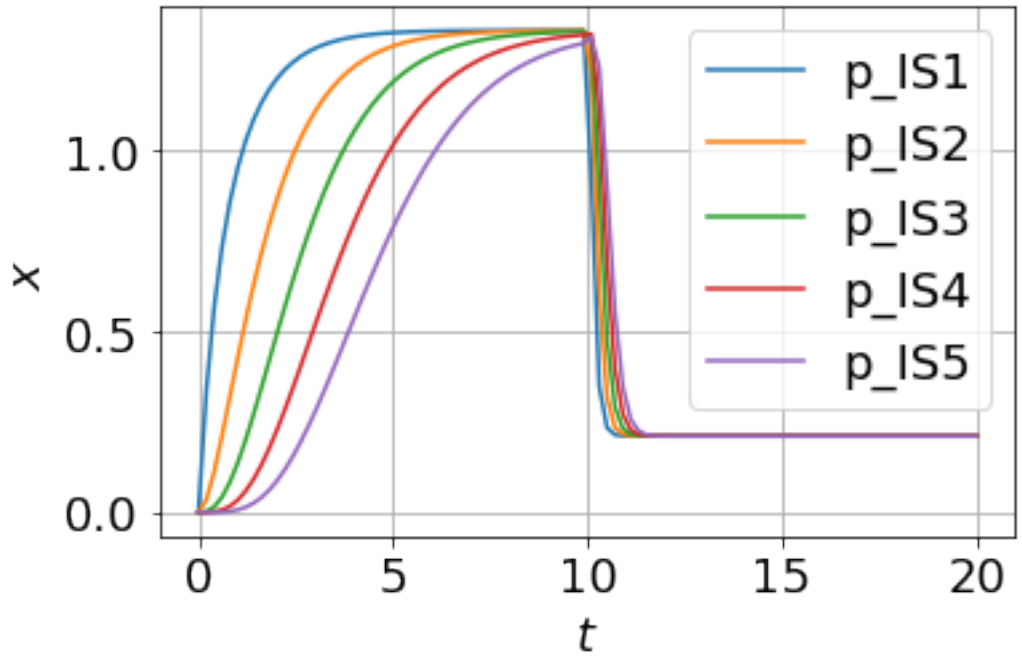


[18]:
```
# st.plot(s,dat,reaction=[],species=['A','B','C1','C2'])
```

[19]:
```
#  st.plot(s,dat,species=[])
```

[20]:
```
# st.plot(s,dat,reaction=[],species=['p_IS1'])
```

[21]:
```
for i,spec in enumerate(species):
    if 'I' in spec:
        x = dat['X'][:,i]
        plt.plot(t,x,label=spec)
plt.grid()
plt.legend()
plt.xlabel('$t$')
plt.ylabel('$x$')
plt.tight_layout()
Savefig('Figs/splitRep_pipe.pdf')
```



4 Enzyme-catalysed reaction

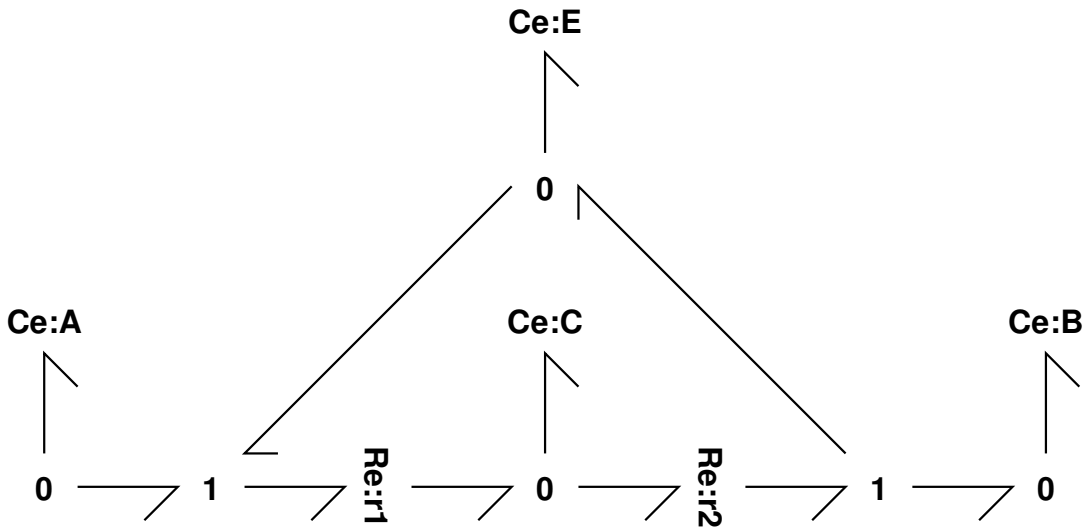
This section reprises the standard bond graph model of an enzyme-catalysed reaction - see ECR. In this case: - Ce_A represents the unbound oxygen entering the blood - Ce_B represents the unbound oxygen leaving the blood - Ce_E represents the unbound haemoglobin - Ce_C represents the haemoglobin bound to oxygen

This corresponds to a simple model of haemoglobin with one binding site (like myoglobin). A more detailed model wouPipe incorporate four cooperative binding sites - see Cooperative.

[ ]:

[22]:
```
## Enzyme catalysed reaction
sbg.model('RE_abg.svg')
import RE_abg
disp.SVG('RE_abg.svg')
```

[22]:



[23]:
```
##Stoichiometry
s = st.stoich(RE_abg.model(),quiet=quiet)
chemostats = ['A','B']
sc = st.statify(s,chemostats=chemostats)
disp.Latex(st.sprintrl(s,chemformula=True))
```

[23]:

$$\mathrm{A} + \mathrm{E} \underset{}{\overset{r_1}{\rightleftharpoons}} \mathrm{C} \tag{37}$$

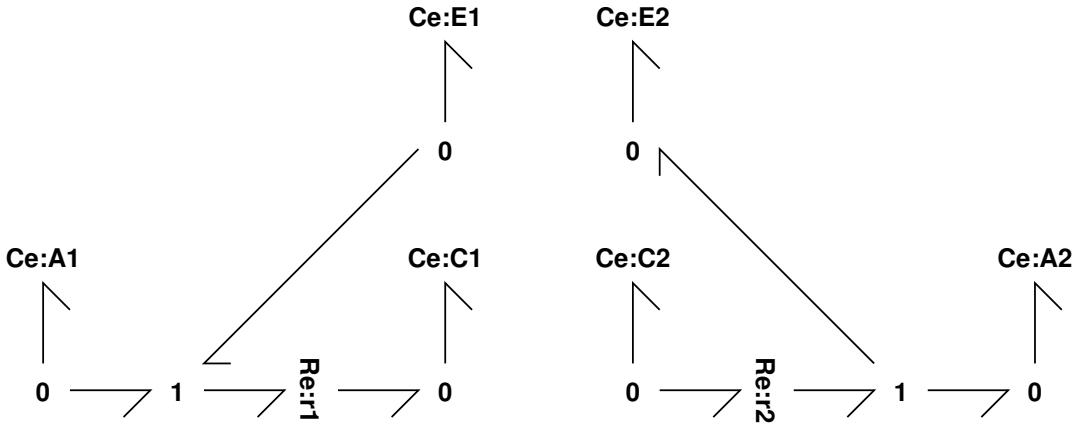
$$\mathrm{C} \overset{r_2}{\rightleftharpoons} \mathrm{B} + \mathrm{E} \tag{38}$$

4.1 Split enzyme-catalysed reaction

In this bond graph, the components Ce_E and Ce_E are split: advection carries the coreponding molecules from one side to the other. An explicit bond graph for this coupling is provided below for tghe case were the advection time-delay is zero; the bond graph for the general case has not yet been determined.

[24]:
```
## Split enzyme catalysed reaction
sbg.model('splitRE_abg.svg')
import splitRE_abg
disp.SVG('splitRE_abg.svg')
```

[24]:



[25]:
```
##Stoichiometry
ss = st.stoich(splitRE_abg.model(),quiet=quiet)
chemostats = ['A1','A2']
ssc = st.statify(ss,chemostats=chemostats)
disp.Latex(st.sprintrl(ss,chemformula=True))
```

[25]:

$$A_1 + E_1 \underset{}{\overset{r_1}{\rightleftharpoons}} C_1 \tag{39}$$

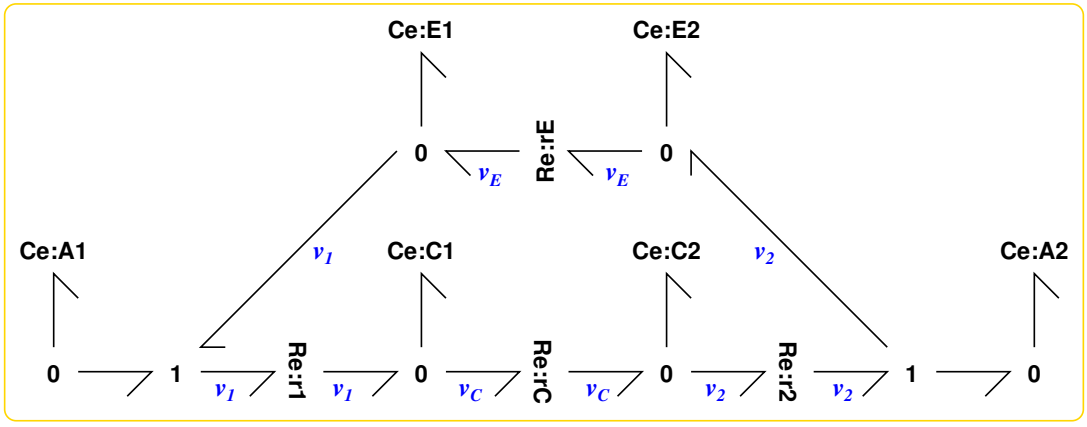
$$C_2 \underset{}{\overset{r_2}{\rightleftharpoons}} A_2 + E_2 \tag{40}$$

5 Circular advection

5.1 Orifice (Symmetric case): Bond graph model

[26]:
```
## Split enzyme catalysed reaction with symmetric connection
sbg.model('splitREos_abg.svg')
import splitREos_abg
disp.SVG('splitREos_abg.svg')
```

[26]:



[27]:
```
##Stoichiometry
snd = st.stoich(splitREos_abg.model(),quiet=quiet)
chemostats = ['A1','A2']
sndc = st.statify(snd,chemostats=chemostats)
print(snd['species'])
print(snd['reaction'])
disp.Latex(st.sprintrl(snd,chemformula=True))
```

```
['A1', 'A2', 'C1', 'C2', 'E1', 'E2']
['r1', 'r2', 'rC', 'rE']
```

[27]:

$$\mathrm{A_1 + E_1} \xrightleftharpoons{\mathrm{r_1}} \mathrm{C_1} \tag{41}$$

$$\mathrm{C_2} \xrightleftharpoons{\mathrm{r_2}} \mathrm{A_2 + E_2} \tag{42}$$

$$\mathrm{C_1} \xrightleftharpoons{\mathrm{rC}} \mathrm{C_2} \tag{43}$$

$$\mathrm{E_2} \xrightleftharpoons{\mathrm{rE}} \mathrm{E_1} \tag{44}$$

5.1.1 Simulate

Note that - the final steady state of the two cases $d = 0$ and $d > 0$ are the same - if the simulation is repeated with $f_0 = 100$, the $x_{C1} \approx x_{C2}$ and $x_{E1} \approx x_{E2}$ as predicted.

The two chemostats corresponding to A1 and A2 are:

$$x_{A1} = \begin{cases} 1 & \text{if } t < 1 \\ 2 & \text{if } t \geq 1 \end{cases} \tag{45}$$

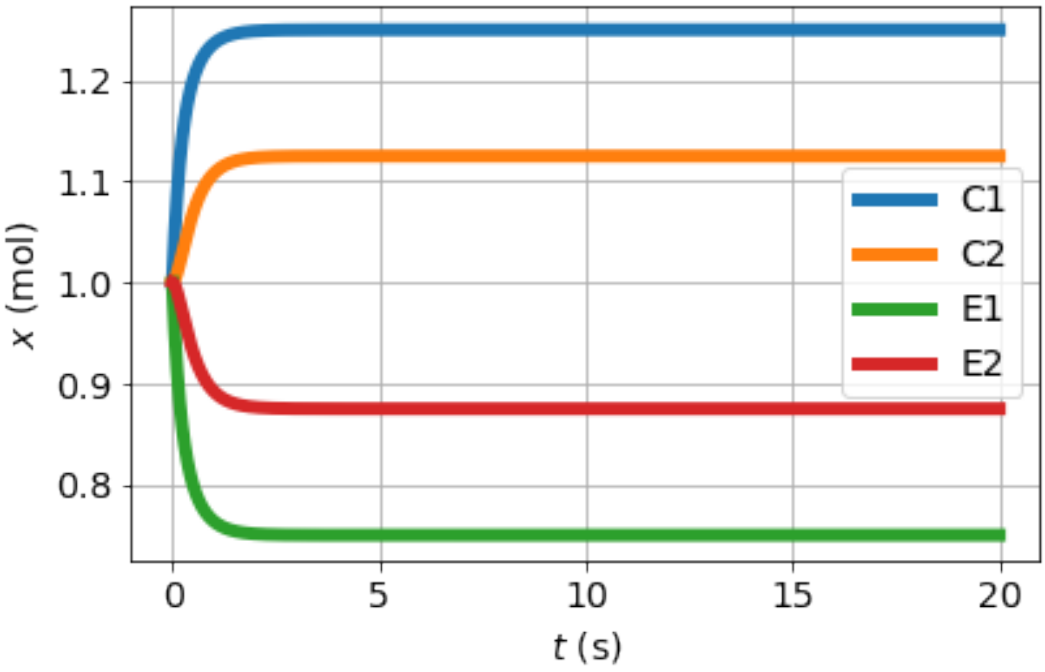
$$x_{A2} = 1 \tag{46}$$

[28]:
```
## Simulate split ECR with no delay
i_chemo = []
for chemo in chemostats:
    i_chemo.append(snd['species'].index(chemo))
# print(i_chemo)
## Time
tt = np.linspace(0, 20,10000)

## Parameters
parameter = {}
parameter['kappa_rC'] = Q0
parameter['kappa_rE'] = Q0

# parameter['K_E'] = 0.5
# parameter['K_C'] = 0.5
## Chemostats
t_0 = 0
X_chemo = {'A1':f'1 + np.heaviside(t-{t_0},1)', 'A2':'1'}

## Simulate
```

```
dat = st.sim(snd,sc=sndc,t=tt,parameter=parameter,X_chemo=X_chemo,quiet=False)
XX0 = dat['X'][:,2:6]
Legend = snd['species'][2:6]
V0 = dat['V'][:,0:2]
dX0 = dat['dX'][:,i_chemo]
```

```
Setting kappa_rC to 2
Setting kappa_rE to 2
```

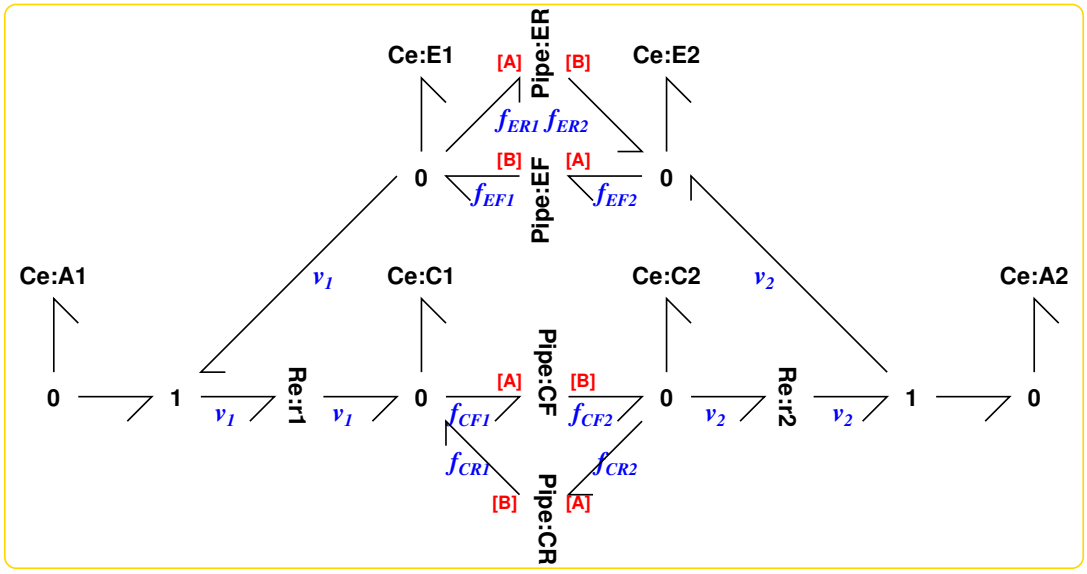
[29]:
```
## Plot
plt.rcParams.update({'font.size': 14})
## Orifice
plt.plot(tt,XX0,lw=lw)

plt.legend(Legend)
plt.grid()
plt.xlabel('$t$ (s)')
plt.ylabel('$x$ (mol)')
plt.tight_layout()
Savefig('Figs/OrificeSim.pdf')
```



5.2 Circular advection: pipe connection

[30]:
```
disp.SVG('splitREp_abg.svg')
```

[30]:

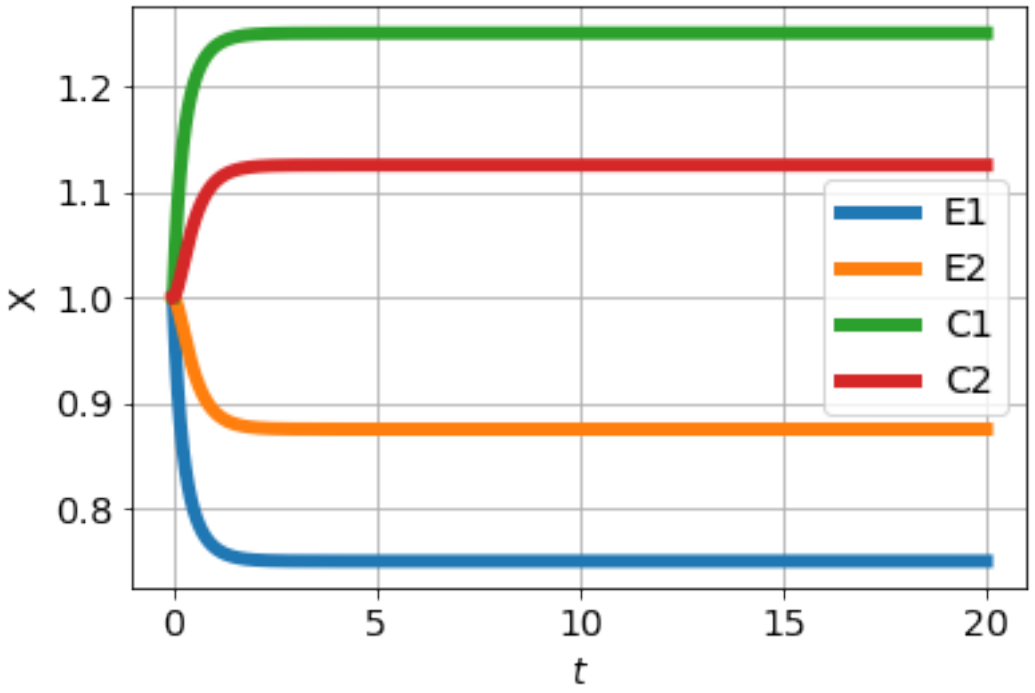


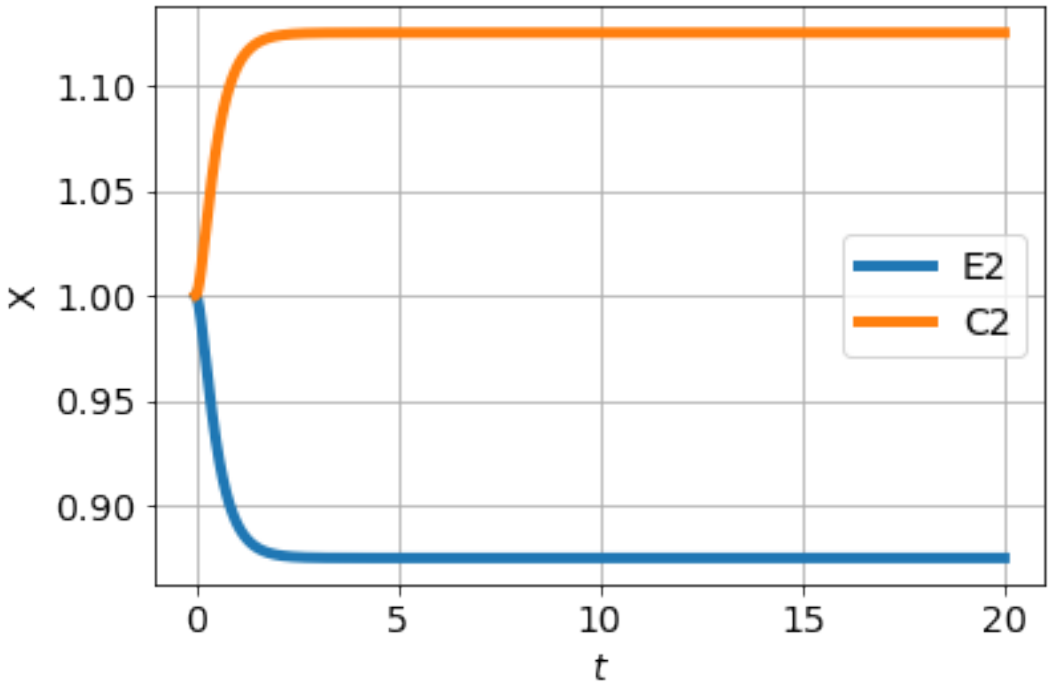
[31]:
```
def CircularAdvection(N_lump=5,Flow=f'{Q0}', quiet=True):

    ## Create Pipe component
    ## Note: N_lump = 0 corresponds to orifice case.
    Pipe = mbg.chain(AB_abg.model(),inport = 'A',outport = 'B',
                     ␣
↪N=N_lump+1,rename_components=True,Comps=['R'],quiet=quiet)
    s = st.stoich(Pipe,UniDir=None, quiet=quiet)
    s['name'] = 'Pipe_abg'
    stbg.model(s)

    ## Set uni directional Re components
    UniDir = []
    for i in range(N_lump+1):
        for comp in ['EF','ER','CF','CR']:
            ud = f'{comp}_r_{i}'
            UniDir.append(ud)

    # print(UniDir)
    ## Create stoichiometry
    sbg.model('splitREp_abg.svg')
    import Pipe_abg
    imp.reload(Pipe_abg)
    import splitREp_abg
    imp.reload(splitREp_abg)
    sPipe = st.stoich(splitREp_abg.model(),UniDir=UniDir,quiet=quiet)

    if not quiet:
        disp.Latex(st.sprintrl(sPipe,chemformula=True))

    ## Setup parameters
    parameter = {}
```

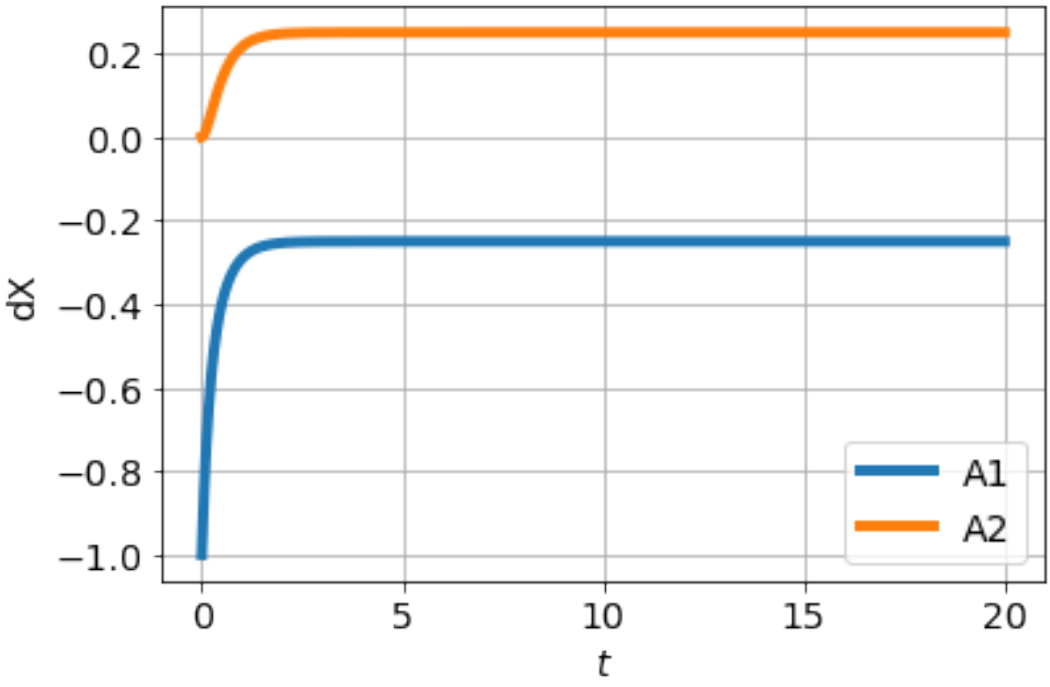
```
    Kappa = {}

    ## Assume K=KK is same for all species
    ## Note that in the simulation code K corresponds to KK/V_i
    KK = 1

    for i in range(N_lump+1):
        for comp in ['EF','ER','CF','CR']:
            kap = f'{comp}_r_{i}'
            Kappa[kap] = f'({Flow})/{KK}'
            #parameter[kap] = Q0
            if i>0:
                V_i = V_p/N_lump
                K0 = KK/V_i
                K0F = K0
                K0R = K0
                K = f'K_{comp}_IS{i}'
                if 'F' in comp:
                    parameter[K] = K0F
                else:
                    parameter[K] = K0R


    ## Setup initial states
    X0 = []
    small = 1e-6
    for spec in sPipe['species']:
    #     print(spec)
        if 'IS' in spec:
            K = f'K_{spec}'
            Xi = 1/parameter[K]
#             Xi = small
        else:
            Xi = 1
        X0.append(Xi)

    X0 = np.array(X0)

#     print(Kappa)
#     print(sPipe['reaction'])
    ## Simulate
    chemostats=['A1','A2']
    i_chemo = []
    for chemo in chemostats:
        i_chemo.append(sPipe['species'].index(chemo))
    # print(i_chemo)
    sPipec = st.statify(sPipe,chemostats=chemostats)
    dat = st.
 ↪sim(sPipe,sc=sPipec,t=tt,parameter=parameter,Kappa=Kappa,X0=X0,X_chemo=X_chemo,quiet=quiet

    return sPipe,dat,i_chemo
```

[32]:
```
#Orifice (Pipe with no lumps)
N_lump = 0
sPipe0,dat0,i_chemo0 = CircularAdvection(N_lump=N_lump)

print('N_lump =', N_lump)

plt.rcParams.update({'font.size': 14})

st.plot(sPipe0,dat0,species=['E1','E2','C1','C2'], reaction = [],lw=5)
#st.plot(sPipe,dat,species=[], reaction = ['PipeEF_r_0','PipeCF_r_0'])
st.plot(sPipe0,dat0,species=['E2','C2'], reaction = [])
st.plot(sPipe0,dat0,dX=True,species=['A1','A2'], reaction = [])
```

```
Creating subsystem: Pipe:CF
Creating subsystem: Pipe:CR
Creating subsystem: Pipe:EF
Creating subsystem: Pipe:ER
N_lump = 0
```







[33]:
```
# ## Show final states
# final0 = dat0['X'][-1,:]
# for i,spec in enumerate(sPipe0['species']):
#     print(f'Final value {spec} = {final0[i]:.2f}')
```

[34]:
```
## Show reactions
disp.Latex(st.sprintrl(sPipe0))
```

[34]:

$$C1 \rightarrow C2 \tag{47}$$
$$C2 \rightarrow C1 \tag{48}$$
$$E2 \rightarrow E1 \tag{49}$$
$$E1 \rightarrow E2 \tag{50}$$
$$A1 + E1 \Leftrightarrow C1 \tag{51}$$
$$C2 \Leftrightarrow A2 + E2 \tag{52}$$

[35]:
```
##Show flow equations
disp.Latex(st.sprintvl(sPipe0))
```

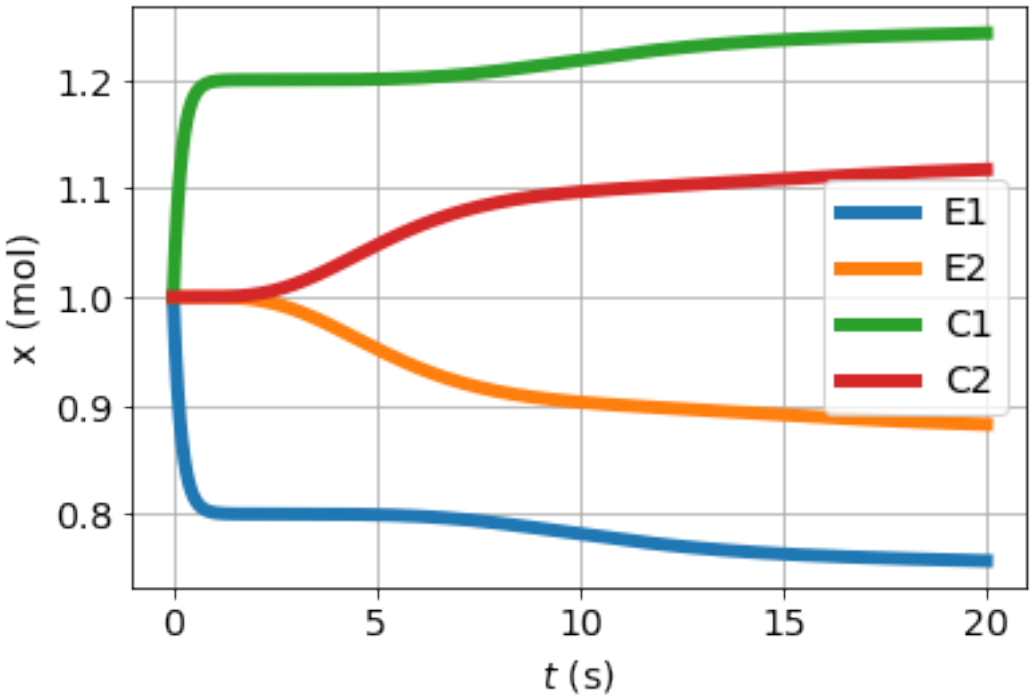
[35]:

$$v_{CFr0} = K_{C1}\kappa_{CFr0}x_{C1} \tag{53}$$
$$v_{CRr0} = K_{C2}\kappa_{CRr0}x_{C2} \tag{54}$$
$$v_{EFr0} = K_{E2}\kappa_{EFr0}x_{E2} \tag{55}$$
$$v_{ERr0} = K_{E1}\kappa_{ERr0}x_{E1} \tag{56}$$
$$v_{r1} = \kappa_{r1}\left(K_{A1}K_{E1}x_{A1}x_{E1} - K_{C1}x_{C1}\right) \tag{57}$$
$$v_{r2} = \kappa_{r2}\left(-K_{A2}K_{E2}x_{A2}x_{E2} + K_{C2}x_{C2}\right) \tag{58}$$

[36]:
```
#Pipe
N_lump = 5
sPipe,dat,i_chemo = CircularAdvection(N_lump=N_lump)

print('N_lump =', N_lump)
```

```
Creating subsystem: Pipe:CF
Creating subsystem: Pipe:CR
Creating subsystem: Pipe:EF
Creating subsystem: Pipe:ER
N_lump = 5
```

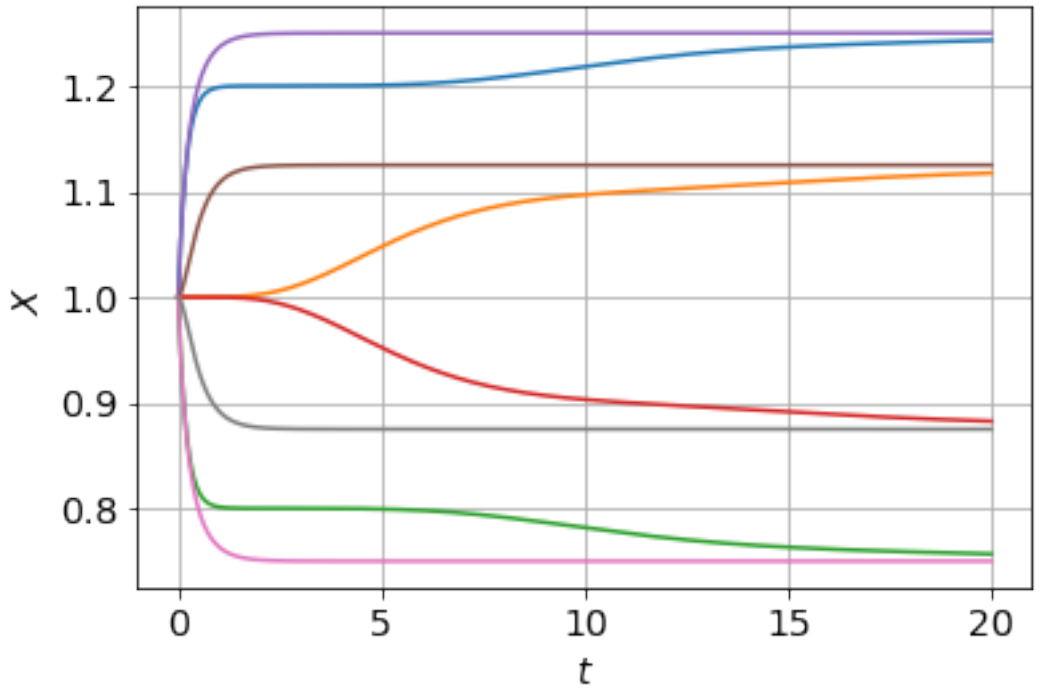
[37]:
```
#Pipe - plot
imp.reload(st)
st.plot(sPipe,dat,species=['E1','E2','C1','C2'], reaction = [],lw=lw,
        xlabel='$t$ (s)',ylabel='x (mol)')
#st.plot(sPipe,dat,species=[], reaction = ['PipeEF_r_0','PipeCF_r_0'])
#st.plot(sPipe,dat,species=['E2','C2'], reaction = [])
#st.plot(sPipe,dat,dX=True,species=['A1','A2'], reaction = [])
```



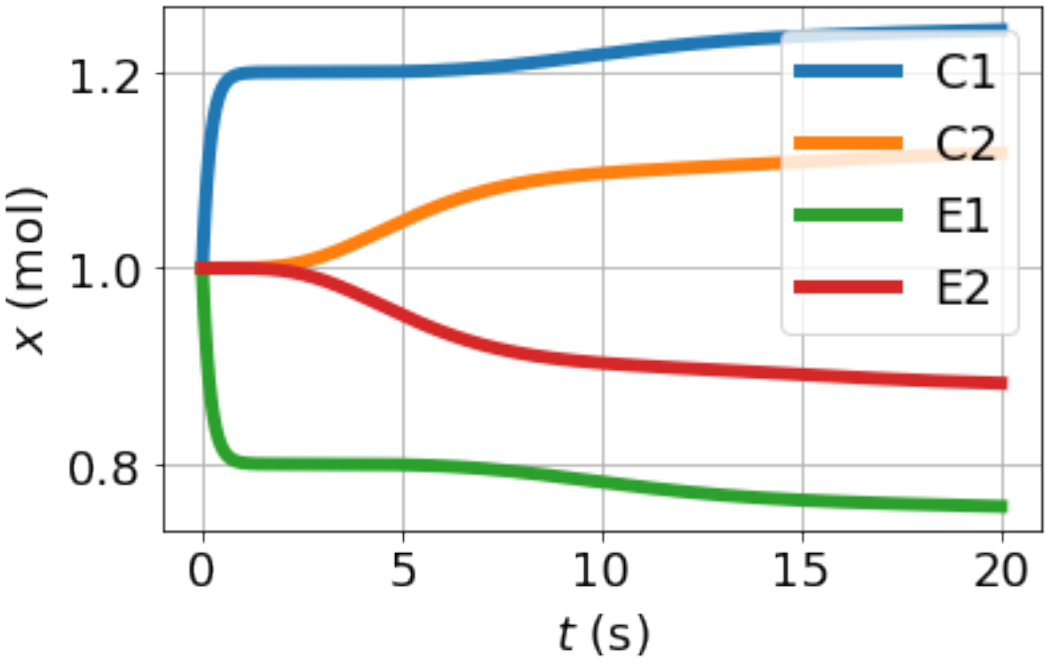
[38]:
```
# ## Show final states
# final = dat['X'][-1,:]
# for i,spec in enumerate(sPipe['species']):
#     print(f'Final value {spec} = {final[i]:.3f}')
```

[39]:
```
## Extract data
XXPipe = dat['X'][:,2:6]
VPipe = dat['V'][:,[0,N_lump+1]]
dXPipe = dat['dX'][:,i_chemo]
dXPipe0 = dat0['dX'][:,i_chemo]

## Plot and compare
## Pipe
plt.plot(tt,XXPipe)

## Orifice
plt.plot(tt,XX0)

plt.grid()
plt.xlabel('$t$')
plt.ylabel('$X$')
```

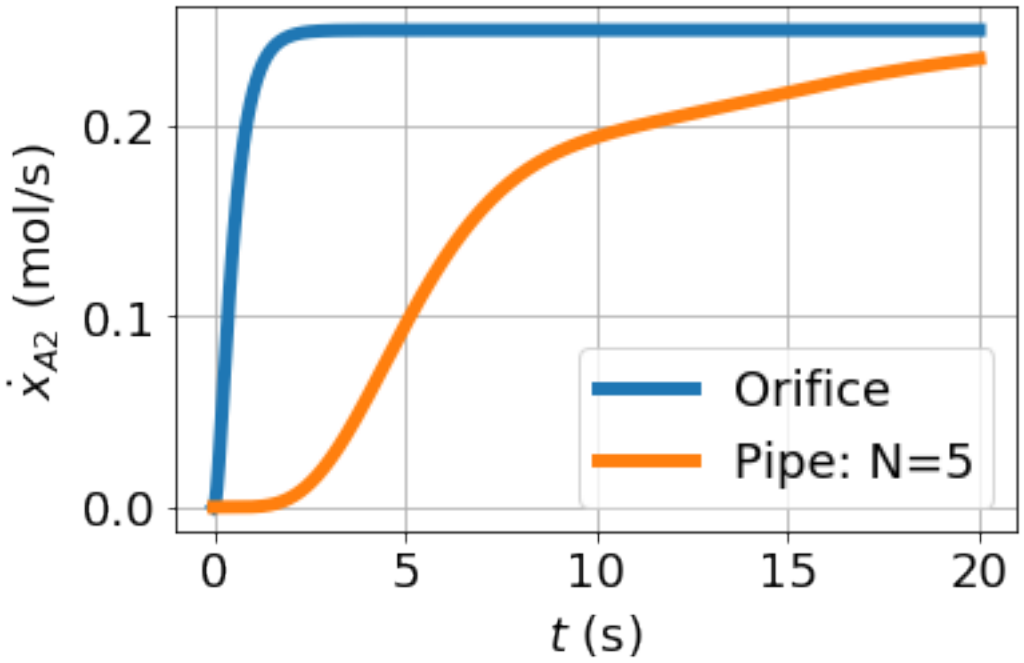
[39]:
```
Text(0,0.5,'$X$')
```



[40]:
```
## Plot

## Pipe
## Font
plt.rcParams.update({'font.size': 18})
plt.plot(tt,XXPipe,lw=lw)

plt.legend(Legend)
plt.grid()
plt.xlabel('$t$ (s)')
plt.ylabel('$x$ (mol)')
plt.tight_layout()
Savefig('Figs/PipeSim.pdf')
```



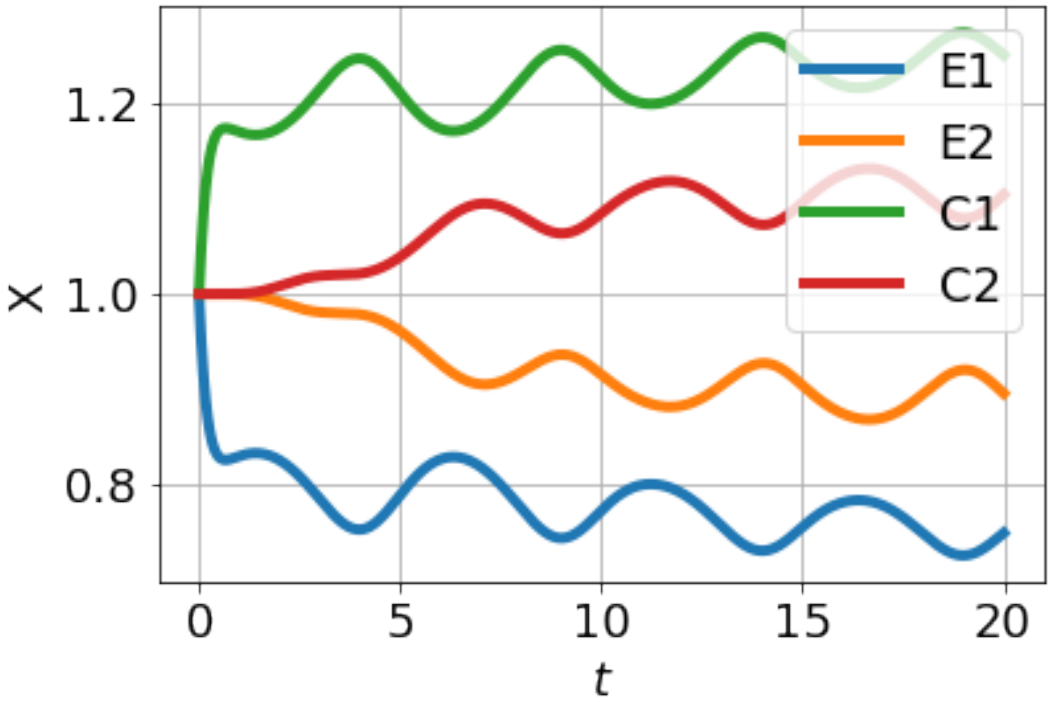
[41]:
```
## Plot Chemostat flows
##Orifice
plt.plot(tt,dX0[:,1],label='Orifice',lw=lw)

## Pipe
#plt.plot(tt,dXPipe0[:,1],label='Pipe: N=0',linestyle='dashed',color='black')
plt.plot(tt,dXPipe[:,1],label=f'Pipe: N={N_lump}',lw=lw)

plt.grid()
plt.legend()
plt.xlabel('$t$ (s)')
plt.ylabel('$\dot{x}_{A2}$ (mol/s)')
plt.tight_layout()
Savefig('Figs/PipeSim_flow.pdf')
```

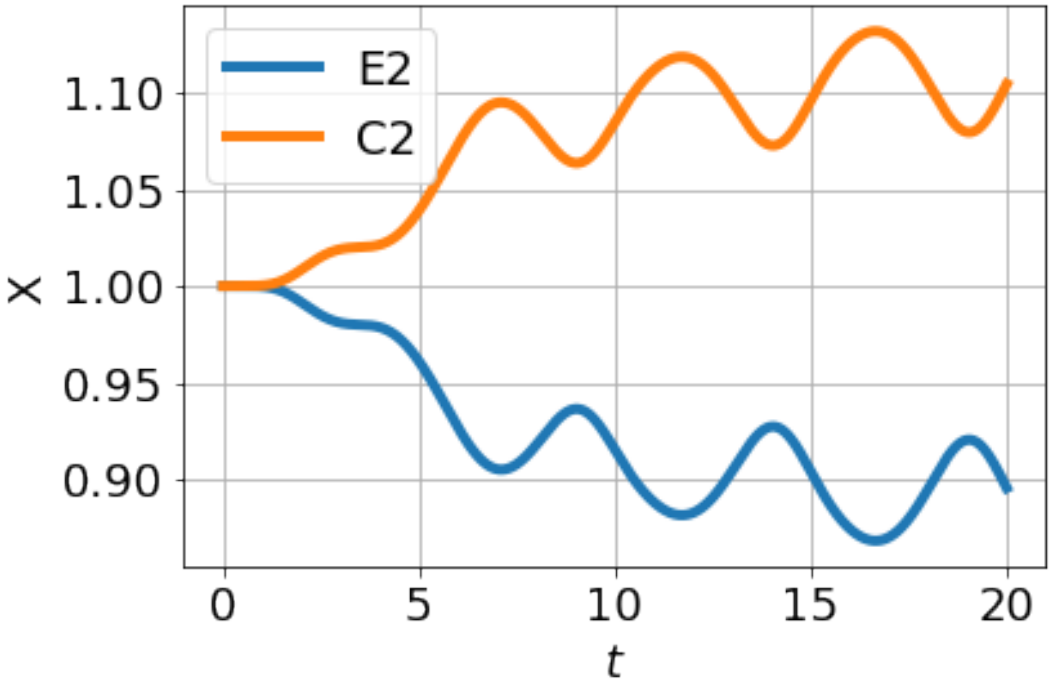


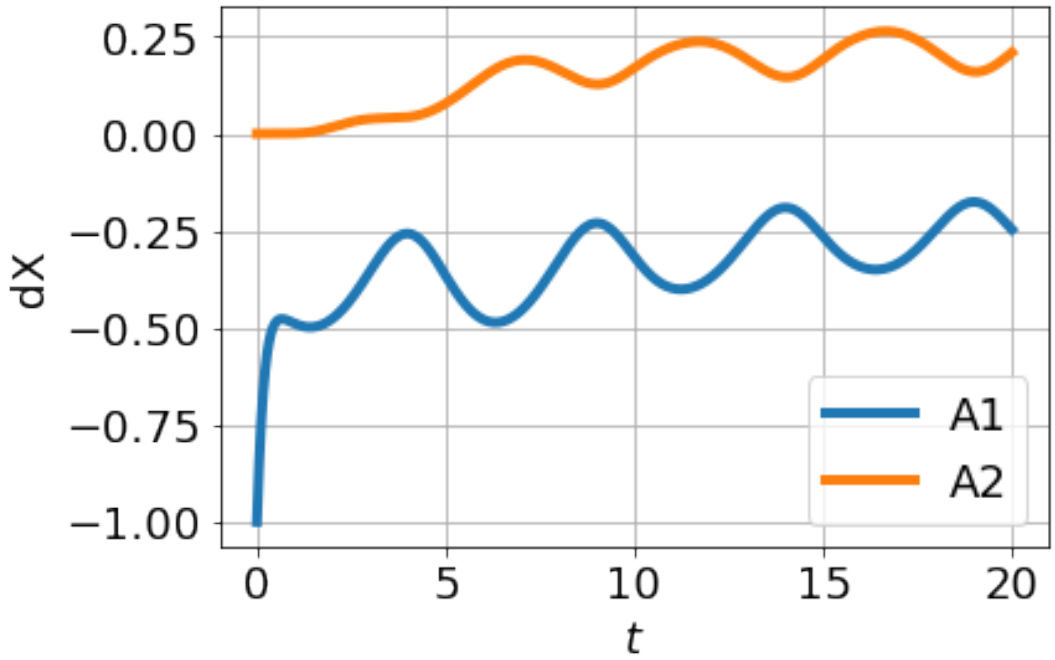
6 Variable flow rate

[42]:
```
N_lump = 5
Period = 5
Flow = f'({Q0})*(1+0.5*np.sin(2*np.pi*t/{Period}))'
print('Flow:',Flow)
sPipe,dat,i_chemo = CircularAdvection(N_lump=N_lump,Flow=Flow)

print('N_lump =', N_lump)
st.plot(sPipe,dat,species=['E1','E2','C1','C2'], reaction = [])
#st.plot(sPipe,dat,species=[], reaction = ['PipeEF_r_0','PipeCF_r_0'])
st.plot(sPipe,dat,species=['E2','C2'], reaction = [])
st.plot(sPipe,dat,dX=True,species=['A1','A2'], reaction = [])
```

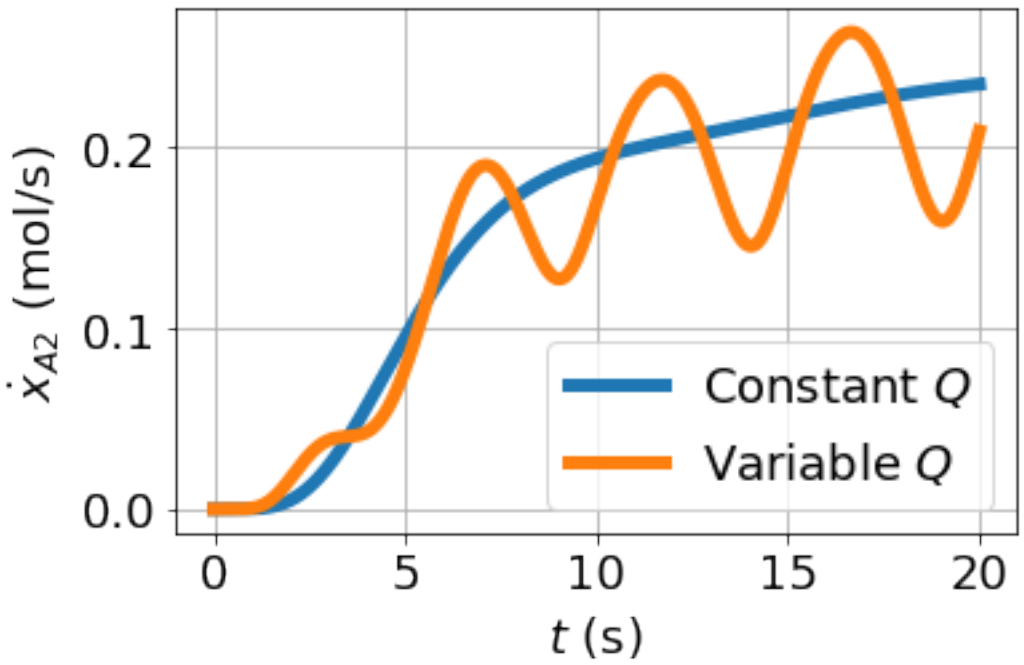
```
Flow: (2)*(1+0.5*np.sin(2*np.pi*t/5))
Creating subsystem: Pipe:CF
Creating subsystem: Pipe:CR
Creating subsystem: Pipe:EF
Creating subsystem: Pipe:ER
N_lump = 5
```


E1
E2
C1
C2
1.2
1.0
0.8
x
0
5
10
15
20
t


E2
C2
1.10
1.05
1.00
0.95
0.90
x
0
5
10
15
20
t



[43]:
```
## Extract data
XXPipev = dat['X'][:,2:6]
VPipev = dat['V'][:,[0,N_lump+1]]
dXPipev = dat['dX'][:,i_chemo]


## Plot Chemostat flows

## Pipe
## Font
plt.rcParams.update({'font.size': 18})

plt.plot(tt,dXPipe[:,1],label='Constant $Q$',lw=lw)
plt.plot(tt,dXPipev[:,1],label='Variable $Q$',lw=lw)
#plt.plot(tt,Q/4,ls='dashed',label='$Q/4$')
plt.grid()
plt.legend()
plt.xlabel('$t$ (s)')
plt.ylabel('$\dot{x}_{A2}$ (mol/s)')
plt.tight_layout()
Savefig('Figs/PipeSim_flow_vary.pdf')
```

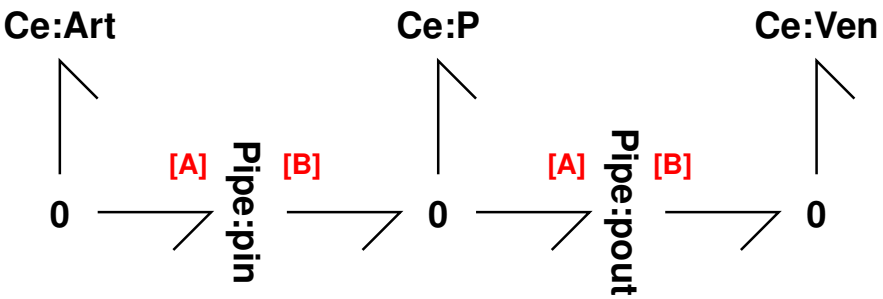


7 Pharmacokinetics

The data is taken from Mapleson (1973).

[44]:
```
disp.SVG('Pool_abg.svg')
```

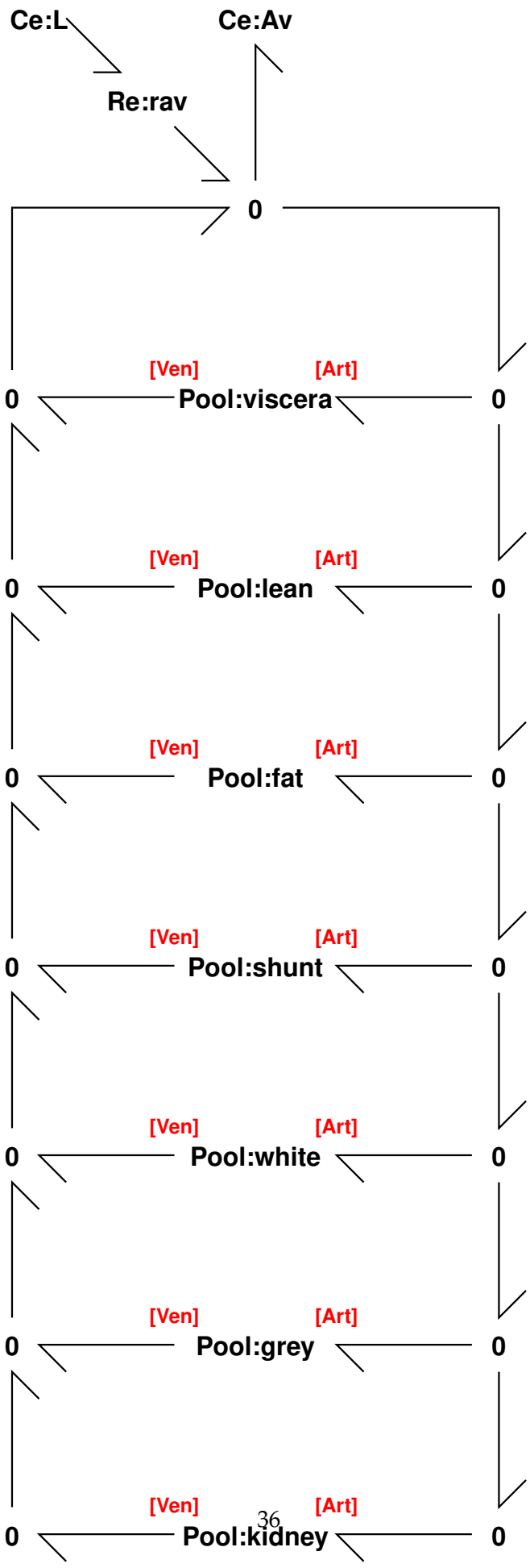
[44]:



[ ]:
```

```

[45]:
```
disp.SVG('PK_abg.svg')
```

[45]:


Ce:L
Ce:Av
Re:rav
0
[Ven]
[Art]
0
Pool:viscera
0
[Ven]
[Art]
0
Pool:lean
0
[Ven]
[Art]
0
Pool:fat
0
[Ven]
[Art]
0
Pool:shunt
0
[Ven]
[Art]
0
Pool:white
0
[Ven]
[Art]
0
Pool:grey
0
[Ven]
[Art]
0
Pool:kidney
0

[46]:
```
def Pool(N_lump=0, quiet=True):

    ## Create Pipe component
    ## Note: N_lump = 0 corresponds to orifice case.
    Pipe = mbg.chain(AB_abg.model(),inport = 'A',outport = 'B',
                     ␣
 ↪N=N_lump+1,rename_components=True,Comps=['R'],quiet=quiet)
    s = st.stoich(Pipe,UniDir=None, quiet=quiet)
    s['name'] = 'Pipe_abg'
    stbg.model(s)

    ## Set unidirectional Re components
    UniDir = []
    for i in range(N_lump+1):
        for comp in ['pin','pout']:
            ud = f'{comp}_r_{i}'
            UniDir.append(ud)

    # print(UniDir)
    ## Create stoichiometry
    sbg.model('Pool_abg.svg')
    import Pipe_abg
    imp.reload(Pipe_abg)
    import Pool_abg
    imp.reload(Pool_abg)
    sPool = st.stoich(Pool_abg.model(),UniDir=UniDir,quiet=quiet)


    if not quiet:
        disp.Latex(st.sprintrl(sPool,chemformula=True))

    ## Flatten
    stbg.model(sPool,filename='Pool_abg')

    return UniDir

def setPool(N_lump=0,Flow=f'{Q0}', quiet=True):
    ## Setup parameters
    parameter = {}
    Kappa = {}

    ## Assume K=KK is same for all species
    ## Note that in the simulation code K corresponds to KK/V_i
    KK = 1

    for i in range(N_lump+1):
        for comp in ['pin','pout']:
            kap = f'{comp}_r_{i}'
```

```
            Kappa[kap] = f'({Flow})/{KK}'
            #parameter[kap] = Q0
            if i>0:
                V_i = V_p/N_lump
                K0 = KK/V_i
                K0F = K0
                K0R = K0
                K = f'K_{comp}_IS{i}'
                if 'f' in comp:
                    parameter[K] = K0F
                else:
                    parameter[K] = K0R


    ## Setup initial states
    X0 = []
    for spec in sPool['species']:
    #     print(spec)
        if 'IS' in spec:
            K = f'K_{spec}'
            Xi = 1/parameter[K]
        else:
            Xi = 1
        X0.append(Xi)

    X0 = np.array(X0)


    return parameter,X0
```

[47]:
```
N_lump = 5
UniDir = Pool(N_lump=N_lump)
import Pool_abg
imp.reload(Pool_abg)
sPool = st.stoich(Pool_abg.model(),UniDir=UniDir,quiet=quiet)

disp.Latex(st.sprintrl(sPool,chemformula=True))
```

```
Creating subsystem: Pipe:pin
Creating subsystem: Pipe:pout
```

[47]:

$$\mathrm{Art} \xrightarrow{\mathrm{pin_{r0}}} \mathrm{pin_IS_1} \tag{59}$$

$$\mathrm{pin_IS_1} \xrightarrow{\mathrm{pin_{r1}}} \mathrm{pin_IS_2} \tag{60}$$

$$\mathrm{pin_IS_2} \xrightarrow{\mathrm{pin_{r2}}} \mathrm{pin_IS_3} \tag{61}$$

$$\mathrm{pin_IS_3} \xrightarrow{\mathrm{pin_{r3}}} \mathrm{pin_IS_4} \tag{62}$$

$$\mathrm{pin_IS_4} \xrightarrow{\mathrm{pin_{r4}}} \mathrm{pin_IS_5} \tag{63}$$

$$\mathrm{pin_IS_5} \xrightarrow{\mathrm{pin_{r5}}} \mathrm{P} \tag{64}$$

$$\mathrm{P} \xrightarrow{\mathrm{pout_{r0}}} \mathrm{pout_IS_1} \tag{65}$$

$$\mathrm{pout_IS_1} \xrightarrow{\mathrm{pout_{r1}}} \mathrm{pout_IS_2} \tag{66}$$

$$\mathrm{pout_IS_2} \xrightarrow{\mathrm{pout_{r2}}} \mathrm{pout_IS_3} \tag{67}$$

$$\mathrm{pout_IS_3} \xrightarrow{\mathrm{pout_{r3}}} \mathrm{pout_IS_4} \tag{68}$$

$$\mathrm{pout_IS_4} \xrightarrow{\mathrm{pout_{r4}}} \mathrm{pout_IS_5} \tag{69}$$

$$\mathrm{pout_IS_5} \xrightarrow{\mathrm{pout_{r5}}} \mathrm{Ven} \tag{70}$$

[48]:
```
import Pool_abg
imp.reload(Pool_abg)
sbg.model('PK_abg.svg')
import PK_abg
imp.reload(PK_abg)
pools = ['viscera','lean','fat','shunt','white','grey','kidney']
PK_UniDir = []
for pool in pools:
    for ud in UniDir:
        PKud = f'{pool}_{ud}'
        PK_UniDir.append(PKud)
# print(PK_UniDir)
sPK = st.stoich(PK_abg.model(),UniDir=PK_UniDir, quiet=quiet)
```

```
Creating subsystem: Pool:fat
Creating subsystem: Pool:grey
Creating subsystem: Pool:kidney
Creating subsystem: Pool:lean
Creating subsystem: Pool:shunt
Creating subsystem: Pool:viscera
Creating subsystem: Pool:white
```

[49]:
```
# disp.Latex(st.sprintrl(sPK,chemformula=True))
```

[50]:
```
# disp.Latex(st.sprintvl(sPK))
```

7.1 Parameters: Map78 Table I

[51]:
```
## Parameters: Volume - litre
Volume={}

Volume['viscera'] = 6.2
Volume['lean'] = 39.2
Volume['fat'] = 12.2
Volume['shunt'] = 1e-6
Volume['white'] = 0.0007
Volume['grey'] = 0.0007
Volume['kidney'] = 0.0007

Volume['arterial'] = 1.4
Volume['venous'] = 4.0
```

[52]:
```
# ## Parameters: Flow (OLD)
# FlowFraction={}
# FlowFraction['viscera'] = 0.63
# FlowFraction['lean'] = 0.131
# FlowFraction['fat'] = 0.04
# FlowFraction['shunt'] = 0.199
# FlowFraction['white'] = 0.000022
# FlowFraction['grey'] = 0.000086
# FlowFraction['kidney'] = 0.000432
```

[53]:
```
## Parameters: Flow (New)
FlowFraction={}
FlowFraction['viscera'] = 0.399
FlowFraction['lean'] = 0.364
FlowFraction['fat'] = 0.111
FlowFraction['shunt'] = 0.126
FlowFraction['white'] = 0.000014
FlowFraction['grey'] = 0.000055
FlowFraction['kidney'] = 0.000274
```

[54]:
```
## Parameters: Partition coefficient
PartitionCoefficient = {}
for pool in pools:
    if pool in ['fat']:
        PartitionCoefficient[pool] = 1.40
    else:
        PartitionCoefficient[pool] = 0.46
print(PartitionCoefficient)
```

```
{'viscera': 0.46, 'lean': 0.46, 'fat': 1.4, 'shunt': 0.46, 'white': 0.46,
'grey': 0.46, 'kidney': 0.46}
```

[55]:
```
Q = 6.48/60 # lit/sec Table II

Delay = {}
```

```
Delay_min = {}
for blood in ['arterial','venous']:
    Delay[blood] = Volume[blood]/Q
    Delay_min[blood] = Delay[blood]/60
    print(f'Delay ({blood}) = {Delay[blood]:.2f} sec = {Delay_min[blood]:0.
 ↪2f} min')


parameter = {}

## Lung diffusion parameters
parameter['kappa_rav'] = 0.5
parameter['K_L'] = 1
parameter['K_Av'] = 0.05

parameter['kappa_rav'] = 0.5
parameter['K_L'] = 1
parameter['K_Av'] = 0.05


## Pool parameters
for pool in pools:

    ## Pool Ce:P
    K = f'K_{pool}'
    par = 1/(Volume[pool]*PartitionCoefficient[pool])
#     frac = VolumeFraction[pool]
#     frac = 0.5 # FIXME
    parameter[f'{K}_P'] = par

    ## Pool pipes RA components
    for i in range(N_lump+1):
        for inout in ['in','out']:
            kappa = f'kappa_{pool}_p{inout}_r_{i}'
            parameter[kappa] = Q*FlowFraction[pool]

    ## Pool pipes C components
    if N_lump>0:
        for i in range(N_lump):
            for inout in ['in','out']:
                K = f'K_{pool}_p{inout}_IS{i+1}'
                if inout in ['in']:
                    V_i = FlowFraction[pool]*Volume['arterial']/N_lump
                else:
                    V_i = FlowFraction[pool]*Volume['venous']/N_lump
                parameter[K] = (1/V_i)


#print(parameter)
```

```
Delay (arterial) = 12.96 sec = 0.22 min
```

```
Delay (venous) = 37.04 sec = 0.62 min
```

7.2 Simulation

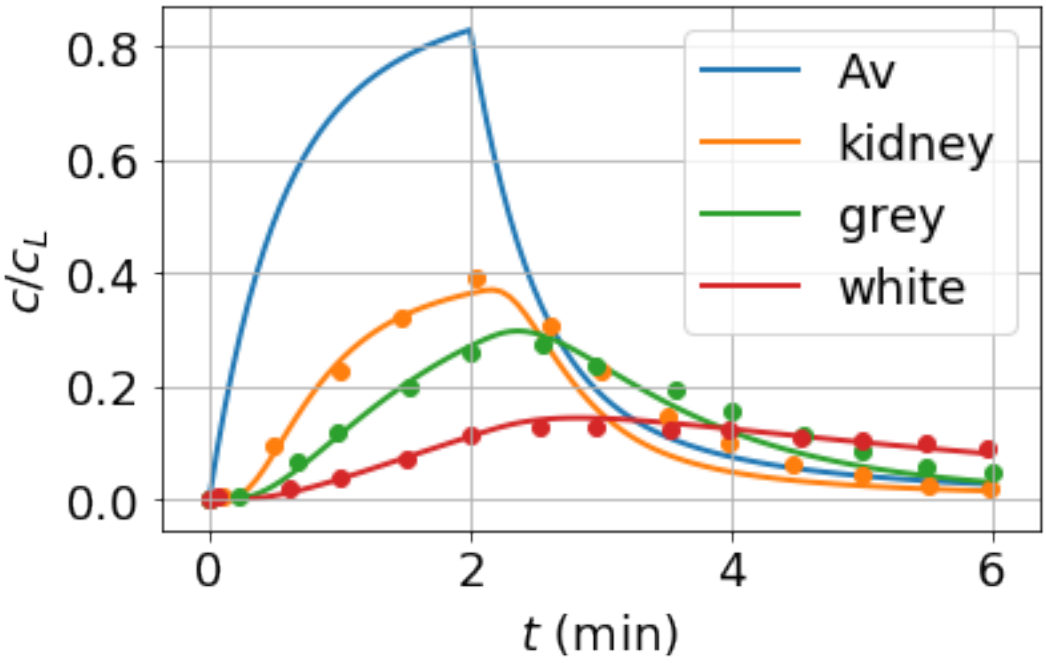
[56]:
```
## Simulation
t = np.linspace(0,360,1000)
t = np.linspace(0,360,100)
t_ss = np.linspace(0,10000,1000)
t_1 = 120

## Initial conditions
x_min = 1e-6
X0 = x_min*np.ones(sPK['n_X'])


## Chemostats
chemostats = ['L']
scPK = st.statify(sPK,chemostats=chemostats)
X_chemo = {'L':f'{x_min} + 1*(t<{t_1})'}
# print(X_chemo)

## Simulate - steady state
dat_ss = st.sim(sPK,sc=scPK,t=t_ss,X0=X0,parameter=parameter)
X_ss = dat_ss['X'][-1,:]
#print(X_ss)

## Simulate
dat = st.sim(sPK,sc=scPK,t=t,X0=X_ss,parameter=parameter,X_chemo=X_chemo)
```

[57]:
```
##Extract Mapleson data
def getCSV(name):
    x = [0]
    y = [0]
    filename = name+'.csv'
    if exists(filename):
        with open(filename) as csvfile:
            csvReader = csv.reader(csvfile, delimiter=',')
            for row in csvReader:
                x += [float(row[0])]
                y += [float(row[1])]

    return np.array(x),np.array(y)

# X = {}
# Y = {}
# x,y = getCSV('kidney')
# plt.plot(x,y)
```

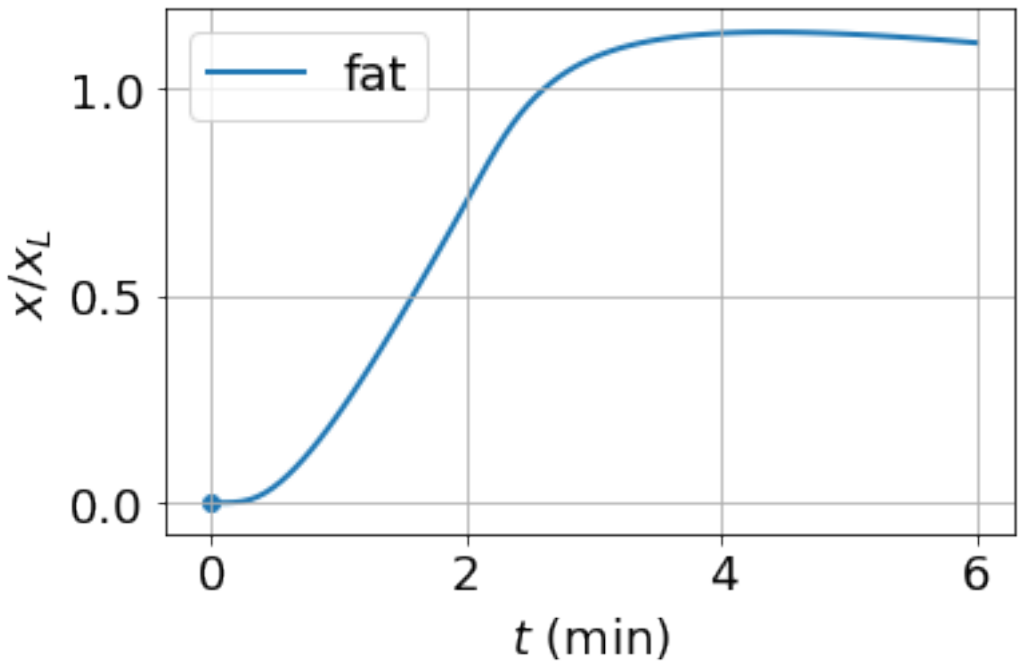
[58]:
```
## Function to plot pool data
def␣
 ↪plotPool(plot_pools=['Av','kidney','grey','white'],figname='sim',conc=True):
```

```
    ## Extract state
    species = sPK['species']
    X = dat['X']

    ## Font
    plt.rcParams.update({'font.size': 18})

    ## Compute concentration, and plot
    for pool in plot_pools:
        if pool in ['L','Av']:
            name = f'{pool}'
            par = f'K_{pool}'
            vol = 1/parameter[par]
        else:
            name = f'{pool}_P'
            vol = Volume[pool]
#         print(name)
        i_pool = species.index(name)

        if conc:
            y = X[:,i_pool]/vol
        else:
            y = X[:,i_pool]

        plt.plot(t/60,y,label=pool,lw=2,zorder=0)

    for pool in plot_pools:
        t_map,y_map = getCSV(pool)

        ## Normalisation factor (units of Tension) to make data
        ## commensurate with simulation data which is normalised concentration.
        T_L = 1000 #mmHg

        ## Plot as dots on top of simulation data - same colours
        plt.scatter(t_map,y_map/T_L,zorder=1)

    plt.legend()
    plt.grid()
    plt.xlabel('$t$ (min)')
    if conc:
        plt.ylabel('$c/c_L$')
    else:
        plt.ylabel('$x/x_L$')
    plt.tight_layout()
    Savefig(f'Figs/PK_{figname}.pdf')
    plt.show()
```

[59]:
```
## Plot data
plotPool()
```



[60]:
```
## Plot Fat amounts to show the long decay
plotPool(plot_pools = ['fat'],conc=False,figname='fat')
```



References

W. W. Mapleson. Circulation-time models of the uptake of inhaled anaesthetics and data for quantifying them. *British Journal of Anaesthesia*, 45:319, 1973.