Oscillation

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

April 16, 2024

Contents

1 Introduction

This Jupyter notebook (Oscillation.ipynb) contains the code used to generate the following examples for the paper "Analysis of Biochemical Oscillators Using Bond Graphs and Linear Control Theory" by Peter Gawthrop and Michael Pan:

- 3. Illustrative Example (system Toy)
- 4. The Sel'kov Oscillator (system Selkov)

The example - 5. The Repressilator is in the notebook Repressilator.ipynb

1.1 Select system to be analysed

[1]:
```
## Select system to be analysed
## NB Repressilator example is in a separate notebook:␣
 ↪Repressilator,Repressilator.ipynb
SystemName = 'Toy'
ToyVersion = 'basic'
# ToyVersion = 'varyAct'
# ToyVersion = 'varyR'

# SystemName = 'Toycc'
# SystemName = 'Toy1'

# SystemName = 'ToyTwo'
# SystemName = 'ToyTwo3'
# SystemName = 'ToyTwo4'

# SystemName = 'Goodwin'

# SystemName = 'Selkov'
# SelkovVersion = 'basic'
# SelkovVersion = 'varyATP'
# SelkovVersion = 'varyR'
# SystemName = 'Selkov1'
# SystemName = 'Selkov3'
```

[2]:
```
if SystemName in ['Toy']:
    if ToyVersion in ['basic']:
        K_Act = 1
        kappa_rf = 10
        SysName = 'Toy'
    elif ToyVersion in ['varyAct']:
        K_Act = 0.2
        kappa_rf = 10
        SysName = 'Toymod'
    elif ToyVersion in ['varyR']:
        K_Act = 1
        kappa_rf = 25
        SysName = 'ToymodR'
    else:
        print('ToyVersion',ToyVersion,'not known.')
        print('K_Act=', K_Act)
elif SystemName in ['Selkov']:
    if SelkovVersion in ['basic']:
        v_ATP = 0.6
        kappa_rf = 10
        SysName = 'Selkov'
    elif SelkovVersion in ['varyATP']:
        v_ATP = 0.7
        kappa_rf = 10
        SysName = 'Selkovmod'
    elif SelkovVersion in ['varyR']:
        v_ATP = 0.6
        kappa_rf = 8
        SysName = 'SelkovmodR'
    else:
        print('SelkovVersion', SelkovVersion,'not known.')
else:
    SysName = SystemName
    v_ATP = 0.6
    K_Act = 1
    kappa_rf = 10
print(SysName)
```

```
Toy
```

[3]:
```
## Decide whether to compute parametric variation results - takes a long time!
ParametricVariation = False

## Save data for comparative plots
SavingData = False
SaveData = SysName in␣
 ↪['Toy','Toy1','Toycc','ToyTwo','ToyTwo3','ToyTwo4','Toymod','ToymodR',
                    'Selkov','Selkovmod','SelkovmodR','Selkov1','Selkov3']
SaveData = SaveData and SavingData
if SaveData:
    SavedData = {}
print("Saving data =",SaveData)
```

```
Saving data = False
```

1.2 Import some python code

The bond graph analysis uses a number of Python modules:

[4]:
```
## For path etc: sys.path
import sys
sys.path.append("/home/peterg/WORK/Research/SystemsBiology/lib/python")
```

[5]:
```
## Some useful imports
import BondGraphTools as bgt
import numpy as np
import sympy as sym
import scipy.optimize as opt
import scipy.integrate as integrate
import matplotlib.pyplot as plt
from cycler import cycler
import IPython.display as disp
import copy

## Stoichiometric analysis
import stoich as st

## SVG bg representation conversion
import svgBondGraph as sbg

## Modularity
import modularBondGraph as mbg

## Control systems package
import control as con
con.config.defaults['xferfcn.display_format'] = 'zpk'
import slycot

## Stoichiometry to BG
import stoichBondGraph as stbg

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

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

# Allow output from within functions
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

## Saving data
import pickle

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

## Plot figure to folder Figs
Plotting = False
```

[6]:
```
def printSS(s,x_ss,parameter):
    print('\nSteady state')
    for i,spec in enumerate(s['species']):
        # print(i,spec,x_ss[i])
        val = x_ss[i]
        if not (val==1):
            print(f'x_{{{spec}}} &= {val:.2g}\\\\')
    print('\nParameters')
    for par in parameter:
        val = parameter[par]
        if not (val==1):
            print(f'{par} & {val:.2g}\\\\')
```

[7]:
```
def zero_crossings(a):
    """Zero crossings from  positive to negative
    """
    return np.where(np.diff(np.sign(a))>0)[0]
```

[8]:
```
def SetPlot(fontsize=14,linewidth=5,RL=False):
        ## Sizes
        plt.rcParams.update({'font.size': fontsize})
        plt.rcParams.update({'lines.linewidth': linewidth})
        plt.rcParams.update({'lines.markersize': 6*linewidth})

        ## set up colour cycling for plot
        if RL:
            ## Root locus colors
            default_cycler = (cycler(color=['grey','grey','r', 'g', 'b']))
        else:
            default_cycler = (cycler(color=['r', 'g', 'b']))
        plt.rc('axes', prop_cycle=default_cycler)

SetPlot()
```

[9]:
```
def SaveFig(SystemName,PlotName,fontsize=14,linewidth=5,RL=False):
    if Plotting:
        SetPlot(RL=RL)
        plotname = f'Figs/{SystemName}_{PlotName}.pdf'
        plt.savefig(plotname)
```

[10]:
```
# def SaveFig(SystemName,PlotName,Plotting=True,fontsize=14,linewidth=5):
#     if Plotting:
#         plt.rcParams.update({'font.size': fontsize})
#         plt.rcParams.update({'lines.linewidth': linewidth})
#         plt.rcParams.update({'lines.markersize': 6*linewidth})
#         plotname = f'Figs/{SystemName}_{PlotName}.pdf'
#         plt.savefig(plotname)
```

[11]:
```
def zpk(tf,display_format='zpk'):
    p = np.real(con.poles(tf))
    z = np.real(con.zeros(tf))
    return con.zpk(z,p,1)
```

[12]:
```
def balred(sys,n_red):
    if n_red<sys.nstates:
        return con.balred(sys,n_red)
    else:
        return sys
```

[13]:
```
def printTF(tf):
    return tf
```

[14]:
```
def Properties(s,sc):
    ## Pathways
    print('Paths')
    print(st.sprintp(sc))

    ## Conserved Moieties (Pools)
    print('Pools:')
    disp.Latex(st.sprintml(s))
```

[15]:
```
def SetChemostatsToy(N=3,Config='Closed',quiet=False):

    print(f'Setting feedback loop with configuration {Config}')
    chemostats = ['fb_Act','fb_E0']

    if Config in ['Closed']:
        chemostats = chemostats
    elif Config in ['Open']:
        chemostats.append('P')
    elif Config in ['Dynamic']:
        chemostats += ['E1']
    elif Config in ['SplitLoop']:
        chemostats += ['Pf','P']
    else:
        print(f'Config={Config} not recognised')

    for i in range(1,N+1):
        # for comp in ['A', 'Z', 'Zf']:
        for comp in ['A', 'Zf']:
            chemostats.append(f'decr{i}_{comp}')
    return chemostats
```

[16]:
```
def SetChemostatsGoodwin(N=3,Config='Closed',quiet=False):

    print(f'Setting feedback loop with configuration {Config}')
    chemostats = ['fb_Act','fb_E0']

    if Config in ['Closed']:
        chemostats = chemostats
    elif Config in ['Open']:
        chemostats.append('P')
    elif Config in ['Dynamic']:
        chemostats += ['E1']
    elif Config in ['SplitLoop']:
        chemostats += ['Pf','P']
    else:
        print(f'Config={Config} not recognised')

    for i in range(1,N+1):
        for comp in ['A', 'Z', 'Zf']:
            chemostats.append(f'decr{i}_{comp}')
    return chemostats
```

[17]:
```
def SetChemostatsSelkov(Config='Closed',quiet=False):

    print(f'Setting feedback loop with configuration {Config}')
    chemostats = []

    if Config in ['Closed']:
        chemostats = chemostats
    elif Config in ['Open']:
        chemostats.append('P')
    elif Config in ['SplitLoop']:
        chemostats += ['Pf','P']
    else:
        print(f'Config={Config} not recognised')

    for comp in ['ATP0', 'Z', 'Zf']:
        chemostats.append(f'selkov_{comp}')

    return chemostats
```

[18]:
```
# def SetChemostatsRepressilator(Config='Closed',quiet=False):

#     print(f'Setting feedback loop with configuration {Config}')
#     chemostats = ['A', 'G1_XM', 'G1_XP', 'G2_XM','G2_XP','G3_XM','G3_XP']
#     if Config in ['Closed']:
#         chemostats = chemostats
#     elif Config in ['Open']:
#         chemostats.append('P3')
#     elif Config in ['SplitLoop']:
#         chemostats += ['P3f','P3']
#     else:
#         print(f'Config={Config} not recognised')
#     return chemostats
```

[19]:
```
def SetChemostats(SystemName,Config='Closed',quiet=False):
    if SystemName in ['Toy','Toy1','Toycc']:
        chemostats = SetChemostatsToy(N=3,Config=Config)
    elif SystemName in ['ToyTwo','ToyTwo3','ToyTwo4']:
        chemostats = SetChemostatsToy(N=2,Config=Config)
    elif SystemName in ['Goodwin']:
        chemostats = SetChemostatsGoodwin(N=3,Config=Config)
    elif SystemName in ['Selkov','Selkov1','Selkov3']:
        chemostats = SetChemostatsSelkov(Config=Config)
    # elif SystemName in ['Repressilator']:
    #     chemostats = SetChemostatsRepressilator(Config=Config)
    else:
        print('System Name',SystemName,'is not known')

    return chemostats
```

[20]:
```
def stoichiometry(abg,chemostats=[]):
#     print('Comp s')
    s = st.stoich(abg,quiet=quiet)
#     print('comp sc')
    sc = st.statify(s,chemostats=chemostats)
    return s,sc
```

[21]:
```
def SetParameterToy(N=3,kappa_rf=1e1,K_Act=1,K_Zf=1e-6): ## K_Zf = 0.2,1e-6
    parameter = {}
    parameter['K_fb_Act'] = K_Act
    for i in range(1,N+1):
        # parameter[f'K_decr{i}_Z'] = 1 # 1e-6
        parameter[f'K_decr{i}_Zf'] = K_Zf
        parameter[f'kappa_decr{i}_r']  = 1e0
        parameter[f'kappa_decr{i}_rf'] = kappa_rf
        parameter[f'K_decr{i}_A'] = 1e2
        parameter[f'K_E{i}'] = 1

    return parameter
```

[22]:
```
def SetParameterGoodwin(N=3,kappa_rf=1e1,K_Act=1,K_Z=1e-6):
    parameter = {}
    parameter['K_fb_Act'] = K_Act
    for i in range(1,N+1):
        parameter[f'K_decr{i}_Z'] = parameter[f'K_decr{i}_Zf'] = K_Z
        if i>1:
            parameter[f'kappa_decr{i}_r']  = 1e0
        else:
            parameter[f'kappa_decr{i}_r1']  = 1e0
            parameter[f'kappa_decr{i}_r2']  = 1e6
            parameter[f'K_decr{i}_C'] = 1e-3

        parameter[f'kappa_decr{i}_rf'] = kappa_rf
        parameter[f'K_decr{i}_A'] = 1e2

    return parameter
```

[23]:
```
def SetParameterSelkov(subname='_selkov',v_ATP=0.0,kappa_rf=1e1):
    parameter = {}
#     parameter['kappa_rd'] = 1

    parameter[f'K{subname}_Z'] = parameter[f'K{subname}_Zf'] = 1e-10
    parameter[f'kappa{subname}_r0']  = 1e3
    parameter[f'kappa{subname}_r1']  = 1e3
    parameter[f'kappa{subname}_r2']  = 1e3
    parameter[f'kappa{subname}_rf'] = kappa_rf
    parameter[f'K{subname}_PFK'] = 1
    parameter[f'K{subname}_C'] = 1

#     Large = 1e2
    Large = 1e3
    parameter[f'K{subname}_ATP0'] = Large
    parameter[f'kappa{subname}_rs'] = v_ATP/Large


    return parameter
```

```
[24]: def SetParameter():
          if SystemName in ['Toy','Toy1','Toycc']:
              parameter = SetParameterToy(N=3,K_Act=K_Act,kappa_rf=kappa_rf)
          elif SystemName in ['ToyTwo','ToyTwo3','ToyTwo4']:
              parameter = SetParameterToy(N=2,K_Act=K_Act,kappa_rf=kappa_rf)
          elif SystemName in ['Goodwin']:
              parameter = SetParameterGoodwin()
          elif SystemName in ['Selkov','Selkov1','Selkov3']:
              parameter = SetParameterSelkov(v_ATP=v_ATP,kappa_rf=kappa_rf)
          elif SystemName in ['Repressilator']:
              parameter = SetParameterRepressilator()
          else:
              print('System Name',SystemName,'is not known')

          return parameter
```

```
[25]: def SetAll(SystemName,Config='Closed',quiet=False):
          if SystemName in ['Toy','Toy1','Toycc']:
              chemostats = SetChemostatsToy(N=3,Config=Config)
              parameter = SetParameterToy(N=3,K_Act=K_Act,kappa_rf=kappa_rf)
              InpVar = 'fb_Act'
              OutpVar = 'P'
              T = np.linspace(0,4,1000)
              T_long = np.linspace(0,10,5000)
              n_red = 2
          elif SystemName in ['ToyTwo','ToyTwo3','ToyTwo4']:
              chemostats = SetChemostatsToy(N=2,Config=Config)
              parameter = SetParameterToy(N=2,K_Act=K_Act,kappa_rf=kappa_rf)
              InpVar = 'fb_Act'
              OutpVar = 'P'
              T = np.linspace(0,4,1000)
              T_long = np.linspace(0,10,5000)
              n_red = 2
          elif SystemName in ['Goodwin']:
              chemostats = SetChemostatsGoodwin(N=3,Config=Config)
              parameter = SetParameterGoodwin(N=3)
              InpVar = 'fb_Act'
              OutpVar = 'P'
              T = np.linspace(0,5,1000)
              T_long = np.linspace(0,20,5000)
```

```
        n_red = 3


    elif SystemName in ['Selkov','Selkov1','Selkov3']:
        chemostats = SetChemostatsSelkov(Config=Config)
        parameter = SetParameterSelkov(v_ATP=v_ATP,kappa_rf=kappa_rf)
        InpVar = 'selkov_ATP0'
        OutpVar = 'P'
        T = np.linspace(0,5,1000)
        T_long = np.linspace(0,50,5000)
        n_red = 2 ## Order reduction
    # elif SystemName in ['Repressilator']:
    #     chemostats = SetChemostatsRepressilator(Config=Config)
    #     parameter = SetParameterRepressilator()
    #     InpVar = 'A'
    #     OutpVar = 'P3'
    #     T = np.linspace(0,5,100)
    #     T_long = np.linspace(0,50,5000)
    else:
        print('System Name',SystemName,'is not known')

    return chemostats,parameter,InpVar,OutpVar,T,T_long,n_red
```

[26]:
```
def extractSysflow(Sys,s,chemo,chemostats,reac):

    ## Index of reaction
    reaction = s['reaction']
    i_v = reaction.index(reac)

    ## Index of input
    i = chemostats.index(chemo)

    sys = con.ss(Sys.A,Sys.B[:,i],Sys.C[i_v,:],Sys.D[i_v,i])

    return sys

def extractSysdX(Sys,s,chemo,chemostats,outp,tol=None):

    ## Index of output
    species = s['species']
    i_v = species.index(outp)

    ## Index of input
    i = chemostats.index(chemo)

    sys = con.ss(Sys.A,Sys.B[:,i],Sys.C[i_v,:],Sys.D[i_v,i])

    return con.minreal(sys,tol=tol)
```

[27]:
```
def IntegrateTF(L0,crite=1e-4):
    num0 = L0.num[0][0]
    den0 = L0.den[0][0]
#     print(den0)
    ln = len(num0)
    if (abs(num0[ln-1])<crite):
        ## remove s factor in numerator
        num = num0[:ln-1]
        den = den0
    else:
        ## Integrator
        ld = len(den0)
        num = num0
        den = np.zeros(ld+1)
        den[:ld] = den0

    L = con.tf(num,den)
    return L
```

[28]:
```
def Lin(s,sc,parameter=None,x_ss=None,outvar='dX',Inp=['P','E'],␣
↪Outp=['P','E'],quiet=True):

    ## Linearise
    SYS = st.lin(s,sc,x_ss=x_ss,parameter=parameter,outvar='dX',quiet=quiet)

    # Extract individual transfer functions
    TF = {}
    Sys = {}
    for inp in Inp:
        for outp in Outp:
            if not quiet:
                print(inp,'-->',outp)
            sys = extractSysdX(SYS,s,inp,chemostats,outp)
            tf = con.tf(sys)
            Sys[f'{inp}_{outp}'] = sys
            TF[f'{inp}_{outp}'] = tf
            if not quiet:
                print(tf)

    return TF,Sys
```

[29]:
```
def step_response(sys,T=None):
    resp = con.step_response(sys,T=T)
    t = resp.t
    y = np.array(resp.y).flatten()
    plt.plot(t,y)
    return y
```

[30]:
```
def impulse_response(sys,T=None):
    resp = con.impulse_response(sys,T=T)
    t = resp.t
    y = np.array(resp.y).flatten()
    plt.plot(t,y)
    return y
```

```
[31]: def SteadyState(s,sc,parameter,x0,OutpVar='P',returnAll=False):

    t = np.linspace(0,1e4)
    ndat = st.sim(s,sc=sc,t=t,parameter=parameter,X0=x0,quiet=True)
    x_ss = ndat['X'][-1,:]

    ## Flow into P
    species = s['species']
    v_ss = ndat['dX'][-1,species.index(OutpVar)]

    if returnAll:
        SS = {}
        ## Save up all steady-state data
        for key in ndat:
            if not key in ['t']:
#                   print(key)
                SS[key] = ndat[key][-1,:]
        return x_ss,v_ss,SS
    else:
        return x_ss,v_ss

def func(x_P):

    x0 = np.ones(s['n_X'])
    x0[species.index(OutpVar)] = x_P
    x_ss,v_ss = SteadyState(s,sc,parameter,x0,OutpVar=OutpVar)
    return v_ss

def findSteadyState(s,sc,parameter,x0,OutpVar='P',returnAll=False):

    species = s['species']
    root = opt.fsolve(func,1)

    x_P_ss = root[0]

    x0[species.index(OutpVar)] = x_P_ss

    if returnAll:
        x_ss,v_ss,SS =␣
↪SteadyState(s,sc,parameter,x0,OutpVar=OutpVar,returnAll=returnAll)
        return x_ss,SS
    else:
        x_ss,v_ss =␣
↪SteadyState(s,sc,parameter,x0,OutpVar=OutpVar,returnAll=returnAll)
        return x_ss,x_P_ss
```

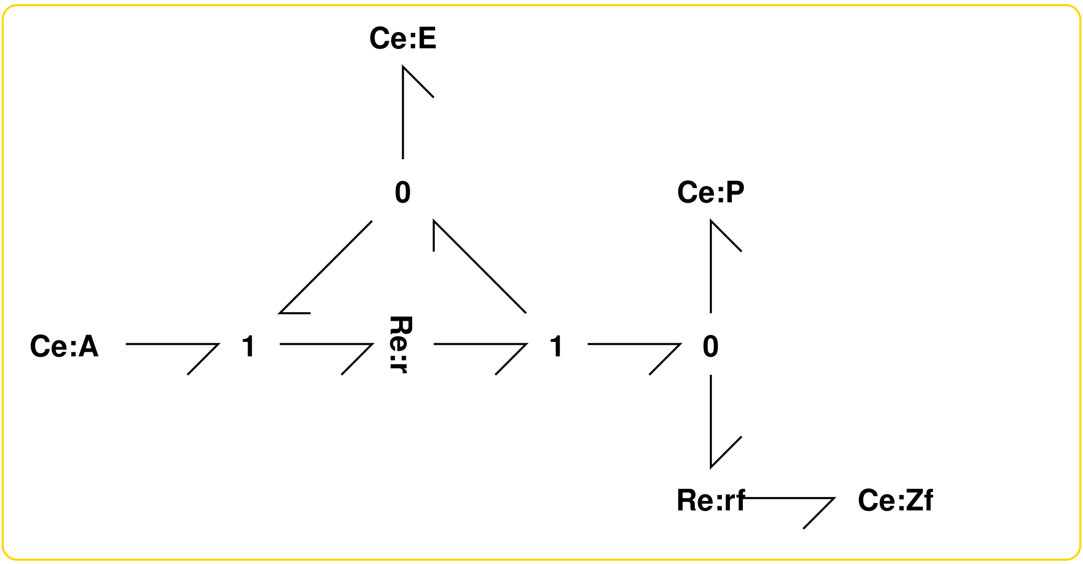
2 Submodels

```
- decr: basic enzyme reaction with degradation
- decrc: basic enzyme reaction with degradation and cooperativity
- dECR: enzyme reaction with intermediate complex and degradation
- dECRc: enzyme reaction withintermediate complex and degradation and cooperativity
- ActInh: activation/inhibition module
- Selkov: The Sel'kov model of glycolitic oscillations from Keener and Sneyd
```

[32]:
```
subTF = {}
V_ss = {}
for subname in ['decr_abg','decrc_abg','decrc3_abg','decrc4_abg',
                'dECR_abg','dECRc_abg','ActInh_abg','Selkov_abg']:

    svg = subname+'.svg'
    print('\n\nUsing',svg)
    disp.SVG(svg)

    sbg.model(svg,convertCe=True,convertR=True,quiet=quiet)
    exec(f'import {subname} as sys_abg')
    imp.reload(sys_abg)

    s = st.stoich(sys_abg.model(),quiet=quiet)
    species = s['species']
    print(species)
    disp.Latex(st.sprintrl(s,all=True))
    disp.Latex(st.sprintvl(s))
    print(st.sprintrl(s,all=True,chemformula=True))
    print(st.sprintvl(s))

    if subname not in ['ActInh_abg','Selkov_abg']:
        parameter={}
        chemostats = ['P','E','A','Zf']
        sc = st.statify(copy.deepcopy(s),chemostats=chemostats)
        parameter['K_A'] = 1
        # parameter['K_Z'] =
        parameter['K_Zf'] = 1e-6
        parameter['kappa_rf'] = 10

        if subname in ['dECR_abg','dECRc_abg']:
            parameter['kappa_r1'] =  parameter['kappa_r2'] = 2*1000
        else:
            parameter['kappa_r'] = 1000

        Sys = st.lin(s,sc,outvar='dX',parameter=parameter)
        for inp in ['E','P']:
            for outp in ['E','P']:
                sys = extractSysdX(Sys,s,inp,chemostats,outp)
                print(inp, ' - ', outp)
                con.tf(sys)
        chemostats = ['A','Zf','P']
        sc = st.statify(copy.deepcopy(s),chemostats=chemostats)

        K = np.linspace(0.1,2)
        e0 = 0.1
        V_ss[subname] = []
        x0 = np.ones(s['n_X'])
        for K_A in K:
            parameter['K_A'] = K_A
            if subname in ['dECR_abg','dECRc_abg']:
                x0[species.index('E')] = x0[species.index('C')] =e0/2
                parameter['kappa_r1']= 1e2
            else:
                x0[species.index('E')] = e0
            x_ss,v_ss = SteadyState(s,sc,parameter,x0,returnAll=False)
            V_ss[subname].append(v_ss)

    if subname in ['ActInh_abg']:
        chemostats = ['E','E0','Inh','Act']
        sc = st.statify(copy.deepcopy(s),chemostats=chemostats)
        parameter = {}
        Sys = st.lin(s,sc,outvar='dX',parameter=parameter)
        for inp in ['Inh','E']:
            for outp in ['Inh','E']:
                sys = extractSysdX(Sys,s,inp,chemostats,outp)
                print(inp, ' - ', outp)
                con.tf(sys)
```

```
Using decr_abg.svg
```

[32]:



```
{}
```

[32]:
```
<module 'decr_abg' from
'/home/peterg/WORK/Research/SystemsBiology/Notes/2024/Oscillation/decr_abg.
 ↪py'>

['A', 'E', 'P', 'Zf']
```
[32]:

$$A + E \Leftrightarrow E + P \tag{1}$$
$$P \Leftrightarrow Zf \tag{2}$$

[32]:

$$v_r = K_E \kappa_r x_E \left(K_A x_A - K_P x_P\right) \tag{3}$$
$$v_{rf} = \kappa_{rf} \left(K_P x_P - K_{Zf} x_{Zf}\right) \tag{4}$$

```
\begin{align}
\ch{A + E & <> [ r ] E + P }\\
\ch{P & <> [ rf ] Zf }
\end{align}

\begin{align}
v_{r} &= K_{E} \kappa_{r} x_{E} \left(K_{A} x_{A} - K_{P} x_{P}\right)\\
v_{rf} &= \kappa_{rf} \left(K_{P} x_{P} - K_{Zf} x_{Zf}\right)
\end{align}

Setting K_A to 1
Setting K_Zf to 1e-06
Setting kappa_r to 1000
Setting kappa_rf to 10
0 states have been removed from the model
E  -  E

/home/peterg/.local/lib/python3.8/site-
packages/scipy/signal/filter_design.py:1625: BadCoefficients: Badly conditioned
filter coefficients (numerator): the results may be meaningless
  warnings.warn("Badly conditioned filter coefficients (numerator): the "
/home/peterg/.local/lib/python3.8/site-
packages/scipy/signal/filter_design.py:1072: RuntimeWarning: invalid value
encountered in divide
  b /= b[0]
```
[32]:

$$\frac{0}{1}$$

```
0 states have been removed from the model
E  -  P
```
[32]:

$$\frac{0}{1}$$

```
0 states have been removed from the model
P  -  E
```

[32]:

$$\frac{0}{1}$$

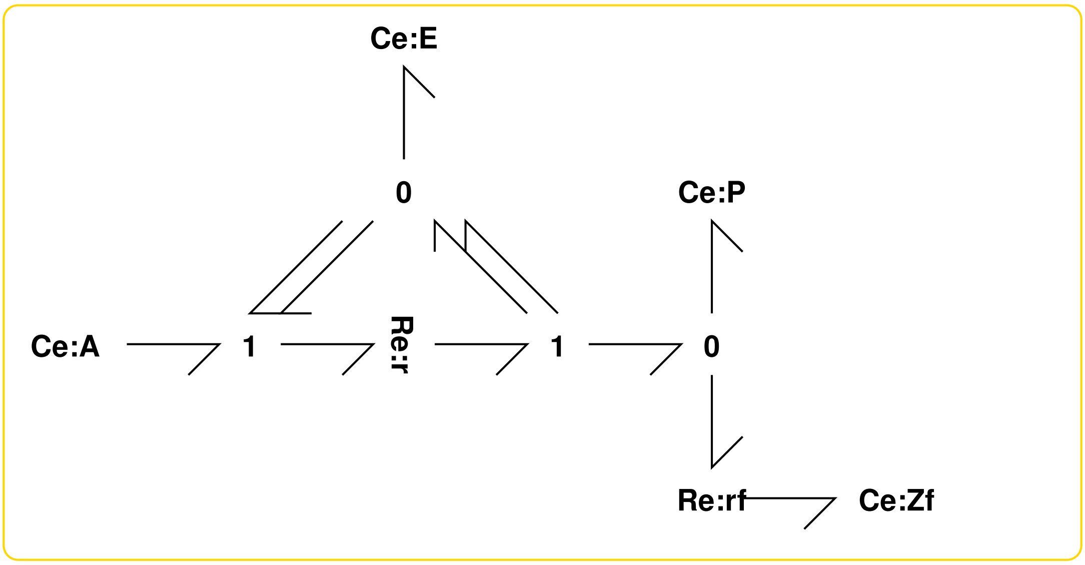
```
0 states have been removed from the model
P  -  P
```

[32]:

$$\frac{-1010}{1}$$

```
Using decrc_abg.svg
```

[32]:



```
{}
```

[32]:
```
<module 'decrc_abg' from
'/home/peterg/WORK/Research/SystemsBiology/Notes/2024/Oscillation/decrc_abg.
↪py'>
```

```
['A', 'E', 'P', 'Zf']
```

[32]:

$$A + 2E \Leftrightarrow 2E + P \tag{5}$$
$$P \Leftrightarrow Zf \tag{6}$$

[32]:

$$v_r = K_E^2 \kappa_r x_E^2 \left(K_A x_A - K_P x_P\right) \tag{7}$$
$$v_{rf} = \kappa_{rf} \left(K_P x_P - K_{Zf} x_{Zf}\right) \tag{8}$$

```
\begin{align}
\ch{A + 2 E & <> [ r ] 2 E + P }\\
\ch{P & <> [ rf ] Zf }
```

```
\end{align}

\begin{align}
v_{r} &= K_{E}^{2} \kappa_{r} x_{E}^{2} \left(K_{A} x_{A} - K_{P} x_{P}\right)\\
v_{rf} &= \kappa_{rf} \left(K_{P} x_{P} - K_{Zf} x_{Zf}\right)
\end{align}

Setting K_A to 1
Setting K_Zf to 1e-06
Setting kappa_r to 1000
Setting kappa_rf to 10
0 states have been removed from the model
E  -  E
```

[32]:

$$\frac{0}{1}$$

```
0 states have been removed from the model
E  -  P
```

[32]:

$$\frac{0}{1}$$

```
0 states have been removed from the model
P  -  E
```

[32]:

$$\frac{0}{1}$$

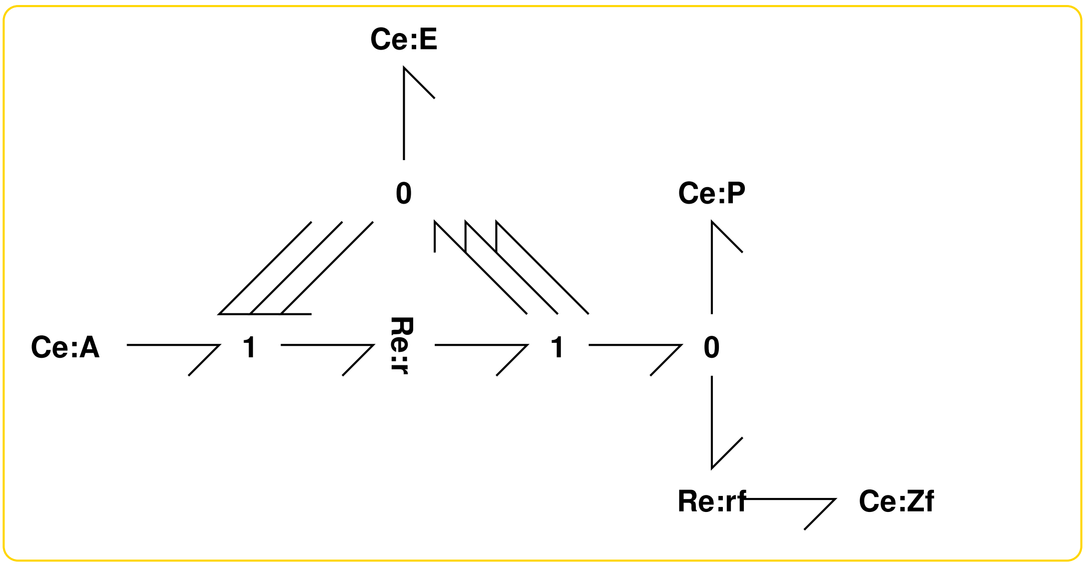
```
0 states have been removed from the model
P  -  P
```

[32]:

$$\frac{-1010}{1}$$

```
Using decrc3_abg.svg
```

[32]:



```
{}
```

[32]:
```
<module 'decrc3_abg' from '/home/peterg/WORK/Research/SystemsBiology/Notes/
  ↪2024/
Oscillation/decrc3_abg.py'>

['A', 'E', 'P', 'Zf']
```

[32]:

$$A + 3E \Leftrightarrow 3E + P \tag{9}$$
$$P \Leftrightarrow Zf \tag{10}$$

[32]:

$$v_r = K_E^3 \kappa_r x_E^3 \left(K_A x_A - K_P x_P\right) \tag{11}$$
$$v_{rf} = \kappa_{rf} \left(K_P x_P - K_{Zf} x_{Zf}\right) \tag{12}$$

```
\begin{align}
\ch{A + 3 E & <> [ r ] 3 E + P }\\
\ch{P & <> [ rf ] Zf }
\end{align}

\begin{align}
v_{r} &= K_{E}^{3} \kappa_{r} x_{E}^{3} \left(K_{A} x_{A} - K_{P} x_{P}\right)\\
v_{rf} &= \kappa_{rf} \left(K_{P} x_{P} - K_{Zf} x_{Zf}\right)
\end{align}

Setting K_A to 1
Setting K_Zf to 1e-06
Setting kappa_r to 1000
Setting kappa_rf to 10
0 states have been removed from the model
E  -  E
```

[32]:

$$\frac{0}{1}$$

```
0 states have been removed from the model
E  -  P
```

[32]:

$$\frac{0}{1}$$

```
0 states have been removed from the model
P  -  E
```

[32]:

$$\frac{0}{1}$$

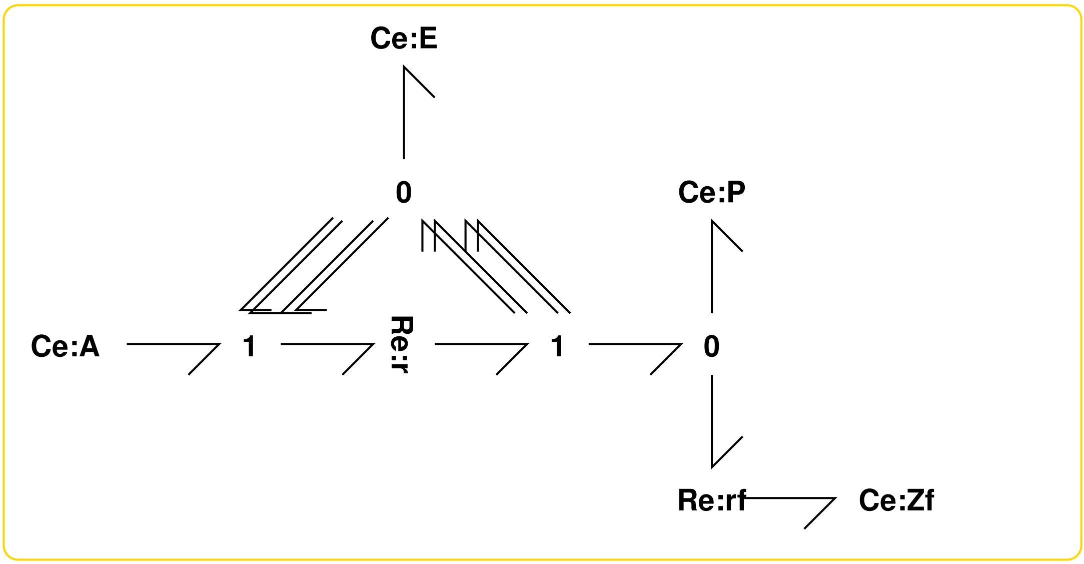
```
0 states have been removed from the model
P  -  P
```

[32]:

$$\frac{-1010}{1}$$

```
Using decrc4_abg.svg
```

[32]:



```
{}
```

[32]:
```
<module 'decrc4_abg' from '/home/peterg/WORK/Research/SystemsBiology/Notes/
↪2024/
Oscillation/decrc4_abg.py'>
```

```
['A', 'E', 'P', 'Zf']
```

[32]:

$$A + 4E \Leftrightarrow 4E + P \tag{13}$$
$$P \Leftrightarrow Zf \tag{14}$$

[32]:

$$v_r = K_E^4 \kappa_r x_E^4 \left(K_A x_A - K_P x_P\right) \tag{15}$$
$$v_{rf} = \kappa_{rf} \left(K_P x_P - K_{Zf} x_{Zf}\right) \tag{16}$$

```
\begin{align}
\ch{A + 4 E & <> [ r ] 4 E + P }\\
\ch{P & <> [ rf ] Zf }
\end{align}

\begin{align}
v_{r} &= K_{E}^{4} \kappa_{r} x_{E}^{4} \left(K_{A} x_{A} - K_{P} x_{P}\right)\\
v_{rf} &= \kappa_{rf} \left(K_{P} x_{P} - K_{Zf} x_{Zf}\right)
\end{align}

Setting K_A to 1
Setting K_Zf to 1e-06
Setting kappa_r to 1000
Setting kappa_rf to 10
0 states have been removed from the model
E  -  E
```

[32]:

$$\frac{0}{1}$$

```
0 states have been removed from the model
E  -  P
```

[32]:

$$\frac{0}{1}$$

```
0 states have been removed from the model
P  -  E
```

[32]:

$$\frac{0}{1}$$

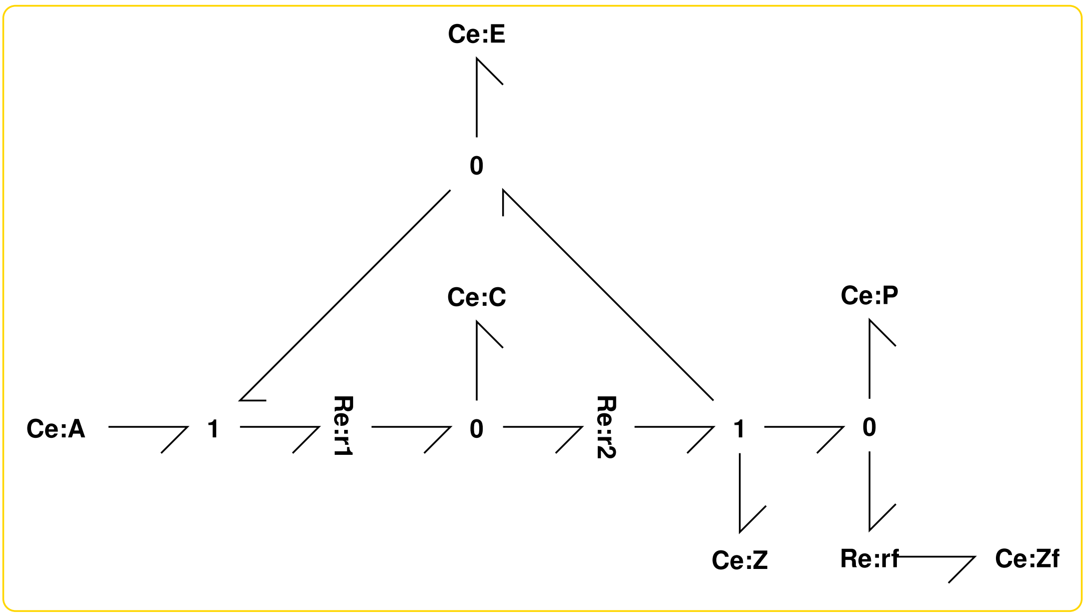
```
0 states have been removed from the model
P  -  P
```

[32]:

$$\frac{-1010}{1}$$

```
Using dECR_abg.svg
```

[32]:



```
{}
```

[32]:
```
<module 'dECR_abg' from
'/home/peterg/WORK/Research/SystemsBiology/Notes/2024/Oscillation/dECR_abg.
↪py'>

['A', 'C', 'E', 'P', 'Z', 'Zf']
```

[32]:

$$A + E \Leftrightarrow C \tag{17}$$
$$C \Leftrightarrow E + P + Z \tag{18}$$
$$P \Leftrightarrow Zf \tag{19}$$

[32]:

$$v_{r1} = \kappa_{r1} \left(K_A K_E x_A x_E - K_C x_C\right) \tag{20}$$
$$v_{r2} = \kappa_{r2} \left(K_C x_C - K_E K_P K_Z x_E x_P x_Z\right) \tag{21}$$
$$v_{rf} = \kappa_{rf} \left(K_P x_P - K_{Zf} x_{Zf}\right) \tag{22}$$

```
\begin{align}
\ch{A + E & <> [ r1 ] C }\\
\ch{C & <> [ r2 ] E + P + Z }\\
\ch{P & <> [ rf ] Zf }
\end{align}

\begin{align}
v_{r1} &= \kappa_{r1} \left(K_{A} K_{E} x_{A} x_{E} - K_{C} x_{C}\right)\\
v_{r2} &= \kappa_{r2} \left(K_{C} x_{C} - K_{E} K_{P} K_{Z} x_{E} x_{P}
x_{Z}\right)\\
v_{rf} &= \kappa_{rf} \left(K_{P} x_{P} - K_{Zf} x_{Zf}\right)
\end{align}
```

```
Setting K_A to 1
Setting K_Zf to 1e-06
Setting kappa_r1 to 2000
Setting kappa_r2 to 2000
Setting kappa_rf to 10
0 states have been removed from the model
E  -  E
```

[32]:

$$\frac{-4000(s - 1.431 \times 10^{-12})(s + 1000)}{(s + 763.9)(s + 5236)}$$

```
0 states have been removed from the model
E  -  P
```

[32]:

$$\frac{-2000(s - 4.367 \times 10^{-5})(s + 4.367 \times 10^{-5})}{(s + 763.9)(s + 5236)}$$

```
0 states have been removed from the model
P  -  E
```

[32]:

$$\frac{-2000(s - (4.657 \times 10^{-13} + 3.088 \times 10^{-5}j))(s - (4.657 \times 10^{-13} - 3.088 \times 10^{-5}j))}{(s + 763.9)(s + 5236)}$$

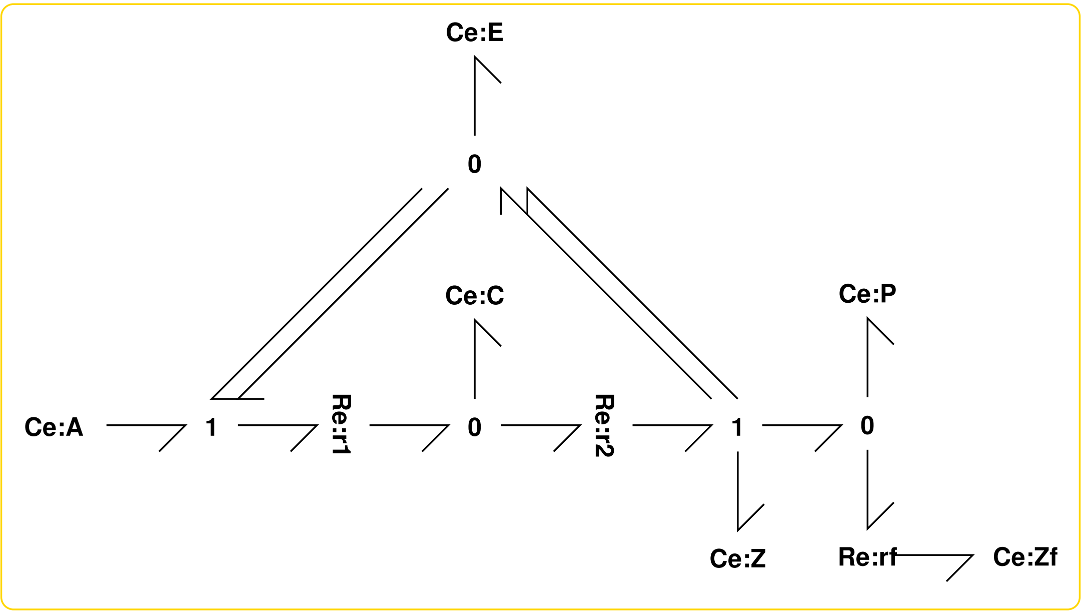
```
0 states have been removed from the model
P  -  P
```

[32]:

$$\frac{-2010(s + 9.901)(s + 2010)}{(s + 763.9)(s + 5236)}$$

```
Using dECRc_abg.svg
```

[32]:



```
{}
```

[32]:
```
<module 'dECRc_abg' from
'/home/peterg/WORK/Research/SystemsBiology/Notes/2024/Oscillation/dECRc_abg.
↪py'>

['A', 'C', 'E', 'P', 'Z', 'Zf']
```

[32]:

$$A + 2E \Leftrightarrow C \tag{23}$$
$$C \Leftrightarrow 2E + P + Z \tag{24}$$
$$P \Leftrightarrow Zf \tag{25}$$

[32]:

$$v_{r1} = \kappa_{r1} \left(K_A K_E^2 x_A x_E^2 - K_C x_C\right) \tag{26}$$
$$v_{r2} = \kappa_{r2} \left(K_C x_C - K_E^2 K_P K_Z x_E^2 x_P x_Z\right) \tag{27}$$
$$v_{rf} = \kappa_{rf} \left(K_P x_P - K_{Zf} x_{Zf}\right) \tag{28}$$

```
\begin{align}
\ch{A + 2 E & <> [ r1 ] C }\\
\ch{C & <> [ r2 ] 2 E + P + Z }\\
\ch{P & <> [ rf ] Zf }
\end{align}

\begin{align}
v_{r1} &= \kappa_{r1} \left(K_{A} K_{E}^{2} x_{A} x_{E}^{2} - K_{C}
x_{C}\right)\\
v_{r2} &= \kappa_{r2} \left(K_{C} x_{C} - K_{E}^{2} K_{P} K_{Z} x_{E}^{2} x_{P}
x_{Z}\right)\\
v_{rf} &= \kappa_{rf} \left(K_{P} x_{P} - K_{Zf} x_{Zf}\right)
\end{align}

Setting K_A to 1
Setting K_Zf to 1e-06
Setting kappa_r1 to 2000
Setting kappa_r2 to 2000
Setting kappa_rf to 10
0 states have been removed from the model
E  -  E
```

[32]:

$$\frac{-1.6\times 10^{4}(s-1.431\times 10^{-12})(s+1000)}{(s+763.9)(s+5236)}$$

```
0 states have been removed from the model
E  -  P
```

[32]:

$$\frac{-4000(s-4.367\times 10^{-5})(s+4.367\times 10^{-5})}{(s+763.9)(s+5236)}$$

```
0 states have been removed from the model
P  -  E
```

[32]:

$$\frac{-4000(s-(4.657\times10^{-13}+3.088\times10^{-5}j))(s-(4.657\times10^{-13}-3.088\times10^{-5}j))}{(s+763.9)(s+5236)}$$

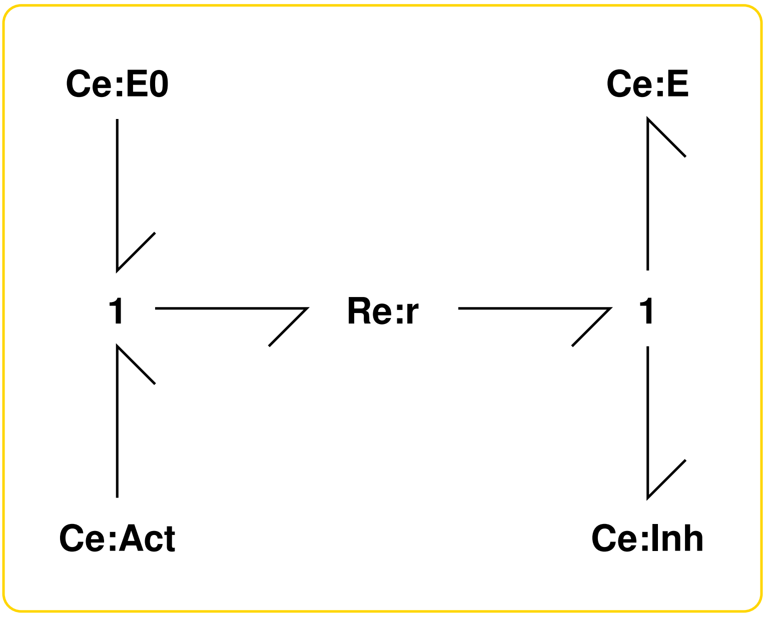
```
0 states have been removed from the model
P  -  P
```

[32]:

$$\frac{-2010(s+9.901)(s+2010)}{(s+763.9)(s+5236)}$$

```
Using ActInh_abg.svg
```

[32]:



```
{}
```

[32]:
```
<module 'ActInh_abg' from '/home/peterg/WORK/Research/SystemsBiology/Notes/
↪2024/
Oscillation/ActInh_abg.py'>
```

```
['Act', 'E', 'E0', 'Inh']
```

[32]:

$$Act + E0 \Leftrightarrow E + Inh \tag{29}$$

[32]:

$$v_r = \kappa_r \left(K_{Act} K_{E0} x_{Act} x_{E0} - K_E K_{Inh} x_E x_{Inh} \right) \tag{30}$$

```
\begin{align}
\ch{Act + E0 & <> [ r ] E + Inh }
```

```
\end{align}

\begin{align}
v_{r} &= \kappa_{r} \left(K_{Act} K_{E0} x_{Act} x_{E0} - K_{E} K_{Inh} x_{E}
x_{Inh}\right)
\end{align}

0 states have been removed from the model
Inh  -  Inh
```

[32]:

$$\frac{-1}{1}$$

```
0 states have been removed from the model
Inh  -  E
```

[32]:

$$\frac{-1}{1}$$

```
0 states have been removed from the model
E  -  Inh
```

[32]:

$$\frac{-1}{1}$$

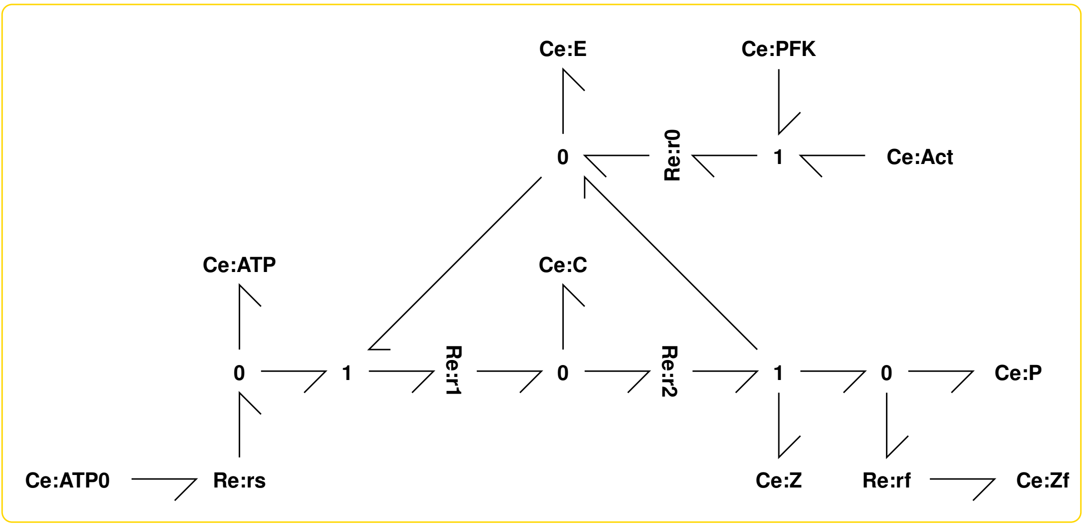
```
0 states have been removed from the model
E  -  E
```

[32]:

$$\frac{-1}{1}$$

```
Using Selkov_abg.svg
```

[32]:



```
{}
```

```
[32]: <module 'Selkov_abg' from '/home/peterg/WORK/Research/SystemsBiology/Notes/
       ↪2024/
      Oscillation/Selkov_abg.py'>

      ['ATP', 'ATP0', 'Act', 'C', 'E', 'P', 'PFK', 'Z', 'Zf']
[32]:
```

$$Act + PFK \Leftrightarrow E \tag{31}$$
$$ATP + E \Leftrightarrow C \tag{32}$$
$$C \Leftrightarrow E + P + Z \tag{33}$$
$$P \Leftrightarrow Zf \tag{34}$$
$$ATP0 \Leftrightarrow ATP \tag{35}$$

```
[32]:
```

$$v_{r0} = \kappa_{r0}\left(K_{Act}K_{PFK}x_{Act}x_{PFK} - K_{E}x_{E}\right) \tag{36}$$
$$v_{r1} = \kappa_{r1}\left(K_{ATP}K_{E}x_{ATP}x_{E} - K_{C}x_{C}\right) \tag{37}$$
$$v_{r2} = \kappa_{r2}\left(K_{C}x_{C} - K_{E}K_{P}K_{Z}x_{E}x_{P}x_{Z}\right) \tag{38}$$
$$v_{rf} = \kappa_{rf}\left(K_{P}x_{P} - K_{Zf}x_{Zf}\right) \tag{39}$$
$$v_{rs} = \kappa_{rs}\left(-K_{ATP}x_{ATP} + K_{ATP0}x_{ATP0}\right) \tag{40}$$

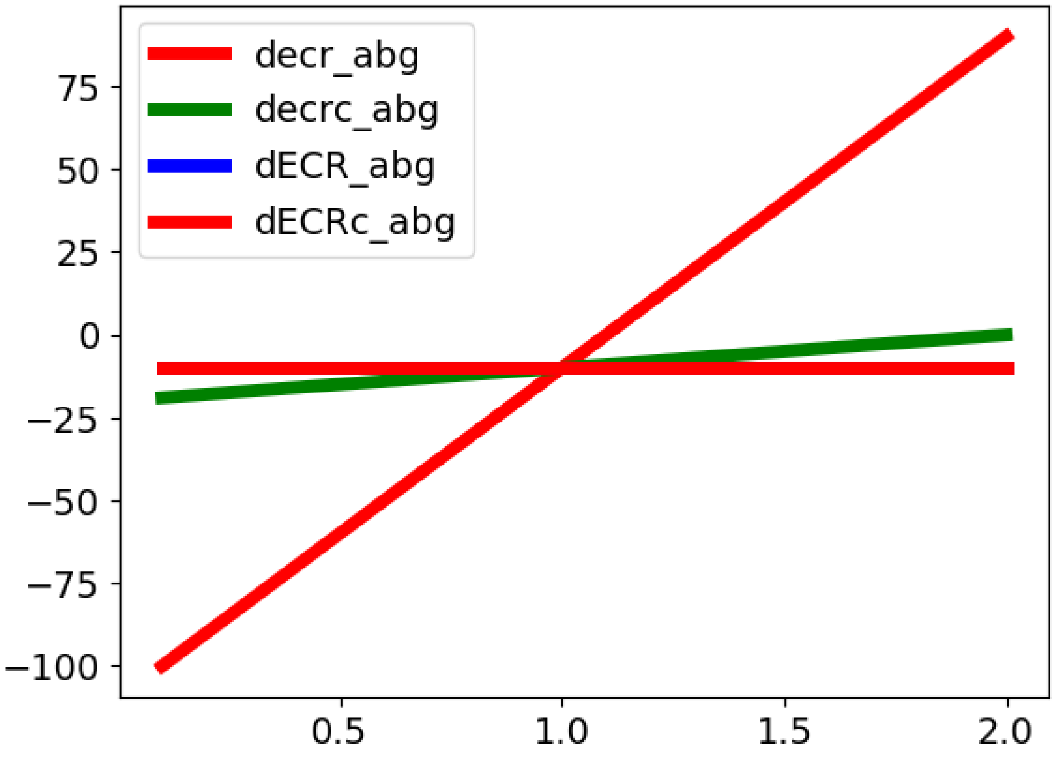
```
\begin{align}
\ch{Act + PFK & <> [ r0 ] E }\\
\ch{ATP + E & <> [ r1 ] C }\\
\ch{C & <> [ r2 ] E + P + Z }\\
\ch{P & <> [ rf ] Zf }\\
\ch{ATP0 & <> [ rs ] ATP }
\end{align}

\begin{align}
v_{r0} &= \kappa_{r0} \left(K_{Act} K_{PFK} x_{Act} x_{PFK} - K_{E}
x_{E}\right)\\
v_{r1} &= \kappa_{r1} \left(K_{ATP} K_{E} x_{ATP} x_{E} - K_{C} x_{C}\right)\\
v_{r2} &= \kappa_{r2} \left(K_{C} x_{C} - K_{E} K_{P} K_{Z} x_{E} x_{P}
x_{Z}\right)\\
v_{rf} &= \kappa_{rf} \left(K_{P} x_{P} - K_{Zf} x_{Zf}\right)\\
v_{rs} &= \kappa_{rs} \left(- K_{ATP} x_{ATP} + K_{ATP0} x_{ATP0}\right)
\end{align}
```

```
[33]: for subname in  ['decr_abg', 'decrc_abg', 'dECR_abg','dECRc_abg']:
          plt.plot(K,V_ss[subname],label=subname)
      plt.legend()
```

```
[33]: [<matplotlib.lines.Line2D at 0x7f6d656e7e80>]

[33]: [<matplotlib.lines.Line2D at 0x7f6d656fd220>]

[33]: [<matplotlib.lines.Line2D at 0x7f6d656fd460>]
```

```
[33]: [<matplotlib.lines.Line2D at 0x7f6d65710a30>]
```

```
[33]: <matplotlib.legend.Legend at 0x7f6dcc246850>
```

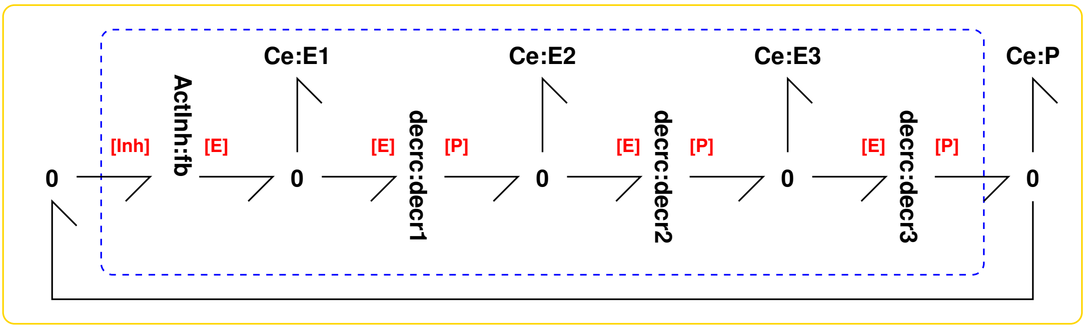


3 Feedback system

```
[34]: name = SystemName+'FB_abg'
      svg = name+'.svg'
      print('Using',svg)
      sbg.model(svg,convertCe=True,convertR=True,quiet=quiet)
      disp.SVG(svg)

      exec(f'import {name} as sys_abg')
      imp.reload(sys_abg)
```

```
Using ToyFB_abg.svg
Creating subsystem: ActInh:fb
Creating subsystem: decrc:decr1
Creating subsystem: decrc:decr2
Creating subsystem: decrc:decr3
{}
```

```
[34]:
```



[34]:
```
<module 'ToyFB_abg' from
'/home/peterg/WORK/Research/SystemsBiology/Notes/2024/Oscillation/ToyFB_abg.
↪py'>
```

[35]:
```
## Set up system
# v_ATP=0.6
chemostats,parameter,InpVar,OutpVar,T,T_long,n_red =␣
↪SetAll(SystemName,Config='Open',quiet=False)
chemostats_open = chemostats
# print(parameter)
```

```
Setting feedback loop with configuration Open
```

3.1 Stoichiometry

[36]:
```
s,sc = stoichiometry(sys_abg.model(),chemostats=chemostats)
species = s['species']
species_open = copy.copy(species)
reaction = s['reaction']
print(species)
print()
print(reaction)
print()
print(parameter)
```

```
['fb_Act', 'fb_E0', 'E1', 'E2', 'E3', 'P', 'decr1_A', 'decr1_Zf', 'decr2_A',
'decr2_Zf', 'decr3_A', 'decr3_Zf']

['fb_r', 'decr1_r', 'decr1_rf', 'decr2_r', 'decr2_rf', 'decr3_r', 'decr3_rf']

{'K_fb_Act': 1, 'K_decr1_Zf': 1e-06, 'kappa_decr1_r': 1.0, 'kappa_decr1_rf': 10,
'K_decr1_A': 100.0, 'K_E1': 1, 'K_decr2_Zf': 1e-06, 'kappa_decr2_r': 1.0,
'kappa_decr2_rf': 10, 'K_decr2_A': 100.0, 'K_E2': 1, 'K_decr3_Zf': 1e-06,
'kappa_decr3_r': 1.0, 'kappa_decr3_rf': 10, 'K_decr3_A': 100.0, 'K_E3': 1}
```

[37]:
```
## Reactions and properties
disp.Latex(st.sprintrl(s,all=True,chemformula=False))
# Properties(s,sc)
```

```
print('Pools:')
disp.Latex(st.sprintml(s))
print('Paths')
print(st.sprintp(sc))
```

[37]:

$$fb_Act + fb_E0 \Leftrightarrow E1 + P \tag{41}$$
$$2E1 + decr1_A \Leftrightarrow 2E1 + E2 \tag{42}$$
$$E2 \Leftrightarrow decr1_Zf \tag{43}$$
$$2E2 + decr2_A \Leftrightarrow 2E2 + E3 \tag{44}$$
$$E3 \Leftrightarrow decr2_Zf \tag{45}$$
$$2E3 + decr3_A \Leftrightarrow 2E3 + P \tag{46}$$
$$P \Leftrightarrow decr3_Zf \tag{47}$$

```
Pools:
```

[37]:

$$fb_Act \Leftrightarrow fb_E0 \tag{48}$$
$$\Leftrightarrow fb_Act + E1 \tag{49}$$
$$\Leftrightarrow E2 + decr1_A + decr1_Zf \tag{50}$$
$$\Leftrightarrow E3 + decr2_A + decr2_Zf \tag{51}$$
$$\Leftrightarrow fb_Act + P + decr3_A + decr3_Zf \tag{52}$$

```
Paths
4 pathways
0:  + decr1_r + decr1_rf
1:  + decr2_r + decr2_rf
2:  + decr3_r
3:  + decr3_rf
```

4 Open-loop analysis

[38]:

```
def OpenLoop(OutpVar='P',InpVar='fb_Act'):
    ## Steady-state analysis

    x0 = np.ones(s['n_X'])
    x_ss,SS =␣
 ↪findSteadyState(s,sc,parameter,x0,OutpVar=OutpVar,returnAll=True)
    # printSS(s,x_ss,parameter)

    ## Linearise
    TF,Sys_OL =␣
 ↪Lin(s,sc,parameter=parameter,x_ss=x_ss,outvar='dX',Inp=[InpVar,OutpVar],␣
 ↪Outp=[OutpVar])

    ## Loop gain
    Lname = OutpVar+'_'+OutpVar
    L0 = -TF[Lname]
    L = IntegrateTF(L0)
    L0_sys = -Sys_OL[Lname]

    ## Forward gain
    F = TF[InpVar+'_'+OutpVar]

    return L,L0,L0_sys,F,x_ss,SS


L,L0,L0_sys,F,x_ss,SS = OpenLoop(OutpVar=OutpVar,InpVar=InpVar)
x_ss_open = x_ss
# print(x_ss)
```

```
0 states have been removed from the model
0 states have been removed from the model
```

[ ]:

```
```

[39]:

```
print('L0:',L0)
print(f'Gain: {con.dcgain(L0):.2f}')

print('L:',L)
print(f'Gain: {con.dcgain(L):.2f}')
```

```
L0:
10.8 (s + (0.4024-14.08j)) (s + (0.4024+14.08j)) (s + 25.15)
------------------------------------------------------------
            (s + 5.936) (s + 10.03) (s + 10.08)

Gain: 89.77
L:
10.8 (s + (0.4024-14.08j)) (s + (0.4024+14.08j)) (s + 25.15)
------------------------------------------------------------
          (s) (s + 5.936) (s + 10.03) (s + 10.08)

Gain: inf
```

[40]:

```
# print(L0)
sysL0 = con.ss(L0)
sysL0_r = balred(sysL0,n_red)
L0_r = con.tf(sysL0_r)

print('L0_r:',L0_r)
print(f'Gain: {con.dcgain(L0_r):.2f}')
print(f'Poles: {con.poles(L0_r)}')
```

```
L0_r:
10.8 (s + (1.934-14.22j)) (s + (1.934+14.22j))
----------------------------------------------
  (s + (4.006-3.068j)) (s + (4.006+3.068j))
```

```
Gain: 87.35
Poles: [-4.00558944+3.06802235j -4.00558944-3.06802235j]
```

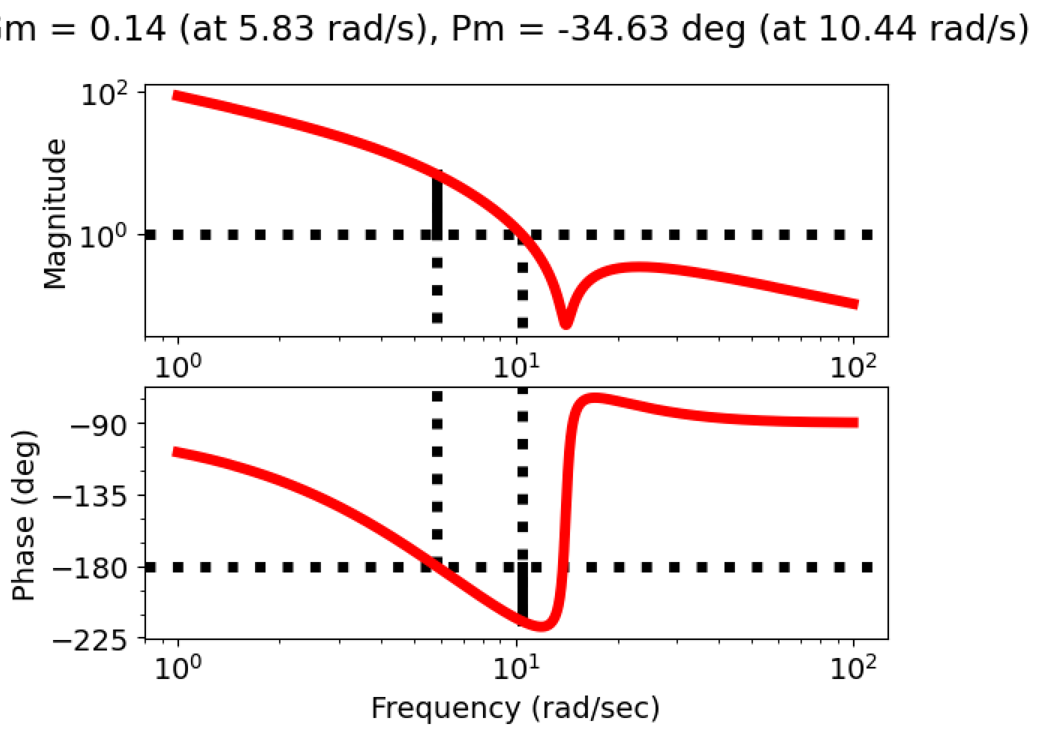
[41]:
```
print('F:',F)
print(f'Gain: {con.dcgain(F):.2f}')
```

```
F:
(s - (26.16+56.63j)) (s - (26.16-56.63j)) (s + 72.44)
-----------------------------------------------------
         (s + 5.936) (s + 10.03) (s + 10.08)

Gain: 469.79
```

[42]:
```
print('Poles', (con.poles(L)))
print('Zeros', (con.zeros(L)))
```

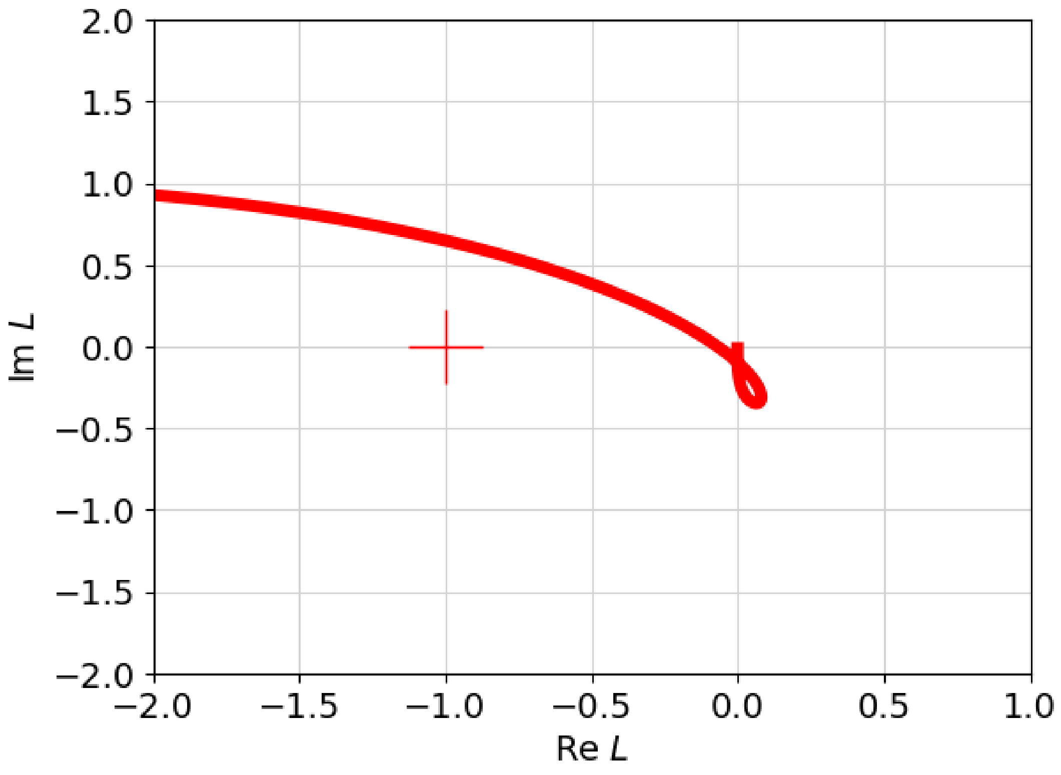
```
Poles [-10.08007675+0.j -10.02837804+0.j  -5.93620394+0.j   0.         +0.j]
Zeros [-25.14726161 +0.j          -0.40240029+14.07807795j
  -0.40240029-14.07807795j]
```

[43]:
```
mag,phase,omega=con.bode_plot(L,margins=True)
SaveFig(SysName,'Bode')
```



[44]:
```
# mag,phase,omega=con.nyquist_plot([L])
con.nyquist_plot(L,mirror_style=False)
plt.xlim(-2,1)
plt.ylim(-2,2)
plt.xlabel('Re $L$')
plt.ylabel('Im $L$')
SaveFig(SysName,'Nyquist')
```

```
[44]: 2

[44]: (-2.0, 1.0)

[44]: (-2.0, 2.0)

[44]: Text(0.5, 0, 'Re $L$')

[44]: Text(0, 0.5, 'Im $L$')
```



```
[45]: gm, pm, wcg, wcp = con.margin(L)
print(gm, pm, wcg, wcp)
freq = wcg/(2*np.pi)
print(f'Frequency: {freq:0.2f}Hz, Period: {1/freq:0.2f}sec')
print(f'$\\theta_{{pm}} = {int(round(pm))}^\\circ$')
print(f'$\\omega_{{pm}} = \\SI{{{wcp:.2f}}}{{\\radian\\per\\second}}$')
```

```
0.14287800387041347 -34.63384689966517 5.829034641551713 10.444107808159126
Frequency: 0.93Hz, Period: 1.08sec
$\theta_{pm} = -35^\circ$
$\omega_{pm} = \SI{10.44}{\radian\per\second}$
```

4.1 Derive linear closed-loop response

- the closed-loop system is derived from the open loop system L(s)
- the input corresponds to an additive input to the product input f_P.
  - this is equivalent to the integrator $1/s$ in the forward path and $L_0(s)$ in the feedback path
  - the feedback() function from the control toolbox is used.
  - a unit impulse on this input is equivalent to a unit perturbation in the *initial* product state $x_P(0)$.

[46]:
```
## Simple integrator 1/s
Integrator = con.tf(1,[1,0])

## Full model
CLL = con.feedback(Integrator,L0)
# CLL = con.feedback(con.series(Integrator,L0))
# CLL = con.feedback(L)
CLL_tf = con.tf(CLL)
print(CLL_tf)
print(con.poles(CLL_tf))

## Reduced model
CLL_r = con.feedback(Integrator,L0_r)
CLL_r_tf = con.tf(CLL_r)
print(CLL_r_tf)
print(con.poles(CLL_r_tf))
```

```
                        (s + 5.936) (s + 10.03) (s + 10.08)
--------------------------------------------------------------------------------
---
(s - (1.326+10.32j)) (s - (1.326-10.32j)) (s + (19.75-10.35j)) (s +
(19.75+10.35j))

[-19.74856549+10.35123185j -19.74856549-10.35123185j
   1.3264659 +10.3244805j    1.3264659 -10.3244805j ]

       (s + (4.006-3.068j)) (s + (4.006+3.068j))
-------------------------------------------------------
(s - (0.9639+10.31j)) (s - (0.9639-10.31j)) (s + 20.74)

[-20.73858697 +0.j           0.96393381+10.30975207j
   0.96393381-10.30975207j]
```

5 Investigate parameter dependencies

[47]:
```
if ParametricVariation:
    if SystemName in␣
↪['Toy','Toy1','Toycc','ToyTwo','ToyTwo3','ToyTwo4','ToyTwo4']:
        RR = np.linspace(1,30,50)
        xlabel = r'$\kappa_{rf}$'
        ACT = [0.2,1,5]
        ActName = '$K_{Act}$'
    elif SystemName in ['Goodwin']:
        RR = np.linspace(1,10,10)
        xlabel = r'$\kappa_{rf}$'
        ACT = [0.5,1,10]
        ActName = '$K_{Act}$'
    elif SystemName in ['Selkov','Selkov1','Selkov3']:
        RR = np.linspace(0.15,1,50)
        xlabel = '$v_{ATP}$'
        ACT = [8,10,12] # used as kappa_rf
        ActName = '$\kappa_{rf}$'
    ylabel = 'Pole'

    # RR = []
    # ACT = []

    PPM = {}
    XX_P_ss = {}
    FREQ = {}
    POLES = {}
    MAXrpole = {}
    MAXipole= {}
    for Act in ACT:
        Freq = []
        PM = []
        Maxrpole = []
        Maxipole = []
        X_P_ss = []
        for rr in RR:
            print(f'rr = {rr:0.2f}')
            if SystemName in ['Toy','Toy1']:
                parameter = SetParameterToy(N=3,kappa_rf=rr,K_Act=Act)
            elif SystemName in ['ToyTwo','ToyTwo3','ToyTwo4']:
                parameter = SetParameterToy(N=2,kappa_rf=rr,K_Act=Act)
            elif SystemName in ['Goodwin']:
                parameter = SetParameterGoodwin(kappa_rf=rr,K_Act=Act)
            elif SystemName in ['Selkov','Selkov1','Selkov3']:
                parameter = SetParameterSelkov(v_ATP=rr,kappa_rf=Act)
            else:
                print('SystemName', SystemName, 'not known')
            LL,LL0,LL0_sys,FF,xx_ss,SS =␣
↪OpenLoop(OutpVar=OutpVar,InpVar=InpVar)

            ## Closed-loop properties
            Integrator = con.tf(1,[1,0])
            CL = con.feedback(Integrator,LL0)
        #           CL = con.minreal(CL)
            gmi, pmi, wcgi, wcpi = con.margin(LL)
            freq = wcpi/(2*np.pi)
            Freq.append(freq)
            PM.append(pmi)
            poles = con.poles(CL)
            if rr == min(RR):
                Poles = poles
            else:
                Poles = np.vstack((Poles,poles))

        #         print(poles)
            maxrpole = max(np.real(poles))
        #         print(f'Max real pole = {maxpole:0.2f}')
            maxipole = max(np.imag(poles))
            Maxrpole.append(maxrpole)
            Maxipole.append(maxipole)

            ## Steady-state properties
            x_P_ss = SS['X'][species.index('P')]
            X_P_ss.append(x_P_ss)


        PPM[Act] = PM
        XX_P_ss[Act] = X_P_ss
        FREQ[Act] = Freq
        POLES[Act] = Poles
        MAXrpole[Act] = Maxrpole
        MAXipole[Act] = Maxipole
```

[48]:
```
if ParametricVariation:
    ## CL poles
    for Act in ACT:
        plt.plot(RR,MAXrpole[Act],label=f'real: {ActName} = {Act}')
        plt.plot(RR,MAXipole[Act],label=f'imag: {ActName} = {Act}')
    plt.grid()
    plt.legend()
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    SaveFig(SysName,'RePole')
```

[49]:
```
if ParametricVariation:
    ## CL poles - root -locus
    for Act in ACT:
        Poles = POLES[Act]
        plt.plot(np.real(Poles),np.imag(Poles),label=f'{ActName} = {Act}')
    if SystemName in ['Toy']:
        plt.xlim(-1,5)
    else:
        plt.xlim(-4,4)
    plt.ylim(bottom=0)
    plt.grid()
    # plt.legend()
    plt.xlabel('Real')
    plt.ylabel('Imag')
    SaveFig(SysName,'RL')
```

[50]:
```
if ParametricVariation:
    ## Steady states
    for Act in ACT:
        plt.semilogy(RR,XX_P_ss[Act],label=f'{ActName} = {Act}')
    plt.grid()
    plt.legend()
    plt.xlabel(xlabel)
    plt.ylabel(r'$\bar{x}_P$')
    SaveFig(SysName,'Pss')
```

[51]:
```
if ParametricVariation:
    ## Phase margin
    for Act in ACT:
        plt.plot(RR,PPM[Act],label=f'{ActName} = {Act}')
    plt.grid()
    plt.legend()
    plt.xlabel(xlabel)
    plt.ylabel('$\phi_m^\circ$')
    SaveFig(SysName,'PM')
```

[52]:
```
if ParametricVariation:
    ## Phase margin frequncy
    for Act in ACT:
        plt.plot(RR,FREQ[Act],label=f'{ActName} = {Act}')
    plt.grid()
    plt.legend()
    plt.xlabel(xlabel)
    plt.ylabel('$F_m$ Hz')
    SaveFig(SysName,'Fm')
```

6 Closed-loop analysis

- The linearised closed-loop system is obtained by linearising the non-linear system from the bond graph
- the transfer function is from the activation (Act) to the product (P)
- the transfer function is compared with that deduced from the loop gain L(s)
  - the denominators are the same
  - the numerators are different as the inputs are different

[53]:
```
## Stoichiometry
chemostats,parameter,InpVar,OutpVar,T,T_long,n_red =␣
↪SetAll(SystemName,Config='Closed',quiet=False)
s,sc = stoichiometry(sys_abg.model(),chemostats=chemostats)


## Linearise
print(InpVar)
X_ss = copy.copy(x_ss)
```

```
TF,Sys = Lin(s,sc,parameter=parameter,x_ss=X_ss,outvar='dX',Inp=[InpVar],␣
↪Outp=['P'])

## Extract closed-loop transfer-function
CL0 = TF[InpVar+'_P']
print(CL0)
CL = IntegrateTF(CL0,crite=0.1)
print(CL)

#     return CL,s,sc
print(chemostats)
```

```
Setting feedback loop with configuration Closed
fb_Act
0 states have been removed from the model

       (s - (26.16+56.63j)) (s - (26.16-56.63j)) (s + 4.22e-15) (s + 72.44)
--------------------------------------------------------------------------------
---
(s - (1.326+10.32j)) (s - (1.326-10.32j)) (s + (19.75-10.35j)) (s +
(19.75+10.35j))


               (s - (26.16+56.63j)) (s - (26.16-56.63j)) (s + 72.44)
--------------------------------------------------------------------------------
---
(s - (1.326+10.32j)) (s - (1.326-10.32j)) (s + (19.75-10.35j)) (s +
(19.75+10.35j))

['fb_Act', 'fb_E0', 'decr1_A', 'decr1_Zf', 'decr2_A', 'decr2_Zf', 'decr3_A',
'decr3_Zf']
```

[54]:
```
## Reactions and properties
disp.Latex(st.sprintrl(s,all=True,chemformula=False))
# Properties(s,sc)

print('Pools:')
disp.Latex(st.sprintml(s))
print('Paths')
print(st.sprintp(sc))
```

[54]:

$$fb_Act + fb_E0 \Leftrightarrow E1 + P \tag{53}$$
$$2E1 + decr1_A \Leftrightarrow 2E1 + E2 \tag{54}$$
$$E2 \Leftrightarrow decr1_Zf \tag{55}$$
$$2E2 + decr2_A \Leftrightarrow 2E2 + E3 \tag{56}$$
$$E3 \Leftrightarrow decr2_Zf \tag{57}$$
$$2E3 + decr3_A \Leftrightarrow 2E3 + P \tag{58}$$
$$P \Leftrightarrow decr3_Zf \tag{59}$$

```
Pools:
```
[54]:

$$
\begin{aligned}
fb_Act &\Leftrightarrow fb_E0 && (60)\\
&\Leftrightarrow fb_Act + E1 && (61)\\
&\Leftrightarrow E2 + decr1_A + decr1_Zf && (62)\\
&\Leftrightarrow E3 + decr2_A + decr2_Zf && (63)\\
&\Leftrightarrow fb_Act + P + decr3_A + decr3_Zf && (64)
\end{aligned}
$$

```
Paths
3 pathways
0:  + decr1_r + decr1_rf
1:  + decr2_r + decr2_rf
2:  + decr3_r + decr3_rf
```

6.1 Compare two versions of closed-loop TF

[55]:
```
# CL,s,sc = ClosedLoopTF(SystemName,sys_abg.model(),InpVar,parameter,x_ss)
print('CLL:',CLL)
print('CL:',CL)
print(f'Gain: {con.dcgain(CL):.2f}')
```

```
CLL:
                        (s + 5.936) (s + 10.03) (s + 10.08)
--------------------------------------------------------------------------------
---
(s - (1.326+10.32j)) (s - (1.326-10.32j)) (s + (19.75-10.35j)) (s +
(19.75+10.35j))

CL:
               (s - (26.16+56.63j)) (s - (26.16-56.63j)) (s + 72.44)
--------------------------------------------------------------------------------
---
(s - (1.326+10.32j)) (s - (1.326-10.32j)) (s + (19.75-10.35j)) (s +
(19.75+10.35j))

Gain: 5.23
```

[56]:
```
print('Poles:', (con.poles(CL)))
```

```
Poles: [-19.74856549+10.35123185j -19.74856549-10.35123185j
   1.3264659 +10.3244805j    1.3264659 -10.3244805j ]
```

[57]:
```
CLr = balred(con.ss(CL),3)
print(con.tf(CLr))
poles = con.poles(CLr)
print('Poles:', poles)
```

```
   3.994 (s - (30.61+39.16j)) (s - (30.61-39.16j))
```

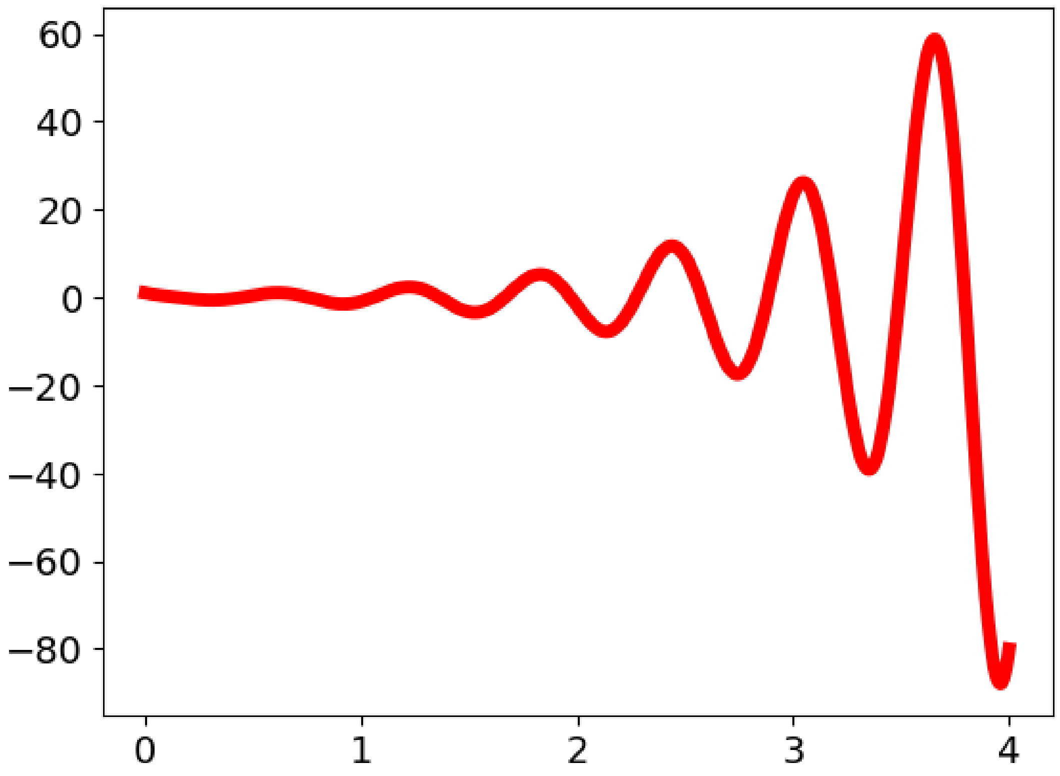
```
-----------------------------------------------------
(s - (1.326+10.32j)) (s - (1.326-10.32j)) (s + 16.97)

Poles: [  1.3264659 +10.3244805j   1.3264659 -10.3244805j
 -16.96616984 +0.j        ]
```

[58]:
```
print(abs(poles[0]))
Omega = abs(np.imag(poles[0]))
Freq = Omega/(2*np.pi)
print(Freq,1/Freq)
```

```
10.40934241269315
1.6431921071431235 0.6085715697226747
```

6.2 Linear time response

[59]:
```
# y_cl = impulse_response(CL,T=T)
y_cll = impulse_response(CLL,T=T)
#y_cll_r = impulse_response(CLL_r,T=T)
```



[60]:
```
# print(CL.state_labels)
```

6.3 Non-linear simulation

- the non-linear simulation has the initial state the same as the steady state *except* that the intial value $x_P(0)$ of the product state x_P is perturbed by pert.

- the trajectory $x_P(t)$ from the nonlinear system is compared with the impulse response of the linearised system
  - the input to the the linearised system is pert*unit_impulse.

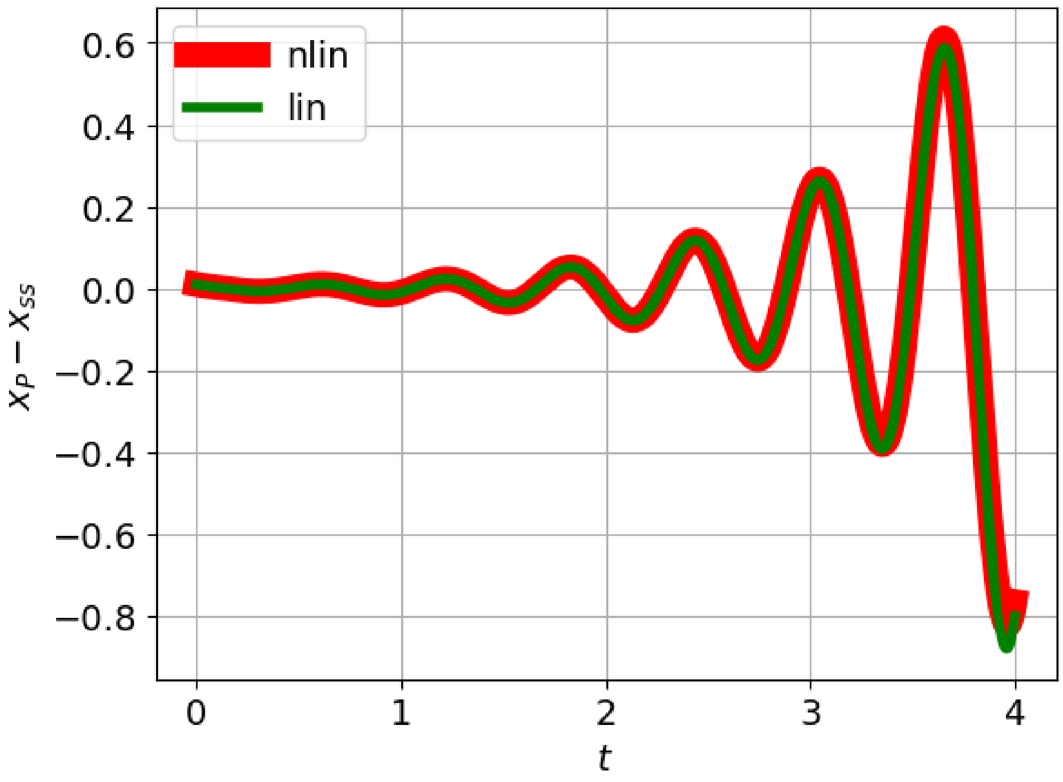
[61]:
```
## Simulate
print(InpVar)
species = s['species']
print(species)
reaction = s['reaction']
X0 = copy.copy(x_ss)
pert = 1e-2
# X0[species.index(InpVar)] *= 1+pert
X0[species.index('P')] += pert
ndat = st.sim(s,sc=sc,t=T,X0=X0,parameter=parameter,quiet=False)
```

```
fb_Act
['fb_Act', 'fb_E0', 'E1', 'E2', 'E3', 'P', 'decr1_A', 'decr1_Zf', 'decr2_A',
'decr2_Zf', 'decr3_A', 'decr3_Zf']
Setting K_fb_Act to 1
Setting K_E1 to 1
Setting K_E2 to 1
Setting K_E3 to 1
Setting K_decr1_A to 100.0
Setting K_decr1_Zf to 1e-06
Setting K_decr2_A to 100.0
Setting K_decr2_Zf to 1e-06
Setting K_decr3_A to 100.0
Setting K_decr3_Zf to 1e-06
Setting kappa_decr1_r to 1.0
Setting kappa_decr1_rf to 10
Setting kappa_decr2_r to 1.0
Setting kappa_decr2_rf to 10
Setting kappa_decr3_r to 1.0
Setting kappa_decr3_rf to 10
```

[62]:
```
if SystemName in␣
↪['Toy','Toy1','Toycc','ToyTwo','ToyTwo3','ToyTwo4','Goodwin']:
    Yname = 'P'
    Xlabel = 't'
    Ylabel = 'P'
elif SystemName in ['Selkov','Selkov1','Selkov3']:
    Yname = 'P'
    Xlabel = 't'
    Ylabel = 'ADP'
else:
    print('System Name',SystemName,'is not known')

y_n = (ndat['X'][:,species.index('P')] - x_ss[species.index('P')] )
plt.plot(T,y_n, label='nlin',lw=10)
plt.plot(T,pert*y_cll,label='lin',lw=4)
# plt.plot(T,pert*y_cll_r,label='lin (reduced)',lw=4,ls='dashed')
```

```
plt.grid()
plt.legend()
plt.xlabel('$t$')
plt.ylabel('$x_{'+Ylabel+'} - x_{ss}$')
if SysName == 'Selkov3':
    plt.xlim((0,0.1))
    plt.ylim((0,0.1))
SaveFig(SysName,'Simulation')
```

```
[62]: [<matplotlib.lines.Line2D at 0x7f6d65ab1c40>]

[62]: [<matplotlib.lines.Line2D at 0x7f6d65ab1880>]

[62]: <matplotlib.legend.Legend at 0x7f6d6531e790>

[62]: Text(0.5, 0, '$t$')

[62]: Text(0, 0.5, '$x_{P} - x_{ss}$')
```

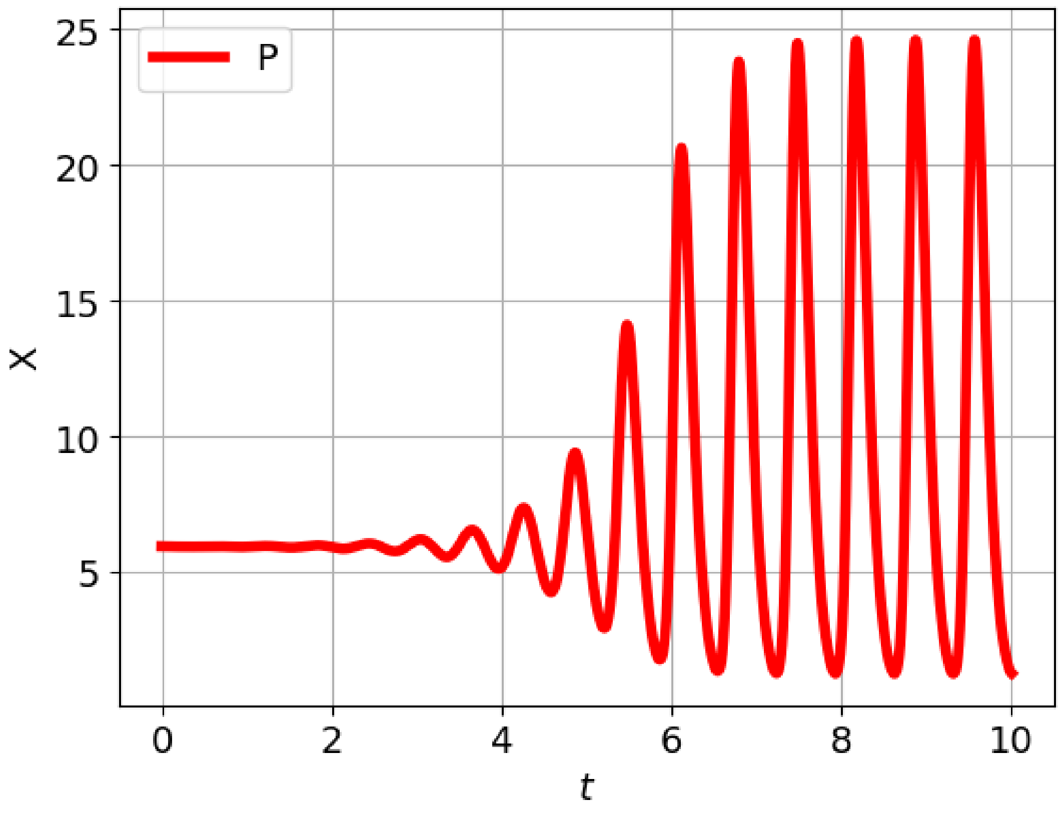


```
[63]: # st.plot(s,ndat,species=['P'],reaction=[])
      # st.plot(s,ndat,species=['E1'],reaction=[])
      # st.plot(s,ndat,species=['E2'],reaction=[])
      # st.plot(s,ndat,species=['E3'],reaction=[])
```

```
[64]: ndat = st.sim(s,sc=sc,t=T_long,X0=X0,parameter=parameter,quiet=False)
```

```
Setting K_fb_Act to 1
Setting K_E1 to 1
Setting K_E2 to 1
Setting K_E3 to 1
Setting K_decr1_A to 100.0
Setting K_decr1_Zf to 1e-06
Setting K_decr2_A to 100.0
Setting K_decr2_Zf to 1e-06
Setting K_decr3_A to 100.0
Setting K_decr3_Zf to 1e-06
Setting kappa_decr1_r to 1.0
Setting kappa_decr1_rf to 10
Setting kappa_decr2_r to 1.0
Setting kappa_decr2_rf to 10
Setting kappa_decr3_r to 1.0
Setting kappa_decr3_rf to 10
```

[65]:
```
st.plot(s,ndat,species=['P'],reaction=[])
# st.plot(s,ndat,species=['E1'],reaction=[])
# st.plot(s,ndat,species=['E2'],reaction=[])
# st.plot(s,ndat,species=['E3'],reaction=[])
```



6.4 Phase plane

[66]:
```
print(species)
if SystemName in␣
↪['Toy','Toy1','Toycc','ToyTwo','ToyTwo3','ToyTwo4','Goodwin']:
    Xname = 'E1'
    Yname = 'P'
    Xlabel = Xname
    Ylabel = Yname
elif SystemName in ['Selkov','Selkov1','Selkov3']:
    Xname = 'selkov_ATP'
    Yname = 'P'
    Xlabel = 'ADP'
    Ylabel = 'ATP'
else:
    print('System Name',SystemName,'is not known')
i_X = species.index(Xname)
x_X = ndat['X'][:,i_X]
i_Y = species.index(Yname)
x_Y = ndat['X'][:,i_Y]
plt.plot(x_X,x_Y,lw=1,color='black')

plt.plot(x_ss[i_X],x_ss[i_Y],marker='+',color='black')
plt.locator_params(nbins=4)

# plt.plot(X[:,0],X[:,2])
plt.xlabel(f'$x_{{{Xlabel}}}$')
plt.ylabel(f'$x_{{{Ylabel}}}$')
# plt.xlim(left=0)
# plt.ylim(bottom=0)
plt.grid()


SaveFig(SysName,'PhasePlane')
```

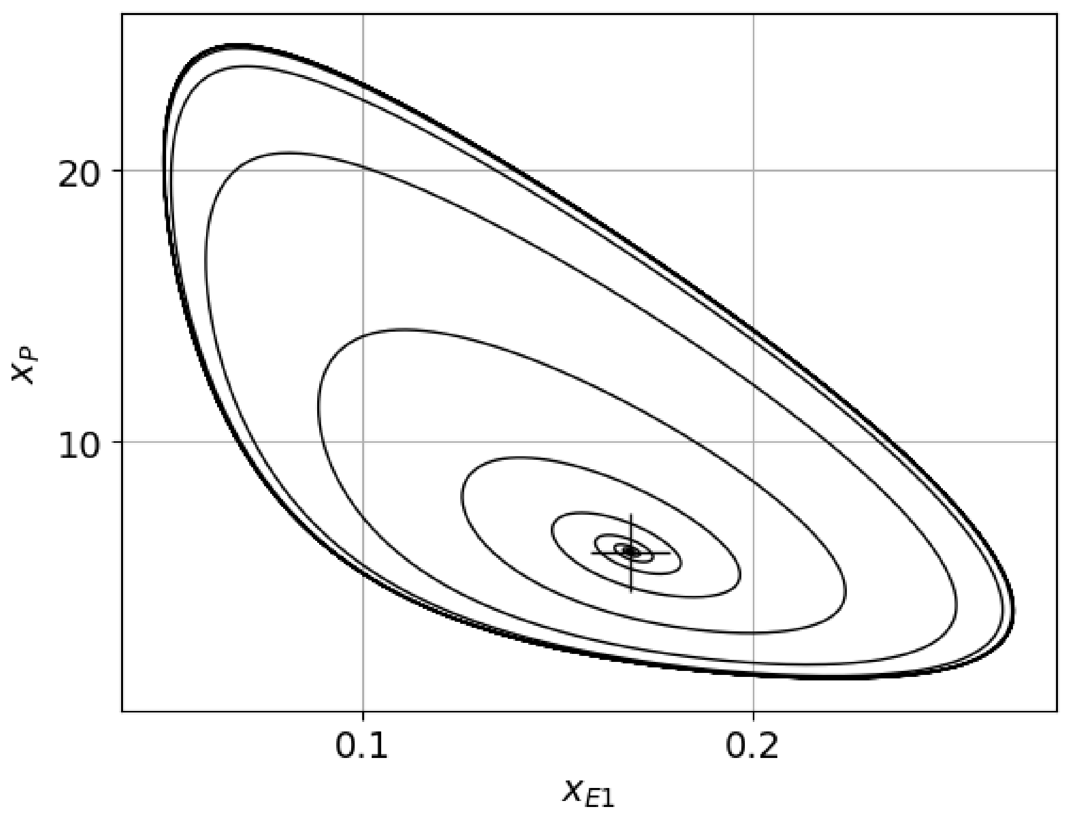
```
['fb_Act', 'fb_E0', 'E1', 'E2', 'E3', 'P', 'decr1_A', 'decr1_Zf', 'decr2_A',
'decr2_Zf', 'decr3_A', 'decr3_Zf']
```

[66]:
```
[<matplotlib.lines.Line2D at 0x7f6d64b6b880>]
```

[66]:
```
[<matplotlib.lines.Line2D at 0x7f6d64f8dca0>]
```

[66]:
```
Text(0.5, 0, '$x_{E1}$')
```

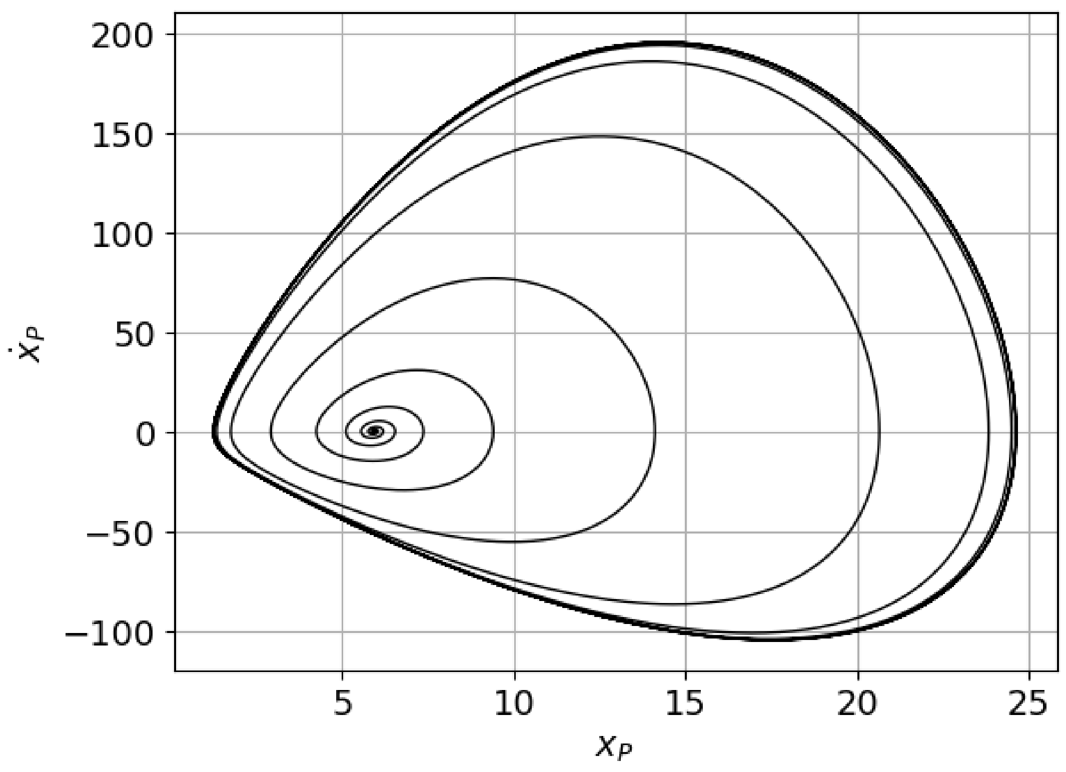
[66]:
```
Text(0, 0.5, '$x_{P}$')
```



6.5 Signals at integrator

```
[67]: i_P = species.index('P')
      x_X = ndat['X'][10:,i_P]
      x_Y = ndat['dX'][10:,i_P]
      tt = ndat['t'][10:]
      plt.plot(x_X,x_Y,lw=1,color='black')
      plt.xlabel('$x_P$')
      plt.ylabel('$\dot{x}_P$')
      # plt.xlim(left=0)
      # plt.ylim(bottom=0)
      plt.grid()

      SaveFig(SysName,'PhasePlaneP')
```

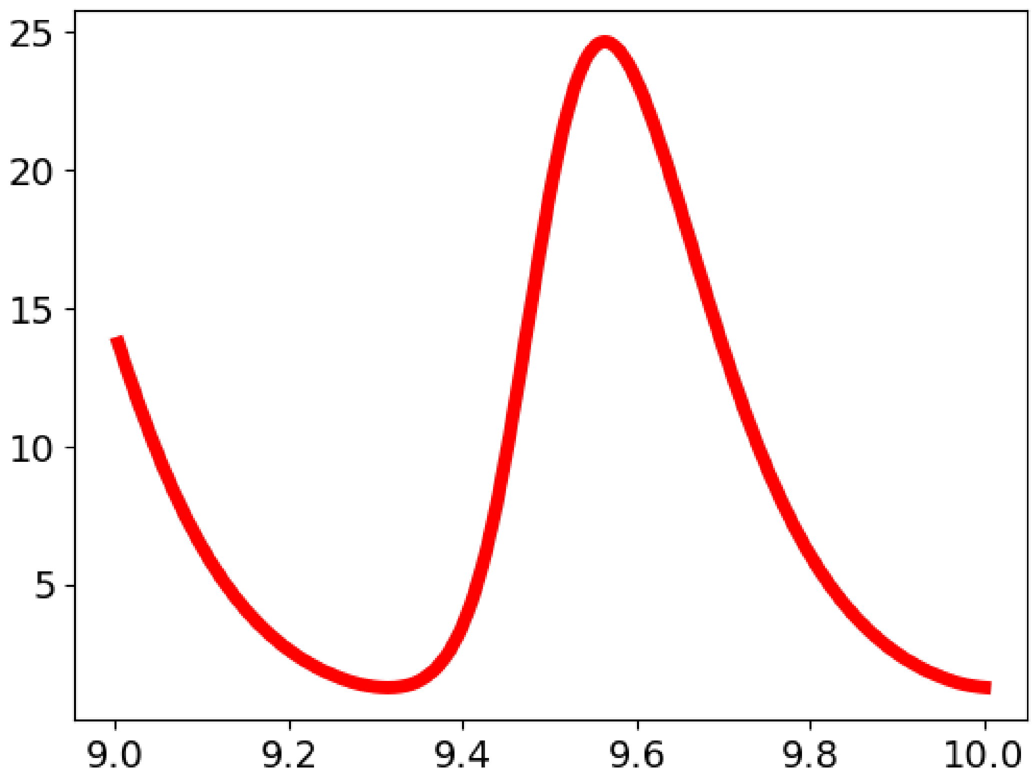
```
[67]: [<matplotlib.lines.Line2D at 0x7f6d64bcbd90>]

[67]: Text(0.5, 0, '$x_P$')

[67]: Text(0, 0.5, '$\\dot{x}_P$')
```



[68]:
```
i0 = int(0.9*len(tt))
print(i0)
plt.plot(tt[i0:],x_X[i0:])
```

```
4491
```

[68]:
```
[<matplotlib.lines.Line2D at 0x7f6d64b94970>]
```



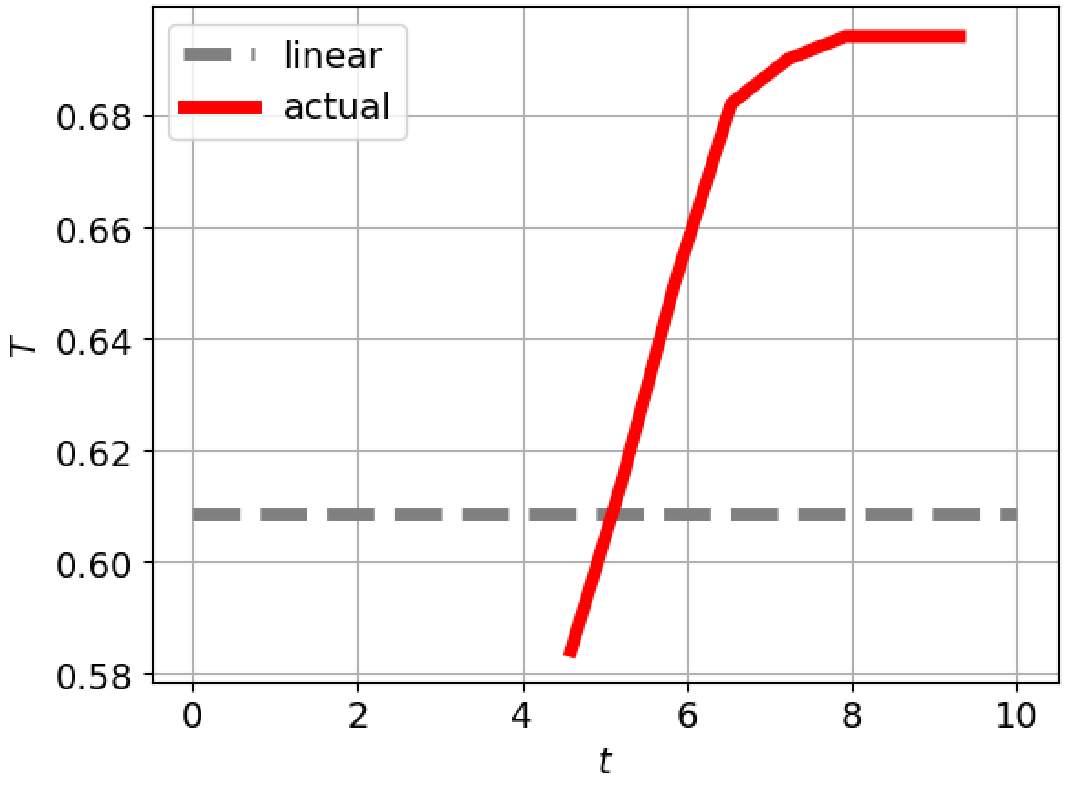
6.6 Period

```
[69]: x_Y_ss = x_ss[i_Y]
      # print(x_Y_ss)
      i_zc = zero_crossings(x_Y-x_Y_ss)
      t = ndat['t']
      t_zc = t[i_zc]
      T_zc = np.diff(t_zc)

      plt.hlines([1/Freq],min(t),max(t),ls='dashed',color='grey',label='linear')
      plt.plot(t_zc[1:],T_zc, label='actual')

      plt.grid()
      plt.legend()
      plt.xlabel('$t$')
      plt.ylabel('$T$')
      SaveFig(SysName,'Period')

      # print(zc)
      # print(1/Freq)
```

```
[69]: <matplotlib.collections.LineCollection at 0x7f6d64fd79d0>
```

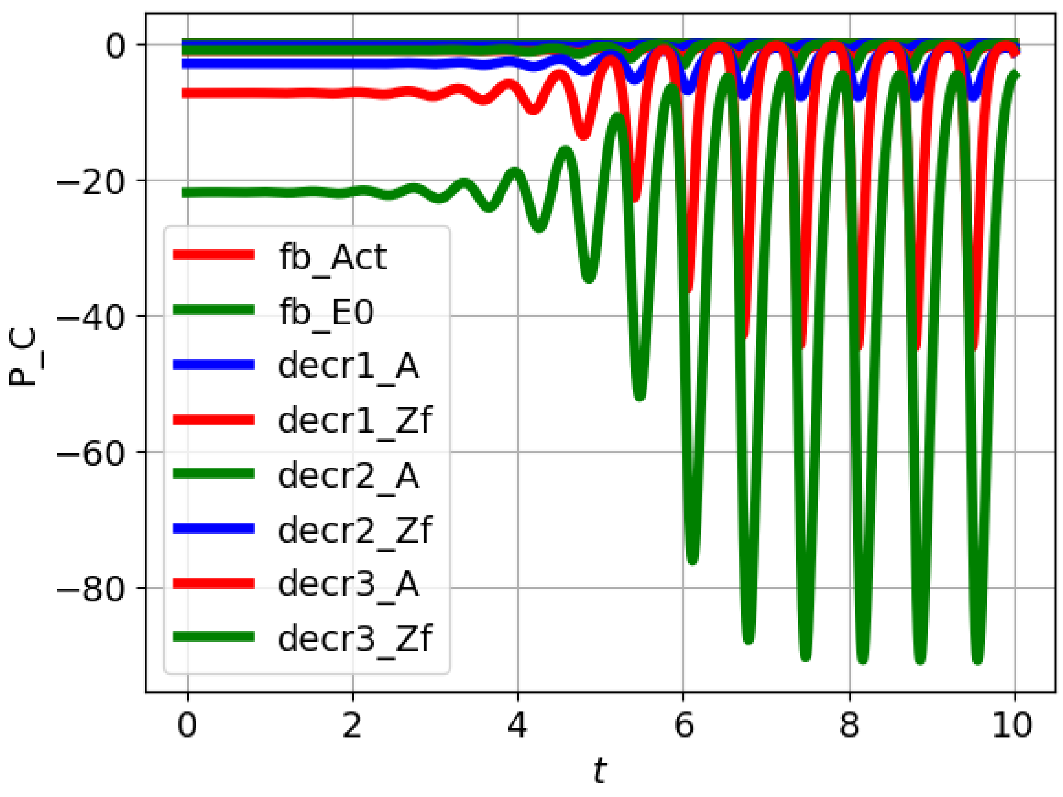
```
[69]: [<matplotlib.lines.Line2D at 0x7f6d64a737f0>]
```

```
[69]: <matplotlib.legend.Legend at 0x7f6d64fc7e20>

[69]: Text(0.5, 0, '$t$')

[69]: Text(0, 0.5, '$T$')
```



7 Power

```
[70]: st.plot(s,ndat,plotPower=True,species=chemostats,reaction=[])
```



[71]:
```
# st.plot(s,ndat,plotPower=True,species=[],reaction=['decr3_r','decr3_rf'])
```

7.1 Power

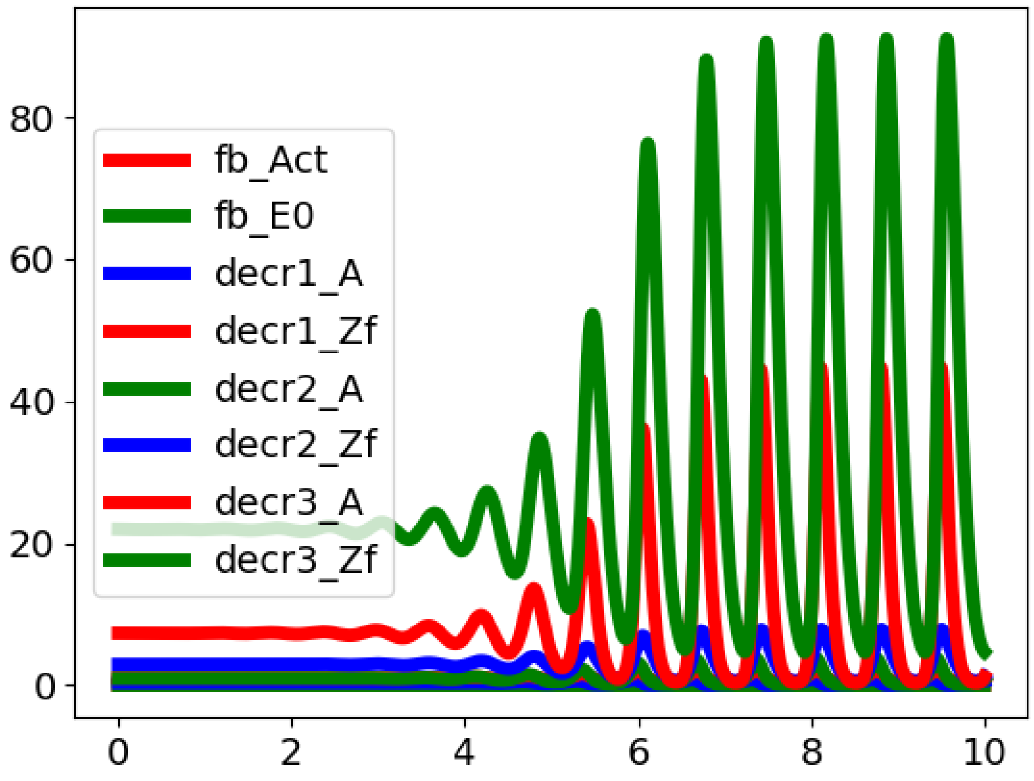
[72]:
```
P_Re = ndat['P_Re']
P_C = -ndat['P_C']
i_chemo = []
for chemo in chemostats:
    i_chemo.append(species.index(chemo))

free = list(set(species)-set(chemostats))
print(free)
i_free = []
for fr in free:
    i_free.append(species.index(fr))

P_chemo = P_C[:,i_chemo]
P_free = P_C[:,i_free]

## Total power
PP_Re = np.sum(P_Re,axis=1)
PP_C = np.sum(P_C,axis=1)
PP_chemo = np.sum(P_chemo,axis=1)
PP_free = np.sum(P_free,axis=1)
```

```
['E3', 'P', 'E2', 'E1']
```

[73]:
```
t = ndat['t']
plt.plot(t,P_chemo)
plt.legend(chemostats,loc='center left')
```

[73]:
```
[<matplotlib.lines.Line2D at 0x7f6d64fb1790>,
 <matplotlib.lines.Line2D at 0x7f6d64744e20>,
 <matplotlib.lines.Line2D at 0x7f6d646dd610>,
 <matplotlib.lines.Line2D at 0x7f6d6472c520>,
 <matplotlib.lines.Line2D at 0x7f6d6476a820>,
 <matplotlib.lines.Line2D at 0x7f6d6476a280>,
 <matplotlib.lines.Line2D at 0x7f6d6476a8e0>,
 <matplotlib.lines.Line2D at 0x7f6d6476a850>]
```

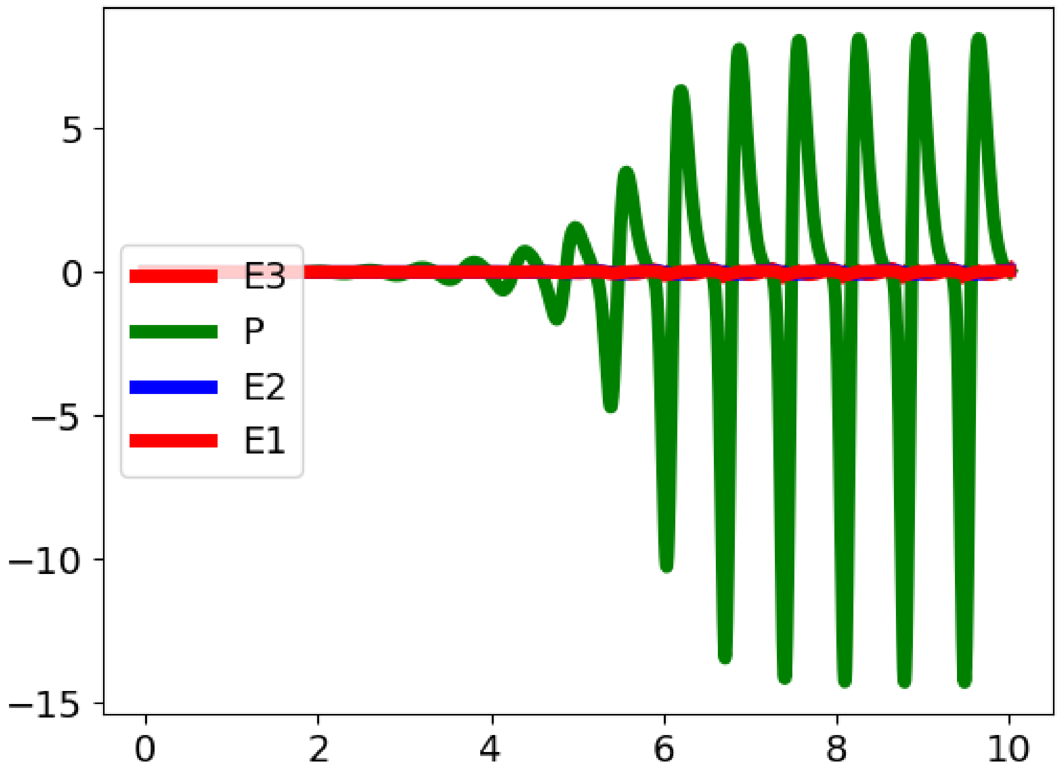
[73]:
```
<matplotlib.legend.Legend at 0x7f6d6471ab50>
```



[74]:
```
t = ndat['t']
plt.plot(t,P_free)
plt.legend(free,loc='center left')
```

[74]:
```
[<matplotlib.lines.Line2D at 0x7f6d646779d0>,
 <matplotlib.lines.Line2D at 0x7f6d64677a00>,
 <matplotlib.lines.Line2D at 0x7f6d64677a30>,
 <matplotlib.lines.Line2D at 0x7f6d64649cd0>]
```

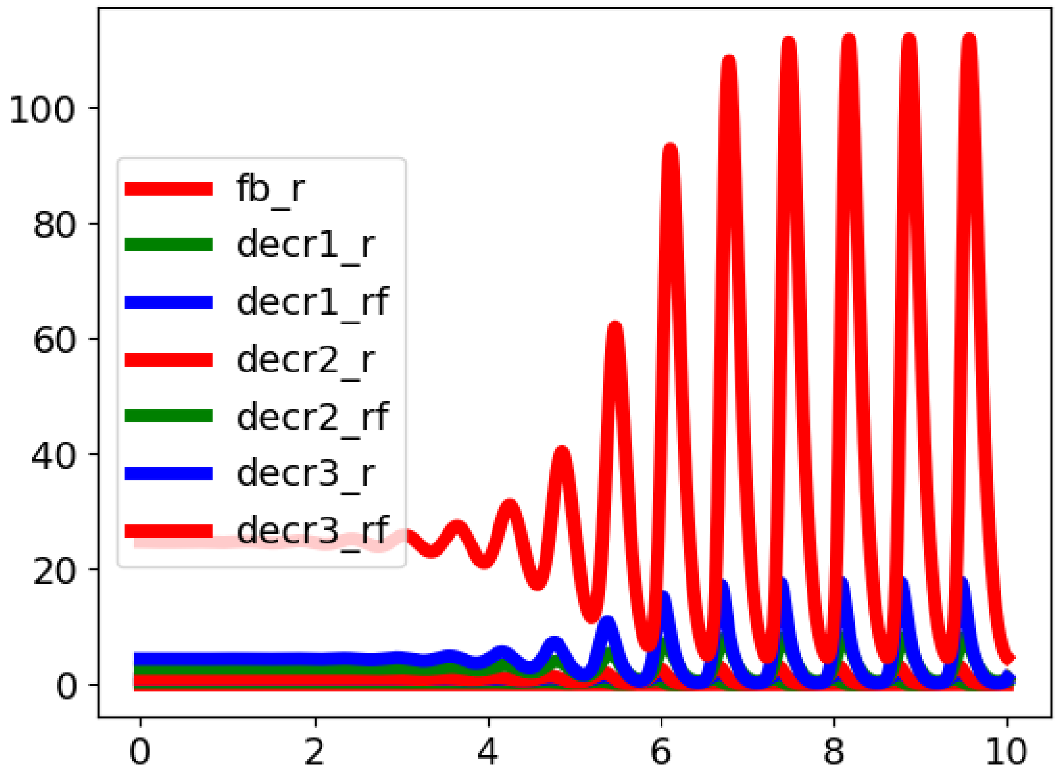
[74]: <matplotlib.legend.Legend at 0x7f6d647534c0>



[75]:
```
t = ndat['t']
plt.plot(t,P_Re)
plt.legend(reaction,loc='center left')
```

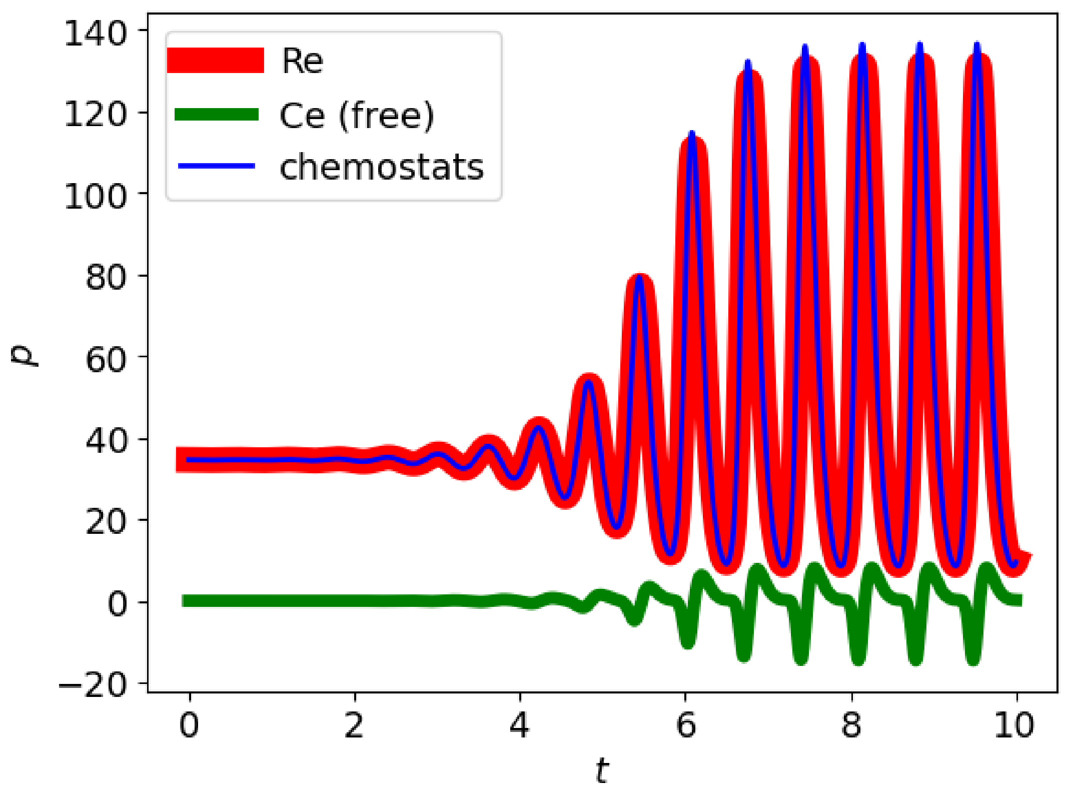
[75]:
```
[<matplotlib.lines.Line2D at 0x7f6d643bce50>,
 <matplotlib.lines.Line2D at 0x7f6d64753f40>,
 <matplotlib.lines.Line2D at 0x7f6d64753f70>,
 <matplotlib.lines.Line2D at 0x7f6d643bce80>,
 <matplotlib.lines.Line2D at 0x7f6d6439c1f0>,
 <matplotlib.lines.Line2D at 0x7f6d646adbe0>,
 <matplotlib.lines.Line2D at 0x7f6d643ac370>]
```

[75]: <matplotlib.legend.Legend at 0x7f6d64385430>



```
[76]: t = ndat['t']
      plt.plot(t,PP_Re,label='Re',lw=10)
      plt.plot(t,PP_free,label='Ce (free)',lw=5)
      plt.plot(t,PP_chemo,label='chemostats',lw=2)
      plt.legend()
      plt.xlabel('$t$')
      plt.ylabel('$p$')
      SaveFig(SysName,'Power')
```

```
[76]: [<matplotlib.lines.Line2D at 0x7f6d642e1af0>]

[76]: [<matplotlib.lines.Line2D at 0x7f6d642e1ac0>]

[76]: [<matplotlib.lines.Line2D at 0x7f6d642e1eb0>]

[76]: <matplotlib.legend.Legend at 0x7f6d64699dc0>

[76]: Text(0.5, 0, '$t$')

[76]: Text(0, 0.5, '$p$')
```

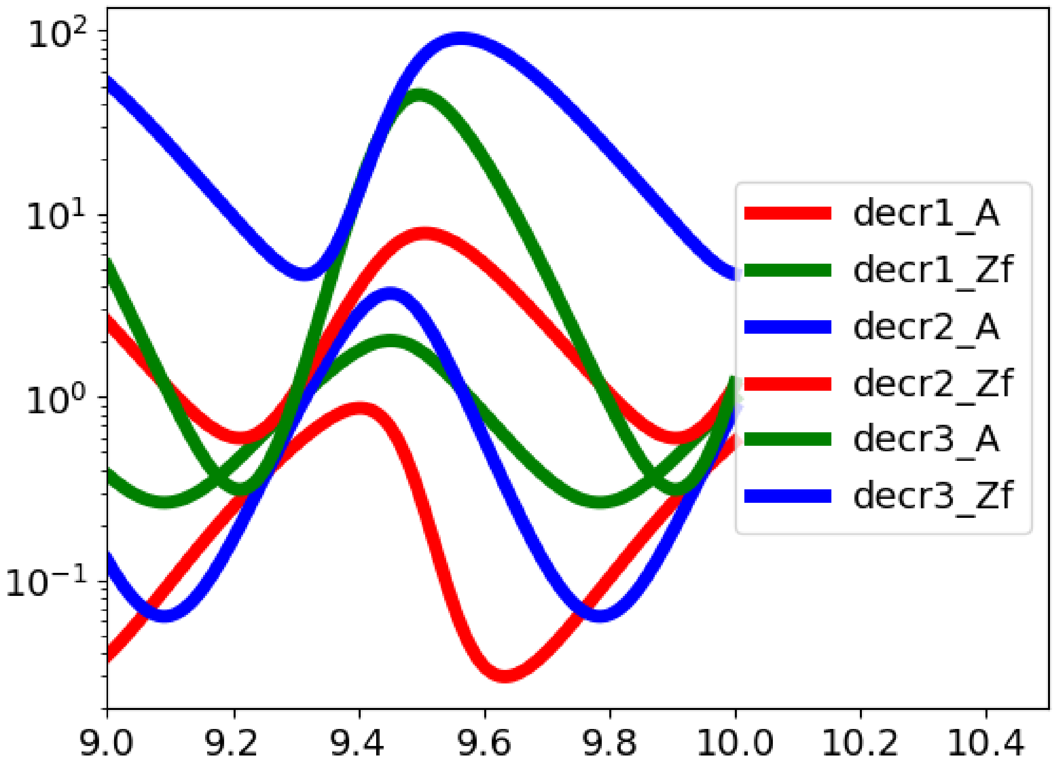


[77]:
```
plt.semilogy(t,P_chemo[:,2:])
plt.xlim(left=0.9*max(t))
plt.legend(sc['chemostats'][2:],loc='center right')
```

[77]:
```
[<matplotlib.lines.Line2D at 0x7f6d63fc1760>,
 <matplotlib.lines.Line2D at 0x7f6d63fc1790>,
 <matplotlib.lines.Line2D at 0x7f6d63fc17c0>,
 <matplotlib.lines.Line2D at 0x7f6d642e1d90>,
 <matplotlib.lines.Line2D at 0x7f6d64259c70>,
 <matplotlib.lines.Line2D at 0x7f6d63fc18b0>]
```

[77]:
```
(9.0, 10.5)
```

[77]:
```
<matplotlib.legend.Legend at 0x7f6d63fd25e0>
```



8 Split-loop analysis

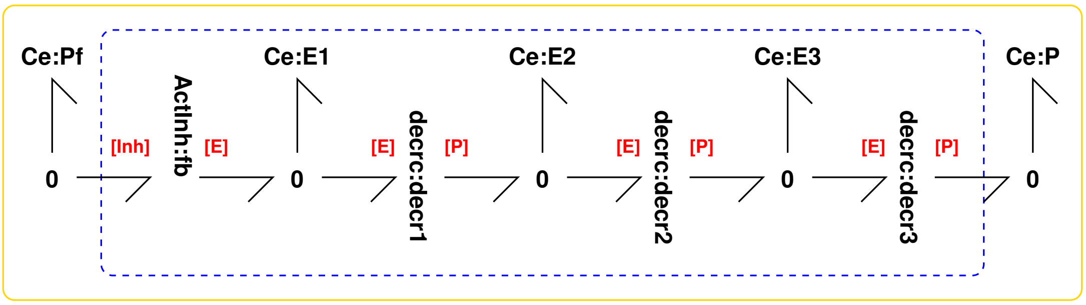
8.1 Model

[78]:
```
name = SystemName+'SL_abg'
svg = name+'.svg'
print('Using',svg)
sbg.model(svg,convertCe=True,convertR=True,quiet=quiet)
exec(f'import {name} as sys_abg')
imp.reload(sys_abg)
disp.SVG(svg)

SplitVar = 'Pf'
```

```
Using ToySL_abg.svg
Creating subsystem: ActInh:fb
Creating subsystem: decrc:decr1
Creating subsystem: decrc:decr2
Creating subsystem: decrc:decr3
{}
```

[78]:
```
<module 'ToySL_abg' from
'/home/peterg/WORK/Research/SystemsBiology/Notes/2024/Oscillation/ToySL_abg.
 ↪py'>
```

[78]:



8.2 Stoichiometry

[79]:
```
chemostats = SetChemostats(SystemName,Config='SplitLoop',quiet=False)
```

```
Setting feedback loop with configuration SplitLoop
```

[80]:
```
print(chemostats)
s,sc = stoichiometry(sys_abg.model(),chemostats=chemostats)
species_sl = s['species']
print(species_sl)
```

```
['fb_Act', 'fb_E0', 'Pf', 'P', 'decr1_A', 'decr1_Zf', 'decr2_A', 'decr2_Zf',
'decr3_A', 'decr3_Zf']
['fb_Act', 'fb_E0', 'E1', 'E2', 'E3', 'P', 'Pf', 'decr1_A', 'decr1_Zf',
'decr2_A', 'decr2_Zf', 'decr3_A', 'decr3_Zf']
```

8.3 Steady-state analysis

[81]:
```
## Create the steady state corresponding to open loop with x_inh=x_P:
parameter = SetParameter()
x_sl_ss = np.ones(s['n_X'])
X_ss = copy.copy(x_ss)
for i,spec in enumerate(species):
    x_sl_ss[species_sl.index(spec)] = X_ss[i]
x_sl_ss[species_sl.index(SplitVar)] = X_ss[species.index('P')]
```

8.4 Linearise

[82]:
```
# ttff = con.zpk([-1,-2],[-3,-4,-5],123,display_format='zpk')
# ttff
```

[83]:
```
Inp = [SplitVar,OutpVar]
TF,Sys =␣
 ↪Lin(s,sc,parameter=parameter,x_ss=x_sl_ss,outvar='dX',Inp=Inp,Outp=Inp)
```

```
2 states have been removed from the model
0 states have been removed from the model
3 states have been removed from the model
3 states have been removed from the model
```

[84]:
```
for name in TF:
    print(name)
    TFr = con.minreal(TF[name])
    TFr
    # zTF = zpk(TF[name])
    # zTF
    print(con.poles(TF[name]))
```

```
Pf_Pf
0 states have been removed from the model
```

[84]:

$$\frac{-0.1685(s)}{s+5.936}$$

```
[-5.93620394+0.j]
Pf_P
0 states have been removed from the model
```

[84]:

$$\frac{-1.639\times10^{-12}(s+2.897\times10^{16})}{(s+5.936)(s+10.03)(s+10.08)}$$

```
[-10.08007675+0.j -10.02837804+0.j  -5.93620394+0.j]
P_Pf
0 states have been removed from the model
```

[84]:

$$\frac{0}{1}$$

```
[]
P_P
0 states have been removed from the model
```

[84]:

$$\frac{-10.63}{1}$$

```
[]
```

8.5 Active and passive loop gains

[85]:
```
LL0 = con.tf(0,1)
L_pas_0 = con.tf(0,1)
for index in TF:
    if not index in ['Pf_P']:
        L_pas_0 = con.minreal(con.parallel(L_pas_0,-TF[index]))

    LL0 = con.minreal(con.parallel(LL0,-TF[index]))

# LL0 = con.minreal(LL0)
print('L0')
LL0
```

```
L_act_0 = -TF[SplitVar+'_P']
print('L0_act')
L_act_0

# L_pas_0 = con.minreal(L_pas_0)
print('L0_pas')
L_pas_0

LL = IntegrateTF(LL0)
# LL = con.minreal(LL)
# LL = con.tf(balred(con.ss(LL),3))

print('L')
L
print('LL')
LL

L_act = IntegrateTF(L_act_0)
print('L_act')
L_act

# L_pas_0 = con.parallel(-TF['P_P'],-TF[SplitVar+'_'+SplitVar])

L_pas = IntegrateTF(L_pas_0)
print('L_pas')
L_pas

# L_pas_P = IntegrateTF(-TF['P_P'])
# print('L_pas_P', L_pas_P)

# L_pas_Inh = IntegrateTF(-TF[SplitVar+'_'+SplitVar])
# print('L_pas_Inh', L_pas_Inh)
```

```
0 states have been removed from the model
0 states have been removed from the model
1 states have been removed from the model
0 states have been removed from the model
0 states have been removed from the model
0 states have been removed from the model
0 states have been removed from the model
L0
```

[85]:

$$\frac{10.8(s+(0.4024-14.08j))(s+(0.4024+14.08j))(s+25.15)}{(s+5.936)(s+10.03)(s+10.08)}$$

```
L0_act
```

[85]:

$$\frac{1.639\times10^{-12}(s+2.897\times10^{16})}{(s+5.936)(s+10.03)(s+10.08)}$$

```
L0_pas
```

[85]:

$$\frac{10.8(s+5.844)}{s+5.936}$$

L

[85]:

$$\frac{10.8(s+(0.4024-14.08j))(s+(0.4024+14.08j))(s+25.15)}{(s)(s+5.936)(s+10.03)(s+10.08)}$$

LL

[85]:

$$\frac{10.8(s+(0.4024-14.08j))(s+(0.4024+14.08j))(s+25.15)}{(s)(s+5.936)(s+10.03)(s+10.08)}$$

L_act

[85]:

$$\frac{1.639\times 10^{-12}(s+2.897\times 10^{16})}{(s)(s+5.936)(s+10.03)(s+10.08)}$$

L_pas

[85]:

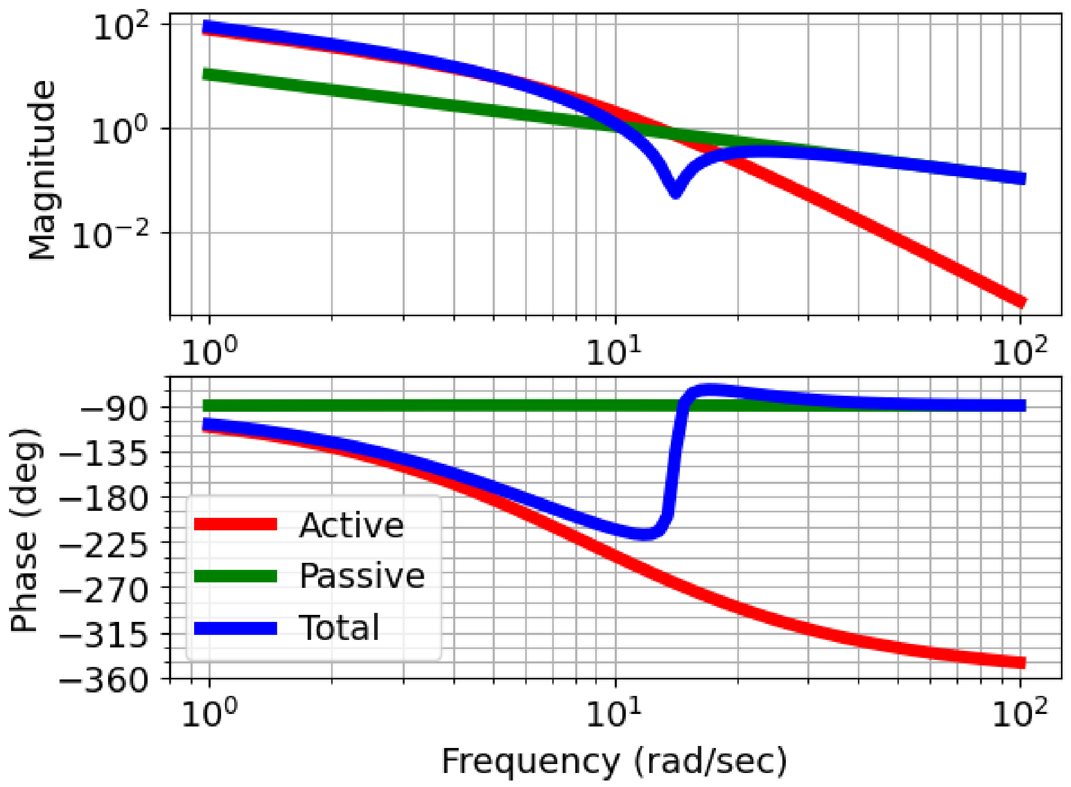
$$\frac{10.8(s+5.844)}{(s)(s+5.936)}$$

[86]:
```
# print(f'Gain L0 = {con.dcgain(L0):.2f}')
# L0_pas_P = TF['P_P']
# K =  parameter['K_decr3_A']*parameter['kappa_decr3_r']
# K_f =  parameter['kappa_decr3_rf']
# print(f'Gain L0_pas_P = {con.dcgain(L0_pas_P):0.2f} ({K_f:0.2f})')
```

8.6 Bode plots

[87]:
```
## Bode
L_list = [L_act,L_pas,LL]
omega = np.logspace(0,2,100)
mag,phase,om=con.bode_plot(L_list,omega)
plt.legend(['Active','Passive','Total'])
SaveFig(SysName,'SplitBode')
```

[87]:
```
<matplotlib.legend.Legend at 0x7f6d64ef5100>
```

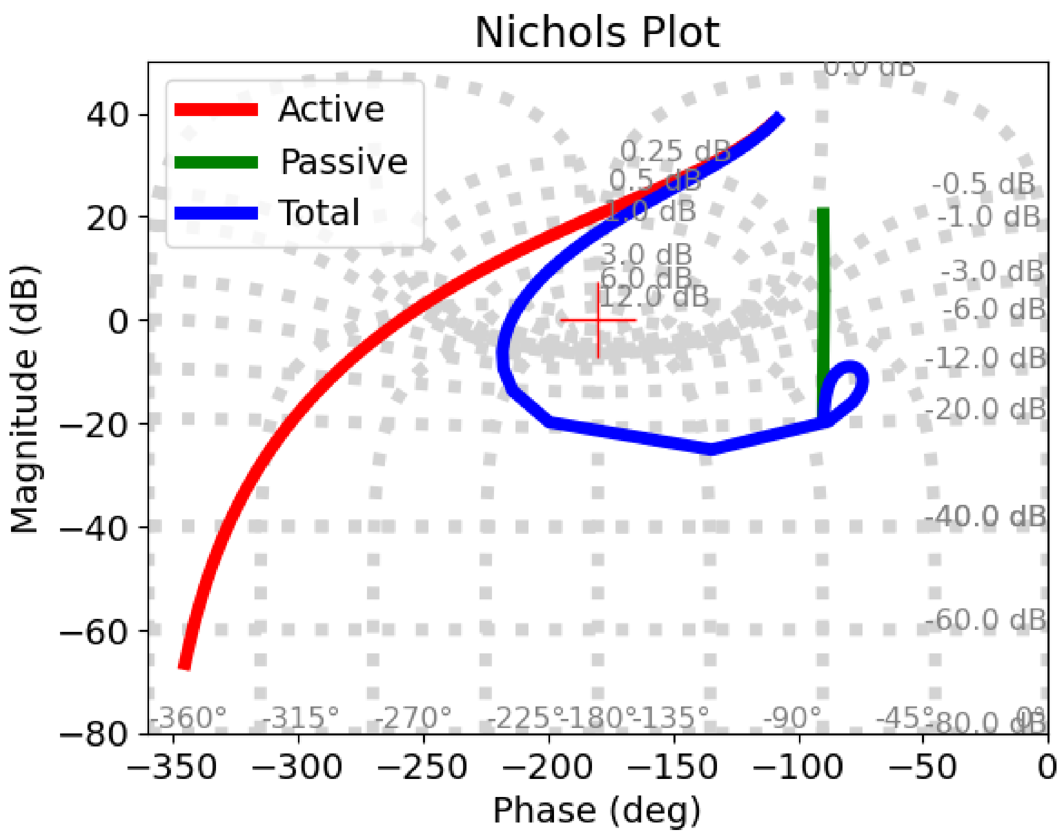


[88]:
```
if SaveData:
    BodeData = {}
    BodeData['L_list'] = L_list
    BodeData['mag'] = mag
    BodeData['phase'] = phase
    BodeData['omega'] = om
    BodeData['wcp'] = wcp
    BodeData['pm'] = pm
    SavedData['Bode'] = BodeData
```

8.7 Nichols plots

[89]:
```
## Nichols
con.nichols_plot(L_list,omega)
plt.legend(['Active','Passive','Total'])
SaveFig(SysName,'SplitNichols')
```

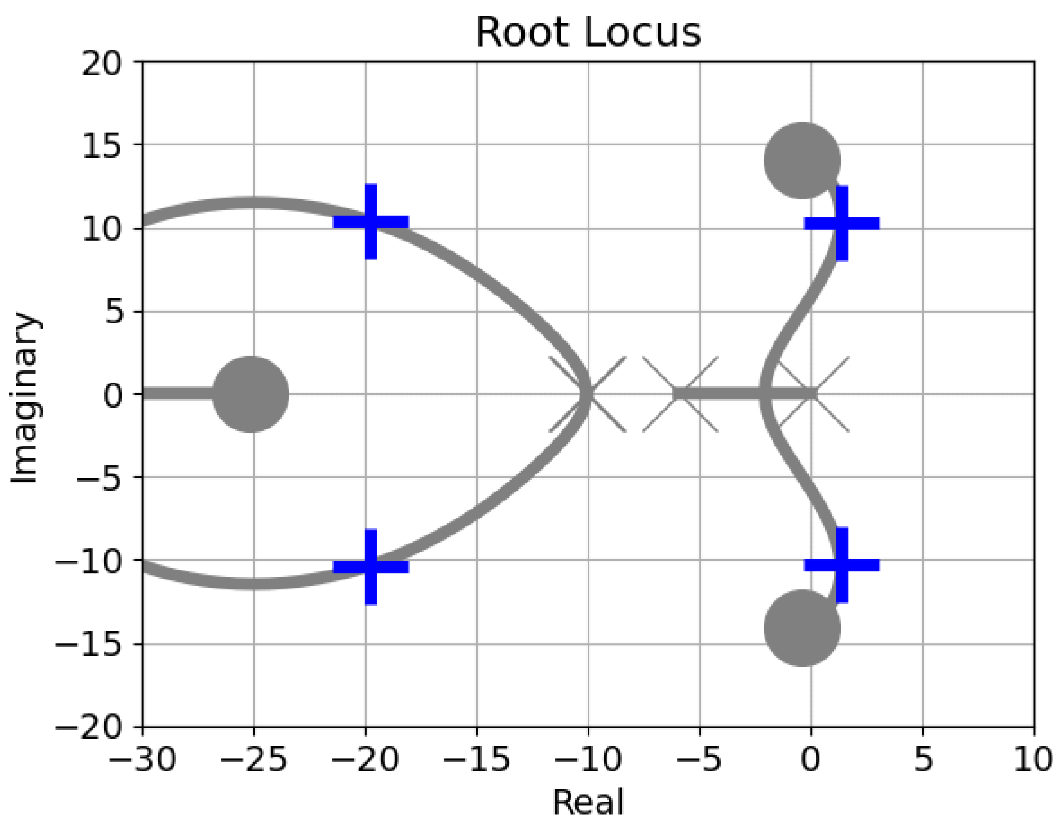
[89]:
```
<matplotlib.legend.Legend at 0x7f6d63ab75b0>
```



8.8 Root Locus

[90]:
```
if SystemName in ['Selkov','Selkov1','Selkov3']:
    xlim=(-6,15)
    ylim=(-10,10)
else:
    xlim=(-30,10)
    ylim=(-20,20)
SetPlot(RL=True)
roots1,gains1 = con.root_locus(L,kvect=[1],plot=False)
roots,gains=con.root_locus(L,xlim=xlim,ylim=ylim,grid=False)
plt.plot(np.real(roots1),np.imag(roots1),color='b', marker='+',mew=5)
plt.grid()
SaveFig(SysName,'SplitRootLocus',RL=True)
SetPlot()
```

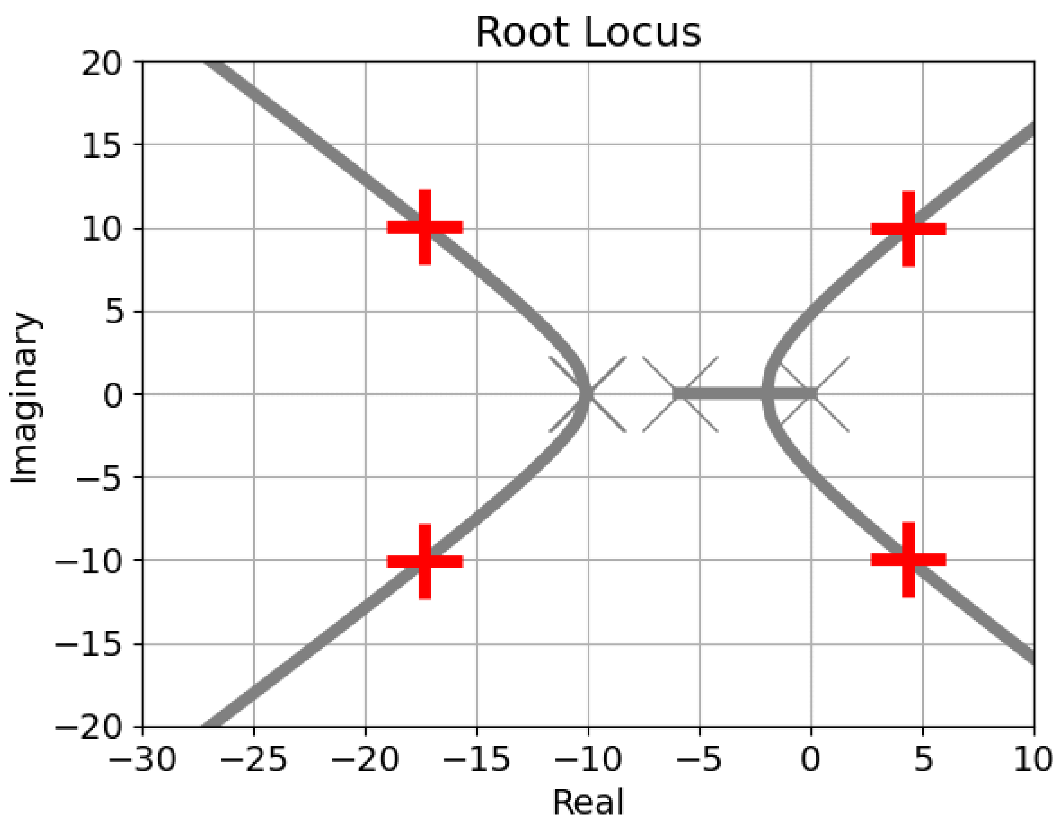
[90]:
```
[<matplotlib.lines.Line2D at 0x7f6d636c8910>,
 <matplotlib.lines.Line2D at 0x7f6d636c8940>,
 <matplotlib.lines.Line2D at 0x7f6d636c8a30>,
 <matplotlib.lines.Line2D at 0x7f6d636c8b20>]
```



8.9 Root Locus - active only

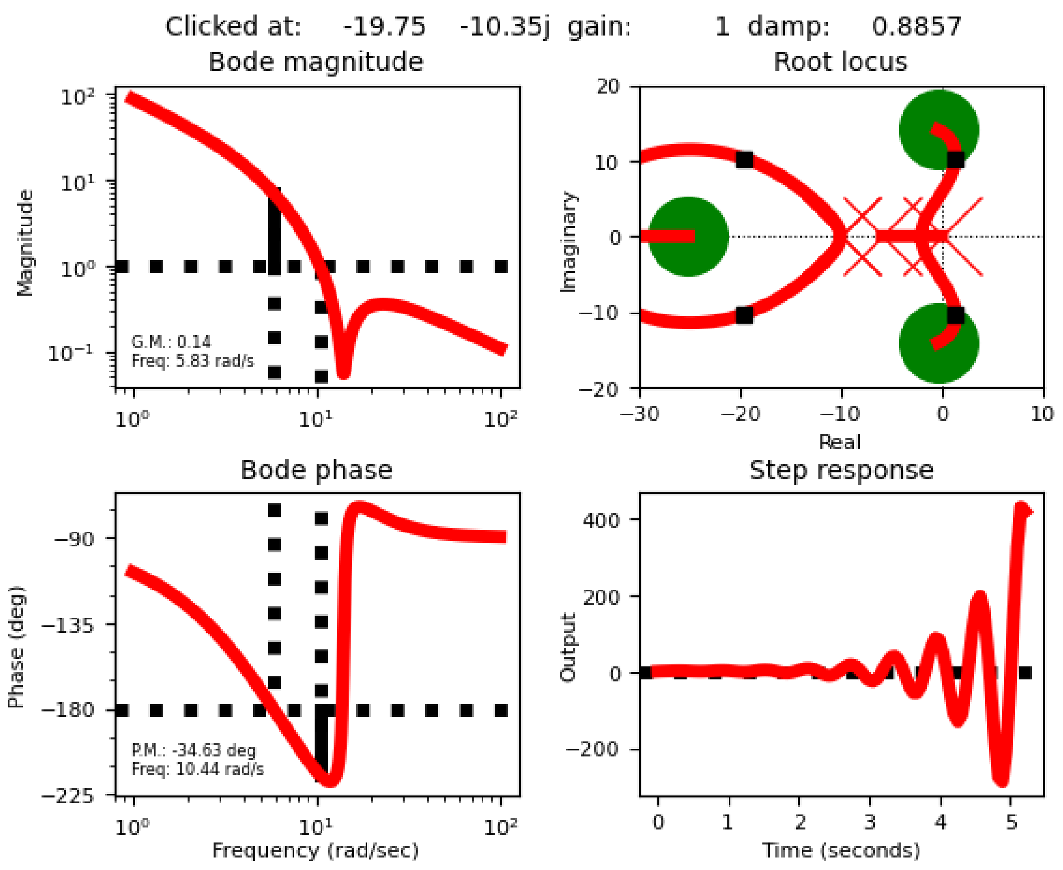
```
[91]: ## RL for L_act
      # if SystemName in ['Selkov','Selkov1','Selkov3']:
      #     xlim=(-10,20)
      #     ylim=(-10,10)
      # else:
      #     xlim=(-30,20)
      #     ylim=(-20,20)
      SetPlot(RL=True)
      roots1,gains1 = con.root_locus(L_act,kvect=[1],plot=False)
      roots,gains=con.root_locus(L_act,xlim=xlim,ylim=ylim,grid=False)
      plt.plot(np.real(roots1),np.imag(roots1),color='r', marker='+',mew=5)
      plt.grid()
      SaveFig(SysName,'SplitRootLocus_act')
      SetPlot()
```

```
[91]: [<matplotlib.lines.Line2D at 0x7f6d6365f100>,
       <matplotlib.lines.Line2D at 0x7f6d6365fa90>,
       <matplotlib.lines.Line2D at 0x7f6d6365f970>,
       <matplotlib.lines.Line2D at 0x7f6d6365fc70>]
```



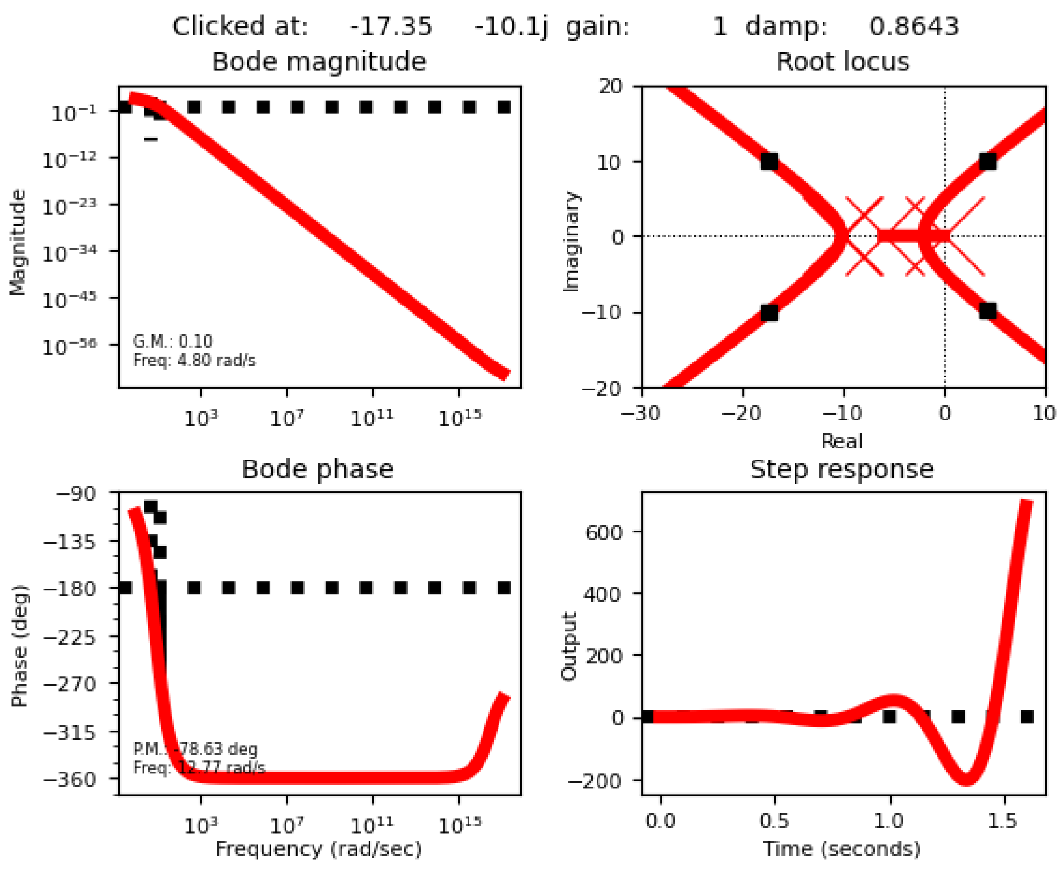
8.10 Sisotool

[92]:
```
con.sisotool(L,xlim_rlocus=xlim,ylim_rlocus=ylim)
SaveFig(SysName,'SplitSisoTool')
```



8.11 Sisotool - active only

[93]:
```
con.sisotool(L_act,xlim_rlocus=xlim,ylim_rlocus=ylim)
SaveFig(SysName,'SplitSisoTool')
```

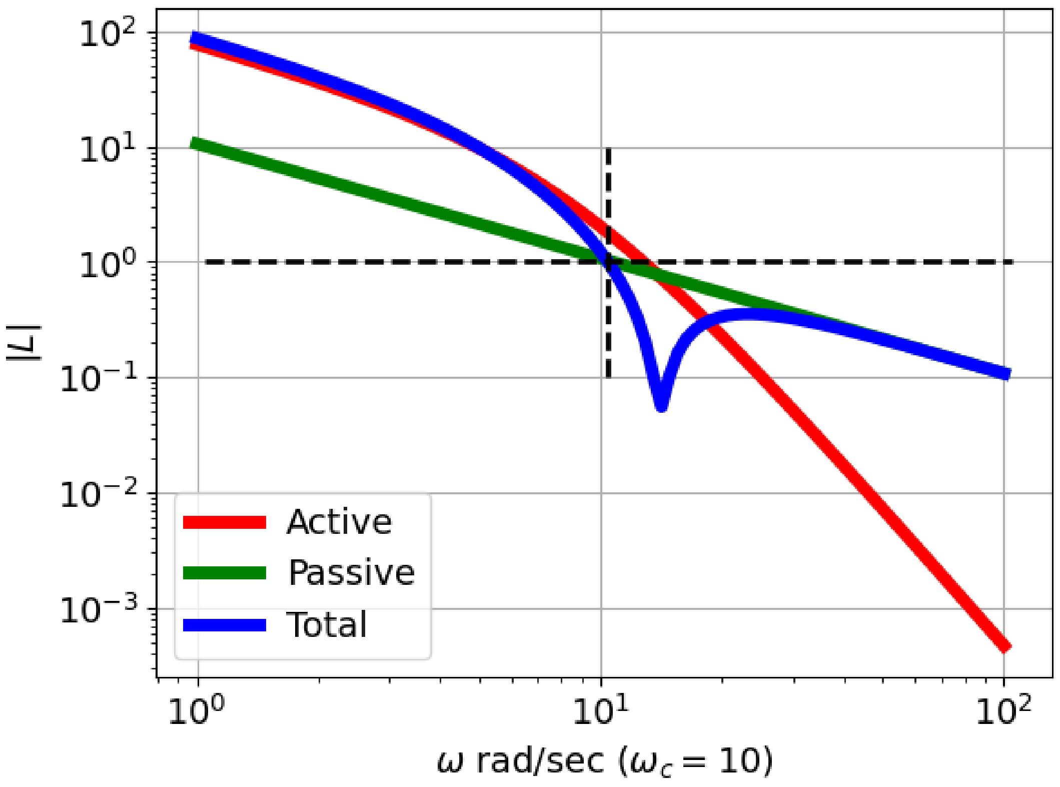


```
[94]: Name = ['Active','Passive','Total']
      for i,l in enumerate(L_list):
      #     print(i,l)
          mag,phase,om = con.bode_plot(l,omega,plot=False)
          plt.loglog(omega,mag,label=Name[i])
      plt.hlines(1,wcp/10,wcp*10,ls='dashed',color='black',lw=2)
      plt.vlines(wcp,0.1,10,ls='dashed',color='black',lw=2)
      plt.legend(loc='lower left')
      plt.grid()
      plt.xlabel(f'$\\omega$ rad/sec ($\\omega_c = {int(round(wcp))}$)')
      plt.ylabel(r'$|L|$')
      SaveFig(SysName,'SplitBodeMag')
```

```
[94]: [<matplotlib.lines.Line2D at 0x7f6d6238e3d0>]

[94]: [<matplotlib.lines.Line2D at 0x7f6d6238ec10>]

[94]: [<matplotlib.lines.Line2D at 0x7f6d623a91c0>]

[94]: <matplotlib.collections.LineCollection at 0x7f6d6277ebe0>

[94]: <matplotlib.collections.LineCollection at 0x7f6d627b3430>

[94]: <matplotlib.legend.Legend at 0x7f6d62b81640>
```

```
[94]: Text(0.5, 0, '$\\omega$ rad/sec ($\\omega_c = 10$)')
```

```
[94]: Text(0, 0.5, '$|L|$')
```



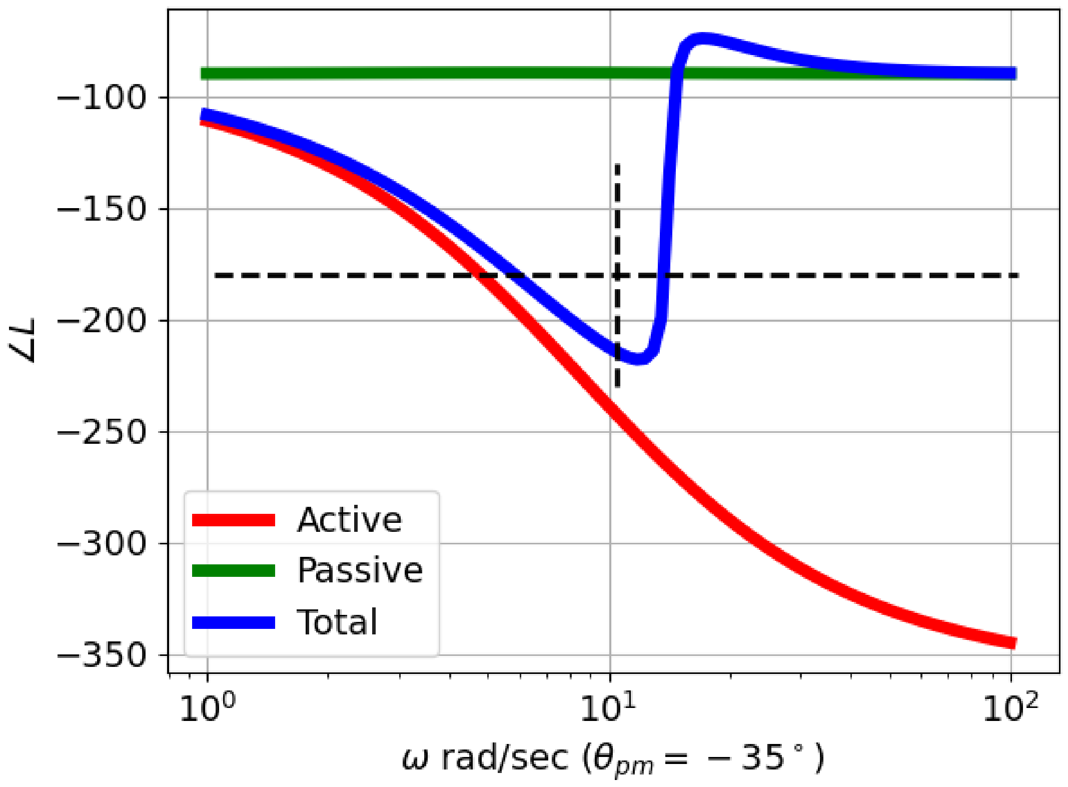
```
[95]: for i,l in enumerate(L_list):
      #     print(i,l)
          mag,phase,om = con.bode_plot(l,omega,plot=False)
          phase_deg = phase*180/np.pi
          plt.semilogx(omega,phase_deg,label=Name[i])
      plt.legend()
      plt.hlines(-180,wcp/10,wcp*10,ls='dashed',color='black',lw=2)
      plt.vlines(wcp,-230,-130,ls='dashed',color='black',lw=2)

      # plt.xlabel(r'$\omega$')
      plt.xlabel(f'$\\omega$ rad/sec ($\\theta_{{pm}} = {int(round(pm))}^\circ$)')

      plt.ylabel(r'$\angle{L}$')
      plt.grid()
      SaveFig(SysName,'SplitBodePha')
```

```
[95]: [<matplotlib.lines.Line2D at 0x7f6d61e734f0>]
```

```
[95]: [<matplotlib.lines.Line2D at 0x7f6d61e739d0>]
```

```
[95]: [<matplotlib.lines.Line2D at 0x7f6d61e731f0>]
```

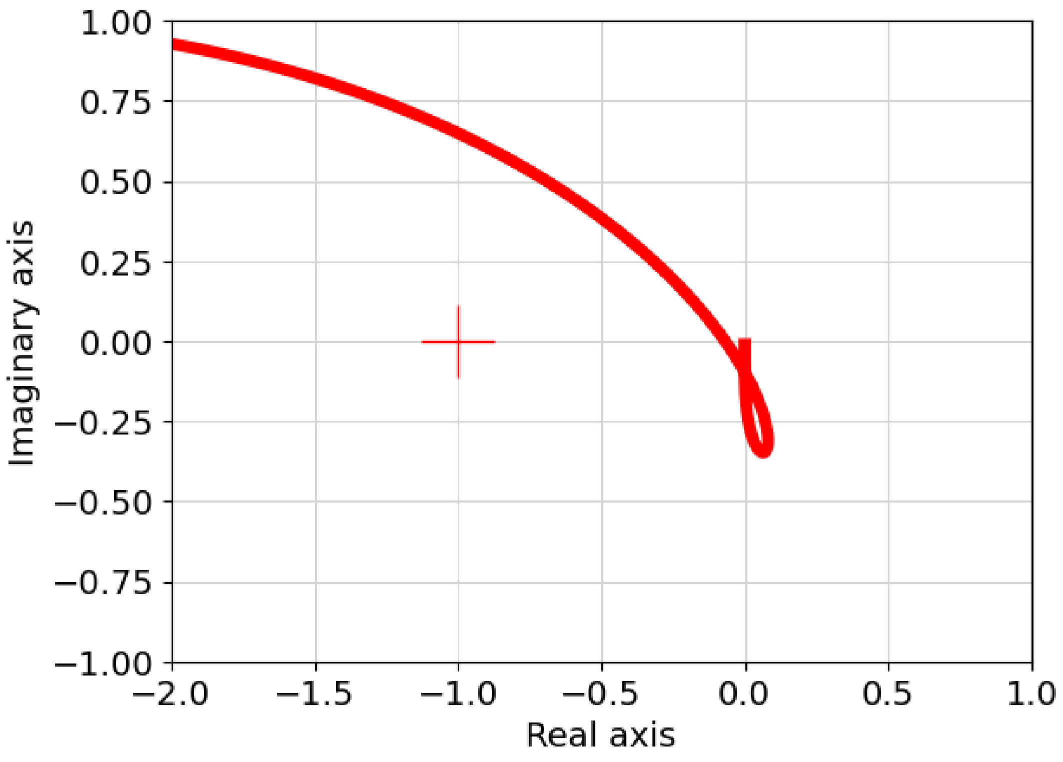
```
[95]: <matplotlib.legend.Legend at 0x7f6d61bd20a0>

[95]: <matplotlib.collections.LineCollection at 0x7f6d61e73250>

[95]: <matplotlib.collections.LineCollection at 0x7f6d61e810d0>

[95]: Text(0.5, 0, '$\\omega$ rad/sec ($\\theta_{pm} = -35^\\circ$)')

[95]: Text(0, 0.5, '$\\angle{L}$')
```



```
[ ]:
```

```
[96]: con.nyquist_plot([LL],mirror_style=False)
      plt.xlim(-2,1)
      plt.ylim(-1,1)
```

```
[96]: 2

[96]: (-2.0, 1.0)

[96]: (-1.0, 1.0)
```

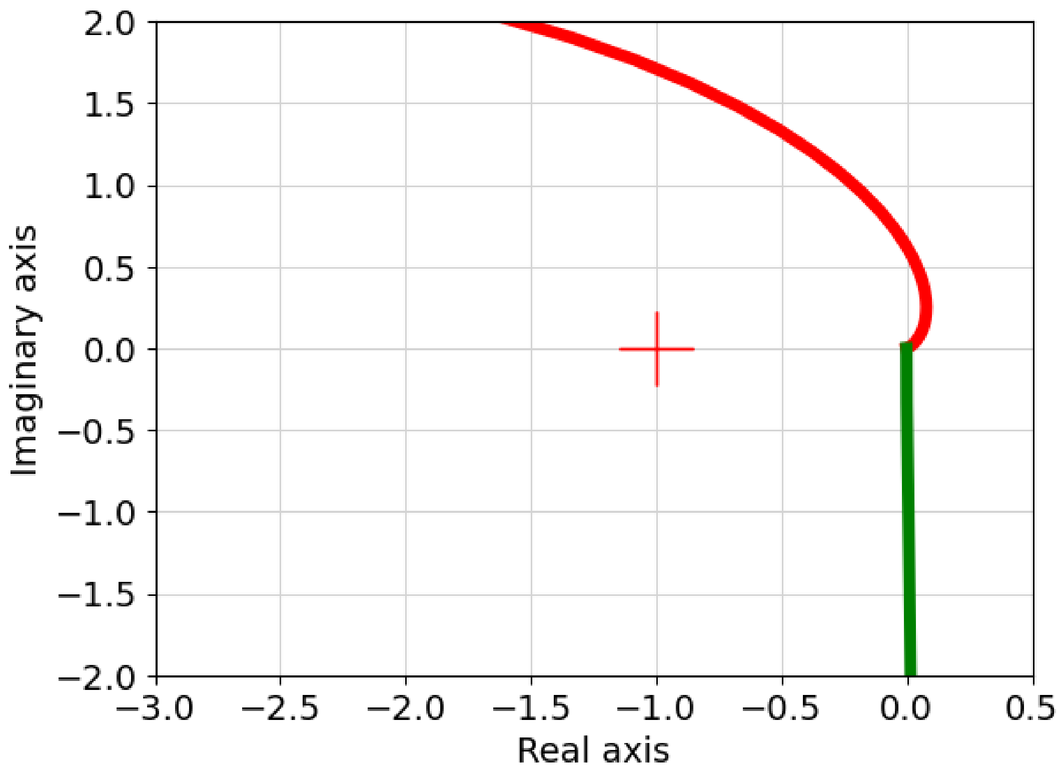


```
[97]: con.nyquist_plot([L_act,L_pas],mirror_style=False)
      plt.xlim(-3,0.5)
      plt.ylim(-2,2)
```

```
[97]: [2, 0]
```

```
[97]: (-3.0, 0.5)
```

```
[97]: (-2.0, 2.0)
```



9 Open-loop analysis of dynamic part of Toy and Goodwin examples

[98]:
```python
if SystemName in ['Toy','Goodwin']:

    # Stoichiometry
    chemostats,parameter,InpVar,OutpVar,T,T_long,n_red =␣
↪SetAll(SystemName,Config='Dynamic',quiet=False)
    s,sc = stoichiometry(sys_abg.model(),chemostats=chemostats)


    species = s['species']

    ## Linearise
    X_ss = copy.deepcopy(x_sl_ss)
    TF,Sys = Lin(s,sc,parameter=parameter,x_ss=X_ss,outvar='dX',Inp=['E1'],␣
↪Outp=['P','Pf'])

    ## Steady state values of the 3 enzymes - and product.
    x_ss_E = 1
    for i in range(3):
        Ei = 'E'+str(i+1)
        x_ss_i = X_ss[species.index(Ei)]
        print(f'{Ei} steady state: {x_ss_i:.3f}')
        x_ss_E *= x_ss_i

    print(f'P steady-state: {X_ss[species.index("P")]:.3f}')
    ## Extract transfer-functions


    for tfName in ['E1_P','E1_Pf']:
        print(tfName)
        tf = TF[tfName]
        OL_d_0 = tf
        # print(OL_d_0)
        OL_d = IntegrateTF(OL_d_0)
        print(OL_d)
        print(con.poles(OL_d))
        g = con.dcgain(OL_d)
        print(f'Gain: {g:.2e} ({g/x_ss_E:.3e})')
```

```
Setting feedback loop with configuration Dynamic
1 states have been removed from the model
3 states have been removed from the model
E1 steady state: 0.168
E2 steady state: 0.283
E3 steady state: 0.794
P steady-state: 5.936
E1_P

             2.819e+05
-----------------------------------
(s + 10.03) (s + 10.08) (s + 10.63)

[-10.63108265+0.j -10.08007675+0.j -10.02837804+0.j]
Gain: 2.62e+02 (6.927e+03)
E1_Pf

  -5.936
----------
s + 0.1685

[-0.16845782+0.j]
Gain: -3.52e+01 (-9.305e+02)
```

[99]:
```
con.config.defaults['xferfcn.display_format'] = 'zpk'
```

10 Linear + saturation

[100]:
```
def linpos_fun(tt,x):
    global _A_MATRIX_
    global _MIN_STATE_
    dx = _A_MATRIX_@x
    # if x[0]<-10:
```

```
    #     dx[0] = 0
    # print(dx.shape)

    for i,xx in enumerate(x):
        min = _MIN_STATE_[i]
        if xx<min:
            # print(i)
            x[i] = min
    return dx

def linpos(A,x0,x_ss_0,t_span):
    global _A_MATRIX_
    global _MIN_STATE_
    _A_MATRIX_ = A
    _MIN_STATE_ = -1*np.array(x_ss_0)
    ret = integrate.solve_ivp(linpos_fun, t_span, x0, max_step=0.01)
    t = ret['t']
    x = ret['y']
    dx = _A_MATRIX_@x

    return t,x.T,dx.T
```

[101]:
```
## Extract variable states
x_ss_0 = []
species_0 = []
for spec in species_open:
    if spec not in chemostats_open:
        species_0.append(spec)
## con.feedback puts P state first - so prepend
species_0 = ['P'] + species_0

for spec in species_0:
    x = x_ss_open[species.index(spec)]
    print(spec,x)
    x_ss_0.append(x)
print(x_ss_0)
print(species_0)
```

```
P 5.93620393877866
E1 0.16845782427848063
E2 0.28297834855767506
E3 0.7944071039555537
[5.93620393877866, 0.16845782427848063, 0.28297834855767506, 0.7944071039555537]
['P', 'E1', 'E2', 'E3']
```

[102]:
```
if SystemName in ['Toy']:
    print(x_ss_0)
    timespan = [0,10]
    x0 = np.zeros(len(x_ss_0))
    # Pert = 1e-2
    x0[0] = pert
```

```
Intsys = con.ss(0,1,1,0)
con.tf(Intsys)
linsys = con.feedback(Intsys,L0_sys)

## Show systems
L0_sys
Intsys
linsys
linsys.state_labels
A = copy.copy(linsys.A)
A.shape
x0.shape

tt,x,dx = linpos(A,x0,x_ss_0,timespan)
X = x + x_ss_0

print(x[1,3])
plt.plot(tt,X)

plt.grid()
plt.legend(species_0)
plt.show()

plt.plot(tt,x[:,0],label='sim')
plt.plot(T,y_cll*pert,label='impulse',lw=2)

# plt.xlim(0,2)
plt.legend()
plt.show()

plt.plot(X[:,1],X[:,0])

plt.grid()
```

```
[5.93620393877866, 0.16845782427848063, 0.28297834855767506, 0.7944071039555537]
```

[102]:

$$\frac{1}{s}$$

[102]:

$$\left(\begin{array}{ccc|c} -5.84 & -0.01 & -0.0169 & -16.8 \\ -33.6 & -10.1 & 0.133 & -0.0494 \\ 0 & -5.61 & -10.1 & 0.0669 \\ \hline -0 & -0 & -14.9 & 10.8 \end{array}\right)$$

[102]:

$$\left(\begin{array}{c|c} 0 & 1 \\ \hline 1 & 0 \end{array}\right)$$

[102]:

$$\left(\begin{array}{cccc|c}
-10.8 & 0 & 0 & 14.9 & 1 \\
-16.8 & -5.84 & -0.01 & -0.0169 & 0 \\
-0.0494 & -33.6 & -10.1 & 0.133 & 0 \\
0.0669 & 0 & -5.61 & -10.1 & 0 \\
\hline
1 & 0 & 0 & 0 & 0
\end{array}\right)$$

```
[102]: ['sys[227]_x[0]', 'sys[82]_x[0]', 'sys[82]_x[1]', 'sys[82]_x[2]']
```

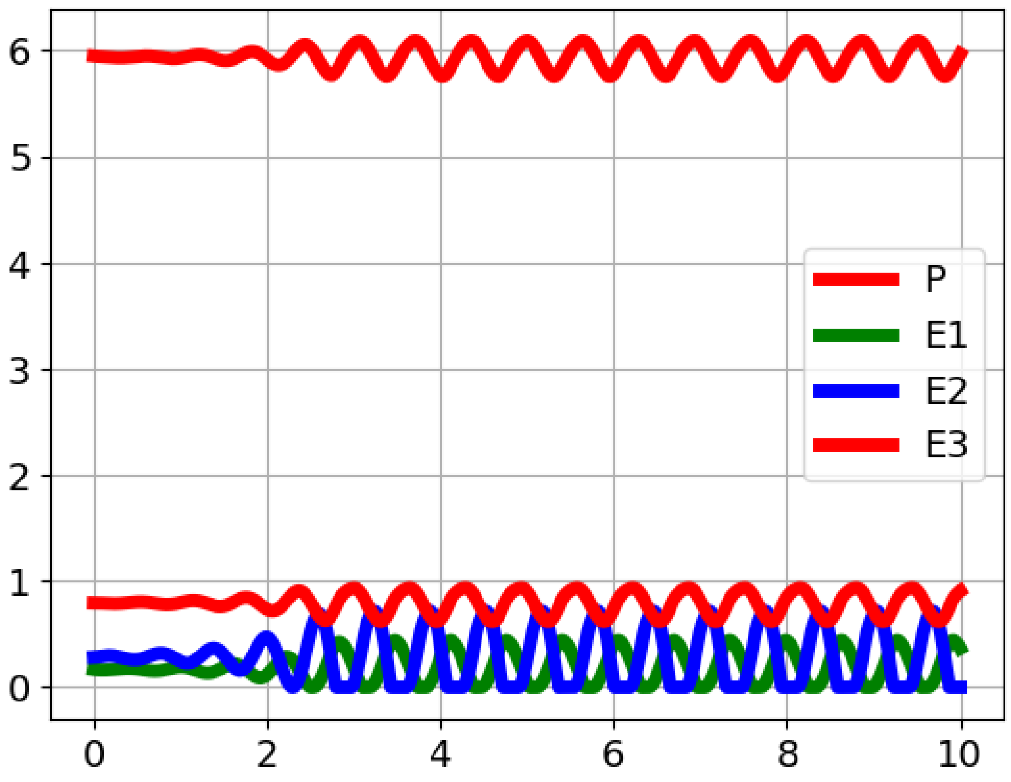
```
[102]: (4, 4)
```

```
[102]: (4,)
```

```
2.657990146241061e-06
```

```
[102]: [<matplotlib.lines.Line2D at 0x7f6d64fa4970>,
        <matplotlib.lines.Line2D at 0x7f6d64fa4c10>,
        <matplotlib.lines.Line2D at 0x7f6d64fa4fa0>,
        <matplotlib.lines.Line2D at 0x7f6d617a7c70>]
```

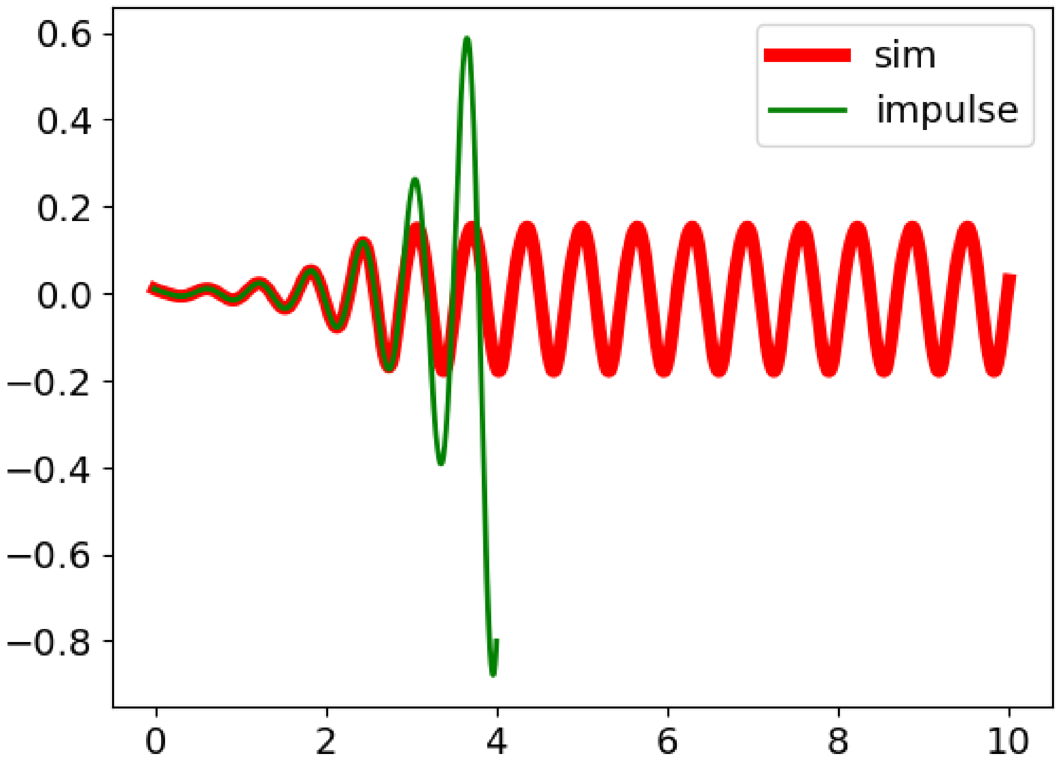
```
[102]: <matplotlib.legend.Legend at 0x7f6d617b2340>
```



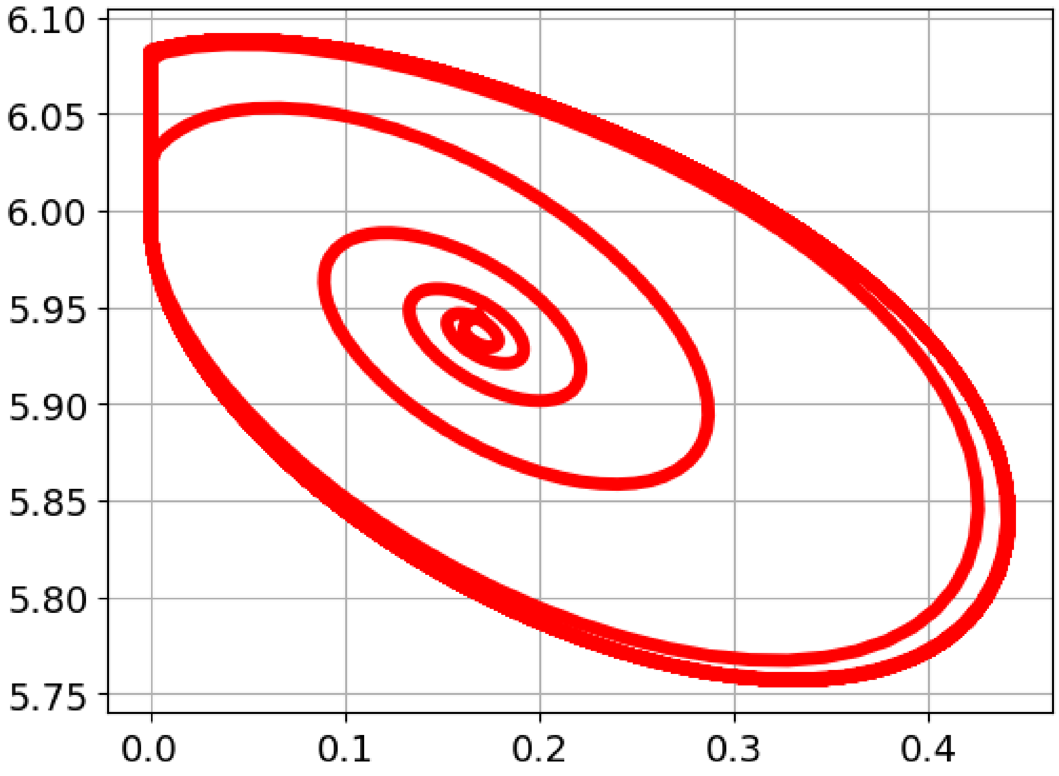
```
[102]: [<matplotlib.lines.Line2D at 0x7f6d616e25b0>]
```

```
[102]: [<matplotlib.lines.Line2D at 0x7f6d616e25e0>]
```

```
[102]: <matplotlib.legend.Legend at 0x7f6d61776190>
```


0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
2
4
6
8
10
sim
impulse

```
[102]: [<matplotlib.lines.Line2D at 0x7f6d6170f430>]
```


6.10
6.05
6.00
5.95
5.90
5.85
5.80
5.75
0.0
0.1
0.2
0.3
0.4

10.1 Signals at integrator

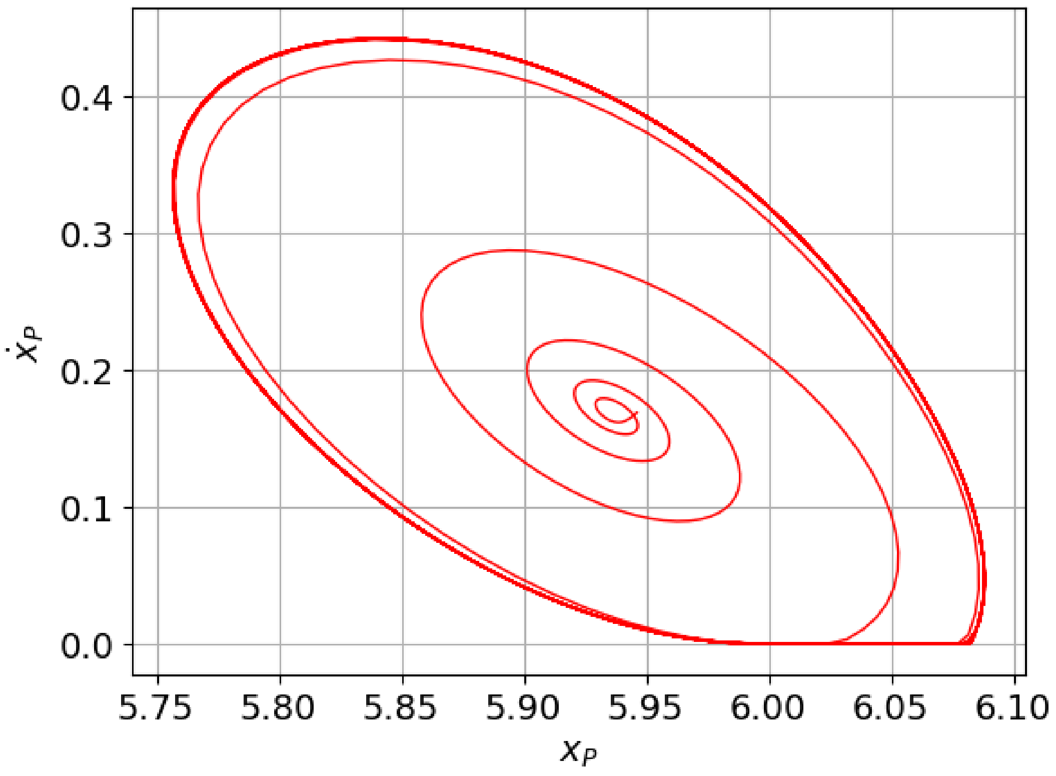
```
[103]: if SystemName in ['Toy']:

           print(species)
           i_P = 0 # See above
           i_E1 = 1
           x_X = X[:,i_P]
           x_Y = X[:,i_E1]
           plt.plot(x_X,x_Y,lw=1)
           plt.xlabel('$x_P$')
           plt.ylabel('$\dot{x}_P$')
           # plt.xlim(left=0)
           # plt.ylim(bottom=0)
           plt.grid()

           SaveFig(SysName,'PhasePlaneP')
```

```
['fb_Act', 'fb_E0', 'E1', 'E2', 'E3', 'P', 'Pf', 'decr1_A', 'decr1_Zf',
'decr2_A', 'decr2_Zf', 'decr3_A', 'decr3_Zf']
```

```
[103]: [<matplotlib.lines.Line2D at 0x7f6d61e5d490>]
```

```
[103]: Text(0.5, 0, '$x_P$')
```

```
[103]: Text(0, 0.5, '$\\dot{x}_P$')
```



[104]:
```python
## Optionally save data
print(SysName)
if SaveData:
    file = open(f'{SysName}.dat', 'wb')
    pickle.dump(SavedData, file)
    file.close()
```

```
Toy
```

References