Physically-Plausible Parameters

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

February 5, 2021

Contents

1 Introduction

This note illustrates an approach to fitting the parameters of a bond graph model to experimental data. Insofar as the parameters are associated with a bond graph, they are *physically-plausible* Gawthrop et al. (2020).

The approach uses a bond-graph derived from a stoichiometric model of *e.coli* Orth et al. (2010) (using a method described elsewhere Gawthrop (2020)) combined with experimental values of *reaction potential*, *reaction flux* and *species concentration* from the literature Park et al. (2016).

1.1 Setup modules

[1]:
```
## Paths
NeedPath=True
if NeedPath:
    import sys
    sys.path += ['/usr/lib/python3/dist-packages']
```

[2]:
```
## Maths library
import numpy as np
import scipy

## BG tools
import BondGraphTools as bgt

## SVG bond graph
import svgBondGraph as sbg

## BG stoichiometric utilities
import stoich as st

## Modular bond graphs
import modularBondGraph as mbg

## Stoichiometric conversion
import CobraExtract as Extract
import stoichBondGraph as stbg

## Potentials
import phiData

## Faraday constant
import scipy.constants as con
F = con.physical_constants['Faraday constant'][0]

## Display
import IPython.display as disp

## Plotting
import matplotlib.pyplot as plt


import copy

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

import importlib as imp

quiet = True
```

```
showMu=True
```

1.2 Quadratic programming QP.

$$\text{minimise } \frac{1}{2}x^T P x + q^T x \tag{1}$$
$$\text{subject to } Gx \leq h \tag{2}$$
$$\text{and } Ax = b \tag{3}$$

In the case considered here, there is no equality constraint and

$$x = \hat{\phi} \tag{4}$$
$$P = NN^T + \mu I_{n_X \times n_X} \tag{5}$$
$$q = (N\Phi)^T \tag{6}$$
$$G = N^T \tag{7}$$
$$h = -\Phi_{min} \tag{8}$$

$\mu > 0$ is required to give a convex QP: in essence it turns a non-unique solution for ϕ into a minimum norm solution.

[3]:
```
## Quadratic programming stuff.
import quadprog

## Function from https://scaron.info/blog/quadratic-programming-in-python.html
def quadprog_solve_qp(P, q, G=None, h=None, A=None, b=None):
    qp_G = .5 * (P + P.T)    # make sure P is symmetric
    qp_a = -q
    if A is not None:
        qp_C = -numpy.vstack([A, G]).T
        qp_b = -numpy.hstack([b, h])
        meq = A.shape[0]
    else:  # no equality constraint
        qp_C = -G.T
        qp_b = -h
        meq = 0
    return quadprog.solve_qp(qp_G, qp_a, qp_C, qp_b, meq)[0]

## Function to compute phi from Phi subject to Phi>positive number
## NN Reduced N corresponding to known Phi
def quadsolve_phi(N0,N1,Phi0,Phi_min=0.0,mu=1e-10):

    (n_X,n_V) = N1.shape
    P = 1.0*N0@(N0.T) + mu*np.eye(n_X)
    q = (N0@Phi0).T
    G = 1.0*N1.T
    h = -Phi_min*np.ones((n_V))
    phi = quadprog_solve_qp(P, q, G=G, h=h)

    return phi
```

2 Conversion factor

[4]:
```
Factor = st.F()/1e6
print(f'To convert from kJ/mol to mV, divide by {1/Factor:4.3}')
```

```
To convert from kJ/mol to mV, divide by 10.4
```

3 Extract Model

This example uses the Glycolysis and Pentose Phosphate pathways.

Notes:

- Reactions RPI, PGK and PGM are reversed to correspond to positive flows.
- The resultant stoichiometric matrix N relates reaction flows (f) to species flows ($\dot{x}$):

$$\dot{x} = Nf \tag{9}$$

3.1 Extract stoichiometry

[5]:
```
sm = Extract.extract(cobraname='textbook',Remove=['_C','__' ],
negReaction=['RPI','PGK','PGM'], quiet=quiet)
```

```
Extracting stoichiometric matrix from: textbook
Cobra Model name: e_coli_core BondGraphTools name: e_coli_core_abg
Extract.Integer only handles one non-integer per reaction
Multiplying reaction BIOMASS_ECOLIORE ( 12 ) by 0.6684491978609626 to avoid non-
integer species 3PG ( 2 )
Multiplying reaction CYTBD ( 15 ) by 2.0 to avoid non-integer species O2 ( 55 )
Multiplying reaction PGK ( 54 ) by -1
Multiplying reaction PGM ( 56 ) by -1
Multiplying reaction RPI ( 65 ) by -1
```

[6]:
```
name = 'GlyPPP_abg'
reaction = []

## Glycolysis
reaction += ['PGI','PFK','FBA','TPI']

## Pentose Phosphate
reaction += ['G6PDH2R','PGL','GND','RPI','TKT2','TALA','TKT1','RPE']

ss = Extract.choose(sm,reaction=reaction)

## Create BG
ss['name'] = name
stbg.model(ss)
import GlyPPP_abg
imp.reload(GlyPPP_abg)
s = st.stoich(GlyPPP_abg.model(),quiet=quiet)
```

[6]:
```
<module 'GlyPPP_abg' from
'/home/peterg/WORK/Research/SystemsBiology/Notes/2021/Parameter/GlyPPP_abg.
↪py'>
```

[7]:
```python
## Set up chemostats
chemostats = ['ADP','ATP','H','H2O','NADP','NADPH','CO2']
chemostats += ['G6P','G3P','R5P']
#chemostats += ['G6P','R5P']
chemostats.sort()
print(chemostats)
sc = st.statify(s,chemostats=chemostats)

sp = st.path(s,sc,pathname='PPP')
print(st.sprintp(sc))
disp.Latex(st.sprintrl(sp,chemformula=True))
```

```
['ADP', 'ATP', 'CO2', 'G3P', 'G6P', 'H', 'H2O', 'NADP', 'NADPH', 'R5P']
3 pathways
0:  + PGI + PFK + FBA + TPI
1:  + G6PDH2R + PGL + GND + RPI
2:  - 2 PGI + 2 G6PDH2R + 2 PGL + 2 GND + TKT2 + TALA + TKT1 + 2 RPE
```

[7]:

$$ATP + G_6P \xrightleftharpoons{PPP_1} ADP + 2\,G_3P + H \tag{10}$$

$$G_6P + H_2O + 2\,NADP \xrightleftharpoons{PPP_2} CO_2 + 2\,H + 2\,NADPH + R_5P \tag{11}$$

$$2\,H_2O + 4\,NADP + R_5P \xrightleftharpoons{PPP_3} 2\,CO_2 + G_3P + 4\,H + 4\,NADPH \tag{12}$$

[8]:
```python
print(st.sprintl(sc,'K',transpose=True))
disp.Latex(st.sprintl(sc,'K',transpose=True))
```

```
\begin{align}
K^T &=
\left(\begin{array}{cccccccccccc}1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &
0\\0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0\\-2 & 0 & 0 & 0 & 2 & 2 & 2 & 0
& 1 & 1 & 1 & 2\end{array}\right)
\end{align}
```

[8]:

$$K^T = \left(\begin{array}{cccccccccccc}1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0\\-2 & 0 & 0 & 0 & 2 & 2 & 2 & 0 & 1 & 1 & 1 & 2\end{array}\right) \tag{13}$$

3.2 Extract reaction potentials Φ and deduce plausible species potentials ϕ.

Because of the energetic constaints implied by the bond graph, the reaction potentials Φ are related to the species potentials ϕ by

$$\Phi = -N^T\phi \tag{14}$$

Typically, there are more species than reactions and so N has more rows than columns. Given the reaction potentials Φ, the species potentials can be estimated using the *pseudo inverse* $N^\dagger$ of $-N^T$:

$$\hat{\phi} = N^\dagger \Phi \tag{15}$$

Notes:

- In general $\hat{\phi} \neq \phi$ but is physically plausible insofar as $-N^T\hat{\phi} = \Phi$.

[9]:
```python
def getPhi(s,Phi_hyd=0.5,phi_6PGL=None,quadprog=False):
    """Extract phi for given system using
    Reaction potentials from ParRubXu16"""

    ## Reaction potentials from ParRubXu16
    PHI = phiData.Phi_ParRubXu16_Measured()

#     Phenotype = 'Mammalian'
#     Phenotype = 'Yeast'
    Phenotype = 'Ecoli'
    Phi_reac = PHI[Phenotype]

    Phi = np.zeros((len(s['reaction']),1))
    N = copy.copy(s['N'])
    N_0 = None
    N_1 = None
    Phi_0 = []
    for j,reac in enumerate(s['reaction']):
        if (reac in Phi_reac.keys()) and not np.isnan(Phi_reac[reac]):
            Phi_0.append(Phi_reac[reac])
            if N_0 is None:
                N_0 = N[:,j]
            else:
                N_0 = np.vstack((N_0,N[:,j]))
        else:
            if N_1 is None:
                N_1 = N[:,j]
            else:
                N_1 = np.vstack((N_1,N[:,j]))

    Phi_0 = np.array(Phi_0)
    #print(N_1)

    ## Compute Phi
    N_0 = N_0.T
    N_1 = N_1.T

    n_X,n_V = N_0.shape
    print(f'Extracting {n_X} values of phi from {n_V} values of Phi')


    if quadprog:
        phi = quadsolve_phi(N_0,N_1,Phi_0,Phi_min=1e-3,mu=1e-10)
    else:
        ## Compute Phi using pseudo inverse
        pinvNT =  scipy.linalg.pinv(N_0.T)
        phi = -pinvNT@Phi_0

    if phi_6PGL is not None:
        ## Reset 6PGL
        i_6PGL = s['species'].index('6PGL')
        phi[i_6PGL] = phi_6PGL
        print (f'Resetting phi_6GPL to {int(1e3*phi[i_6PGL])} mV' )

    ## Sanity check
    Phi_new = -N_0.T@phi
    err = np.linalg.norm(Phi_new-Phi_0)
    print(f'Phi error = {int(err*1000)}mV\n')

    Phi = -N.T@phi

    return Phi,phi,Phi_0,Phi_reac
```

[10]:
```
Phi_,phi_est_,Phi_0_,Phi_reac_ = getPhi(s,quadprog=False)
print('Minimum Phi = ', int(round(np.min(1e3*Phi_))), 'mV')
```

```
Extracting 19 values of phi from 10 values of Phi
Phi error = 0mV

Minimum Phi =  -3 mV
```

[11]:
```
Phi,phi_est,Phi_0,Phi_reac = getPhi(s,quadprog=True)
print('Minimum Phi = ', int(round(np.min(1e3*Phi))), 'mV')

print('\nChange in phi')
for i,spec in enumerate(s['species']):
    change = int(1e3*(phi_est[i]-phi_est_[i]))
    if not change==0:
        print(f'{i} {spec}\t {change}')

print('\nChange in Phi')
for i,reac in enumerate(s['reaction']):
    change = int(round(1e3*(Phi[i]-Phi_[i])))
    if not change == 0:
        print(f'{i} {reac}\t {change} {int(round(1e3*Phi[i]))}␣
 ↪{int(round(1e3*Phi_[i]))}')
```

```
Extracting 19 values of phi from 10 values of Phi
Phi error = 0mV

Minimum Phi =  0 mV
```

```
Change in phi
1 6PGL    1
12 H2O    1

Change in Phi
5 PGL     4 1 -3
```

3.3 Extracted reactions and reaction potentials

[12]:
```
disp.Latex(st.sprintrl(s,chemformula=True,Phi=Phi,units=['mV','kJ']␣
 ↪,showMu=showMu))
```

[12]:

$$
\begin{aligned}
\mathrm{G_6P} &\xrightleftharpoons{\mathrm{PGI}} \mathrm{F_6P} && (16\ \mathrm{mV})\ [-1.59\ \mathrm{kJ\ mol^{-1}}]\\
\mathrm{ATP + F_6P} &\xrightleftharpoons{\mathrm{PFK}} \mathrm{ADP + FDP + H} && (68\ \mathrm{mV})\ [-6.64\ \mathrm{kJ\ mol^{-1}}]\\
\mathrm{FDP} &\xrightleftharpoons{\mathrm{FBA}} \mathrm{DHAP + G_3P} && (20\ \mathrm{mV})\ [-1.93\ \mathrm{kJ\ mol^{-1}}]\\
\mathrm{DHAP} &\xrightleftharpoons{\mathrm{TPI}} \mathrm{G_3P} && (7\ \mathrm{mV})\ [-0.77\ \mathrm{kJ\ mol^{-1}}]\\
\mathrm{G_6P + NADP} &\xrightleftharpoons{\mathrm{G_6PDH_2R}} \mathrm{{}_6PGL + H + NADPH} && (82\ \mathrm{mV})\ [-7.99\ \mathrm{kJ\ mol^{-1}}]\\
\mathrm{{}_6PGL + H_2O} &\xrightleftharpoons{\mathrm{PGL}} \mathrm{{}_6PGC + H} && (1\ \mathrm{mV})\ [-0.10\ \mathrm{kJ\ mol^{-1}}]\\
\mathrm{{}_6PGC + NADP} &\xrightleftharpoons{\mathrm{GND}} \mathrm{CO_2 + NADPH + RU_5PD} && (114\ \mathrm{mV})\ [-11.05\ \mathrm{kJ\ mol^{-1}}]\\
\mathrm{RU_5PD} &\xrightleftharpoons{\mathrm{RPI}} \mathrm{R_5P} && (0\ \mathrm{mV})\ [-0.00\ \mathrm{kJ\ mol^{-1}}]\\
\mathrm{E_4P + XU_5PD} &\xrightleftharpoons{\mathrm{TKT_2}} \mathrm{F_6P + G_3P} && (16\ \mathrm{mV})\ [-1.58\ \mathrm{kJ\ mol^{-1}}]\\
\mathrm{G_3P + S_7P} &\xrightleftharpoons{\mathrm{TALA}} \mathrm{E_4P + F_6P} && (54\ \mathrm{mV})\ [-5.25\ \mathrm{kJ\ mol^{-1}}]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{R_5P + XU_5PD} &\xrightleftharpoons{\mathrm{TKT_1}} \mathrm{G_3P + S_7P} && (4\ \mathrm{mV})\ [-0.39\ \mathrm{kJ\ mol^{-1}}]\\
\mathrm{RU_5PD} &\xrightleftharpoons{\mathrm{RPE}} \mathrm{XU_5PD} && (0\ \mathrm{mV})\ [-0.08\ \mathrm{kJ\ mol^{-1}}]
\end{aligned}
$$

4 Deduce Pathway Flows

From basic stoichiometric analysis, steady-state flows can be written as:

$$f = K_p f_p \tag{16}$$
$$\text{where } K_p N^{cd} = 0 \tag{17}$$

Note that the *pathway matrix* K_p is dependent on the choice of chemostats.

Given a set of experimental flows f, an estimate $\hat{f}_p$ of f_p can be obtained from the *least-squares* formula:

$$(K_p^T K_p)\hat{f}_p = K_p^T f \tag{18}$$

Notes:

- v_p is a n_p vector containg the pathways flows

- $(K_p^T K_p)$ is a square $n_p \times n_p$ matrix where n_p is the number of pathways
- If some flows are not measured, the corresponding rows of K_p are deleted
- the reaction flows (including the missing ones) can be estimated from $\hat{f} = K_p \hat{f}_p$.
- the estimated chemostat flows are given by the non-zero elements of

$$\hat{\dot{x}} = N\hat{f} \tag{19}$$

[13]:
```
def PathwayFlux(K,reaction,Reaction,flux):

    #KK = st.singleRemove(K)
    KK = K
    Kp = None
    Flux = {}
    reac_known = []
    #flux = phiData.ParRubXu16_flux()
    for i,reac in enumerate(reaction):
        if reac in flux.keys():
            reac_known.append(reac)
            fi = flux[reac]
            #Ki = np.abs(KK[i,:])
            Ki = KK[i,:]
            #print(reac,Ki)
            if Kp is None:
                Kp = Ki
                f = fi
            else:
                Kp = np.vstack((Kp,Ki))
                f = np.vstack((f,fi))
    #print(Kp)

    if Kp is not None:
        #print(f)
        f_p = np.linalg.solve(Kp.T@Kp,Kp.T@f)
        for i,Reac in enumerate(Reaction):
            Flux[Reac] = f_p[i][0]
        #print(f_p)
        f_est = Kp@f_p
        #print(Kp@f_p-f)

    error = np.linalg.norm(f_est-f)/len(f)
    print(f'Flux error = {error:.2e}')

    return Flux,f_p,f_est,f,reac_known
```

5 Reaction constants (modified mass-action)

The modified mass-action formula is Gawthrop et al. (2020):

$$f = \kappa \left(\exp \frac{\Phi^f}{\alpha V_N} - \exp \frac{\Phi^r}{\alpha V_N} \right) \tag{20}$$

Thus an estimate for κ can be computed as:

$$\hat{\kappa} = \frac{\hat{f}}{f_0} \tag{21}$$

$$\text{where } f_0 = \left(\exp \frac{\Phi^f}{\alpha V_N} - \exp \frac{\Phi^r}{\alpha V_N} \right) \tag{22}$$

```
[14]: def reactionConstant(s,phi_est,f_est,alpha=1):

          V_N = st.V_N()

          ## Extract stoichiometry
          N = s['N']
          Nf = s['Nf']
          Nr = s['Nr']
          reaction = s['reaction']

          ## Compute Phis from estimated phi
          Phi_ = -N.T@phi_est
          Phi_f = Nf.T@phi_est
          Phi_r = Nr.T@phi_est

          ## Compute normalised flow rates
          f0 = (np.exp(Phi_f/(alpha*V_N)) - np.exp(Phi_r/(alpha*V_N)))

          parameter = {}
          for i,react in enumerate(reaction):
              kap = f_est[i][0]/f0[i]
              parameter[f'kappa_{react}'] = kap
              #print(f'{react}: \tPhi = {int(Phi_[i]*1000)}mV, \tf_est =␣
       ↪{f_est[i][0]:.2e}, \tkappa = {kap:.2}')

          return parameter
```

5.1 Normalise data

```
[15]: imp.reload(phiData)
      ## Extract experimetal data
      concentration = phiData.ParRubXu16_conc() # M
      Flux = phiData.ParRubXu16_flux() # mM/min
      flux = Flux['Ecoli']

      c_0 = concentration['G6P']
      f_0 = flux['PGI']/60
      t_0 = (1000*c_0)/f_0

      print(f'c_0 = {c_0*1000} mM, f_0 = {f_0} mM/sec, t_0 = {t_0} sec')
```

```
[15]: <module 'phiData' from
      '/home/peterg/WORK/Research/SystemsBiology/lib/python/phiData.py'>
```

```
c_0 = 0.675 mM, f_0 = 0.9916666666666667 mM/sec, t_0 = 0.680672268907563 sec
```

5.2 Show computed reaction flows

[16]:
```
K = sc['K']
n_path = K.shape[1]
Reaction = []
for i in range(n_path):
    Reaction += [f'PPP{i+1}']

print(Reaction)

for reac in flux.keys():
    flux[reac] *= 1/f_0

fluxp,f_p,f_est,f,reaction = PathwayFlux(sc['K'],s['reaction'],Reaction,flux)


## Reaction constants
f_est = sc['K']@f_p
parameter = reactionConstant(s,phi_est,f_est)

#f_est = sc['K']@f_p
j=0

print('\n\n%% LaTeX table')
print('\\hline')
print('Reaction &\t $\Phi$ &\t $\hat{\Phi}$ &\t $f$ & $\\hat{f}$ &␣
 ↪$\\hat{\\kappa}$\\\\')
print('\\hline')
for i,reac in enumerate(s['reaction']):
    if reac in flux.keys():
        ff = f'{int(round(f[j][0]))}'
        j += 1
    else:
        ff = '--'
    if reac in Phi_reac.keys():
        PP = f'{int(round(1e3*Phi_reac[reac]))}'
    else:
        PP = '--'
    kappa = 'kappa_'+reac
    print(f'{reac} &\t {PP} &\t {int(round(1e3*Phi[i]))} &\t {ff} &␣
 ↪{int(round(f_est[i][0]))} & {parameter[kappa]:.2} \\\\')
print('\\hline')
```

```
['PPP1', 'PPP2', 'PPP3']
Flux error = 1.86e-01


%% LaTeX table
\hline
```

```
Reaction &       $\Phi$ &        $\hat{\Phi}$ &  $f$ & $\hat{f}$ &
$\hat{\kappa}$\\
\hline
PGI &    16 &    16 &    60 & 60 & 1.5e+02 \\
PFK &    69 &    69 &    63 & 63 & 5.5e+01 \\
FBA &    20 &    20 &    63 & 63 & 1.6e+02 \\
TPI &    8 &     8 &     63 & 63 & 3.5e+02 \\
G6PDH2R &        -- &    83 &    -- & 12 & 4.7 \\
PGL &    -- &    1 &     -- & 12 & 2.9e+02 \\
GND &    115 &   115 &   12 & 12 & 1.3 \\
RPI &    0 &     0 &     8 & 8 & 4.2e+03 \\
TKT2 &   16 &    16 &    1 & 2 & 9.2 \\
TALA &   54 &    54 &    -- & 2 & 1.7 \\
TKT1 &   4 &     4 &     3 & 2 & 8.8 \\
RPE &    1 &     1 &     4 & 4 & 9.6e+01 \\
\hline
```

5.3 Show computed chemostat flows

[17]:
```python
dx_est = s['N']@f_est

print('\n\n%% LaTeX table')
print('\\hline')
print('Chemostat &\t flow \\\\')
print('\\hline')
for i,spec in enumerate(s['species']):
    if spec in chemostats:
        print(f'{spec} &\t {int(round(dx_est[i][0]))} \\\\')
print('\\hline')
```

```
%% LaTeX table
\hline
Chemostat &      flow \\
\hline
ADP &    63 \\
ATP &    -63 \\
CO2 &    12 \\
G3P &    128 \\
G6P &    -71 \\
H &      86 \\
H2O &    -12 \\
NADP &   -23 \\
NADPH &  23 \\
R5P &    6 \\
\hline
```

5.4 Show pathway flows

[18]:
```
print('\n\n%% LaTeX table')
print('\\hline')
print('Pathway &\t $\hat{f}_p$ \\\\')
print('\\hline')
for reac in fluxp.keys():
    print(f'{reac} &\t {int(round(fluxp[reac]))} \\\\')
print('\\hline')
```

```
%% LaTeX table
\hline
Pathway &        $\hat{f}_p$ \\
\hline
PPP1 &   63 \\
PPP2 &   8 \\
PPP3 &   2 \\
\hline
```

6 Species constants

$$K = \frac{\exp \phi}{x^\circ} = \frac{\exp \phi}{V c^\circ} \tag{23}$$

[19]:
```
#imp.reload(phiData)

print('\n\n%% LaTeX table')
print('\\hline')
print('Species &\t $\\hat{\\phi}~mV$ & $c$ & $\\hat{K}$ \\\\')
print('\\hline')


#concentration['H'] = 1e-7

## Data in mM
scale = 1e3
K_spec = np.ones(s['n_X'])
conc = np.ones(s['n_X'])
c_G6P = concentration['G6P']
#print('c_G6P',c_G6P)
for i,spec in enumerate(s['species']):
    if spec in concentration.keys():
        conc[i] = concentration[spec]/c_G6P
        K_spec[i] = np.exp(phi_est[i]/st.V_N())/conc[i]
        print(f'{spec} & {int(round(1e3*phi_est[i]))} &  \t{conc[i]:.2} &␣
 ↪{K_spec[i]:.4} \\\\')
    else:
        K_spec[i] = np.exp(phi_est[i]/st.V_N())
#         print(f'{spec} &{phi_est[i]:.2} & -- & --\\\\')
```

```
print('\\hline')

print(conc)
```

```
%% LaTeX table
\hline
Species &        $\hat{\phi}~mV$ & $c$ & $\hat{K}$ \\
\hline
6PGC & 29 &     0.024 & 122.0 \\
ADP & -27 &     0.84 & 0.4307 \\
ATP & 27 &      6.9 & 0.3981 \\
CO2 & -30 &     1.1e+01 & 0.02853 \\
DHAP & -10 &    2.4 & 0.2861 \\
E4P & -27 &     0.015 & 23.61 \\
F6P & -21 &     0.14 & 3.15 \\
FDP & -8 &      2.3 & 0.3323 \\
G3P & -18 &     0.21 & 2.453 \\
G6P & -5 &      1.0 & 0.8377 \\
NADP & 30 &     0.042 & 73.7 \\
NADPH & -30 &   0.097 & 3.329 \\
R5P & 5 &       0.042 & 29.06 \\
RU5PD & 5 &     0.0078 & 156.8 \\
S7P & 24 &      0.027 & 90.79 \\
XU5PD & 5 &     0.044 & 26.8 \\
\hline
[2.44444444e-02 1.00000000e+00 8.42962963e-01 6.91851852e+00
 1.13037037e+01 2.41481481e+00 1.52592593e-02 1.43555556e-01
 2.25185185e+00 2.08888889e-01 1.00000000e+00 1.00000000e+00
 1.00000000e+00 4.20740741e-02 9.68888889e-02 4.20740741e-02
 7.80740741e-03 2.68148148e-02 4.42962963e-02]
```

7 Simulation

7.1 Set up parameters

- Reaction constants already set

[20]:
```
for i,spec in enumerate(s['species']):
    #K_spec = np.exp(phi_est[i]/st.V_N())
    parameter['K_'+spec] = K_spec[i]
```

7.2 Set up chemostats and flowstats

[21]:
```
def setPath(s,path='R5P'):

    print('\n Path =', path)

    if path == 'R5P':
        chemostats = ['ADP', 'ATP', 'CO2', 'G6P', 'H', 'H2O', 'NADP',␣
↪'NADPH', 'R5P']
        flowstats =['G6PDH2R']
        dX_G6P = 5
    elif path == 'NADPH':
        chemostats = ['ADP', 'ATP', 'CO2', 'G6P', 'H', 'H2O', 'NADP', 'NADPH']
        flowstats = []
        dX_G6P = 1
    elif path == 'both':
        chemostats = ['ADP', 'ATP', 'CO2', 'G6P', 'H', 'H2O', 'NADP',␣
↪'NADPH', 'R5P']
        flowstats =  ['PGI', 'TKT2']
        dX_G6P = 1
    elif path == 'all':
        chemostats = ['ADP', 'ATP', 'CO2', 'G6P', 'H', 'H2O', 'NADP',␣
↪'NADPH', 'R5P','G3P']
        flowstats = []
        dX_G6P = 10

    sc = st.statify(s,chemostats=chemostats)
    sf = st.statify(s,flowstats=flowstats)

    return sc,sf,dX_G6P
```

7.3 Time unit

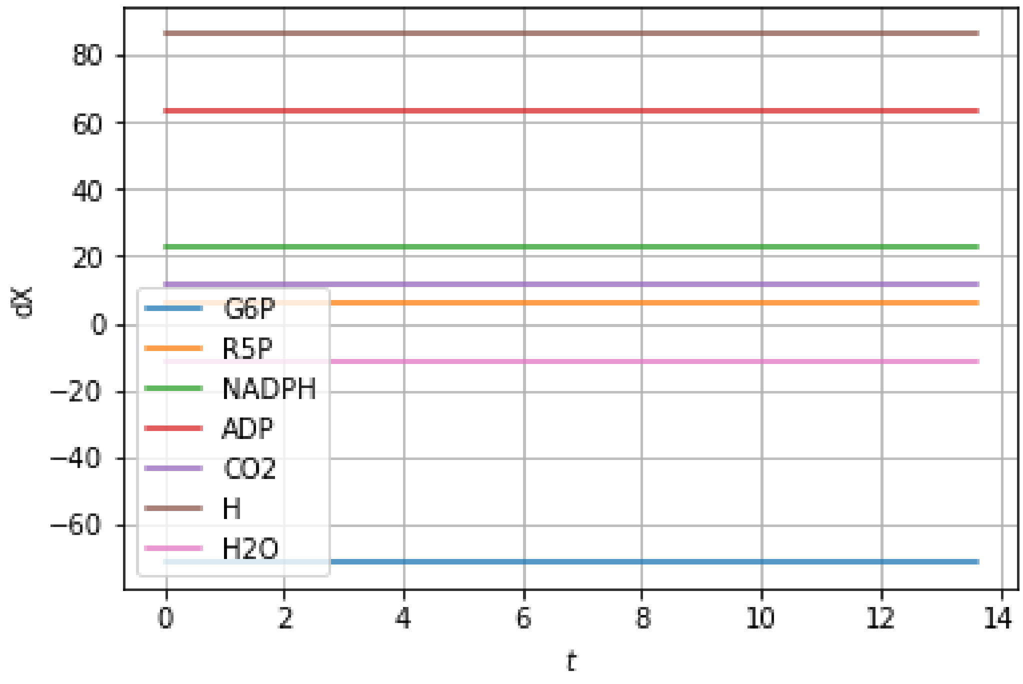
[22]:
```
##t_0 = ((1000*c_G6P)/flux_PGI)*100
print("Time unit:", t_0)
```

```
Time unit: 0.680672268907563
```

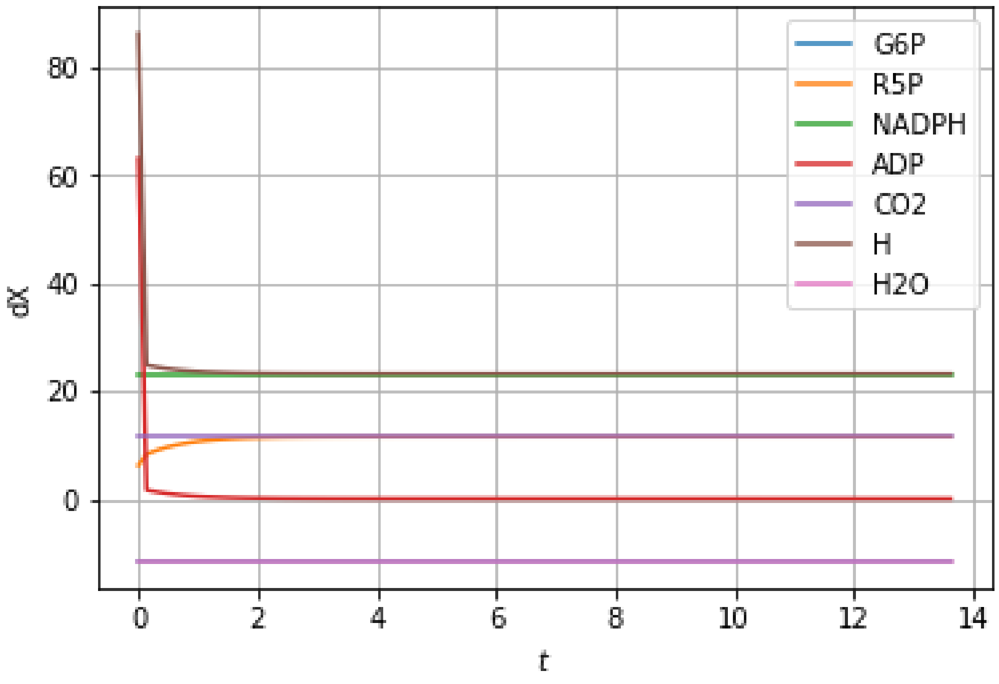
7.4 Simulation

[23]:
```
approximateFlowstats = True

Spec = ['G6P','R5P','NADPH','ADP','CO2','H','H2O']
paths = ['all','both','R5P','NADPH']
#paths = ['R5P']
RATIO = {}
for path in paths:
    Ratio = {}
    normalisedRatio = {}

    ## Set up pathway]
    spec = sc['species']
    sc,sf,dX_G6P_0 = setPath(s,path=path)

    ## Set up parameters
    par = copy.copy(parameter)
    if approximateFlowstats:
        small = 1e-3
        par = copy.copy(parameter)
        for fs in sf['flowstats']:
            par['kappa_'+fs] = small
        sf = None

    ## Simulate
    t = np.linspace(0,20*t_0,100)
    dat = st.sim(s,sc=sc,sf=sf,t=t,parameter=par,X0=conc)
    #st.plot(s,dat,species=[])
    st.plot(s,dat,reaction=[],species=Spec,dX=True)

    ## Extract some external flows
    DX = dat['dX']
    dX = {}
    for Sp in Spec:
        dX[Sp] = DX[:,spec.index(Sp)]
        Ratio[Sp] = -dX[Sp]/dX['G6P']
        normalisedRatio[Sp] =  -dX_G6P_0*dX[Sp]/dX['G6P']

    RATIO[path] = normalisedRatio

    ## Print steady-state values
    for Sp in Spec:
        ratio = Ratio[Sp][-1]
        print(f'{Sp}:\t{dX[Sp][0]:3.1f} \t{dX[Sp][-1]:3.
↪1f}\t{(dX_G6P_0*ratio):3.1f}\t{100*ratio:3.1f}%')
```

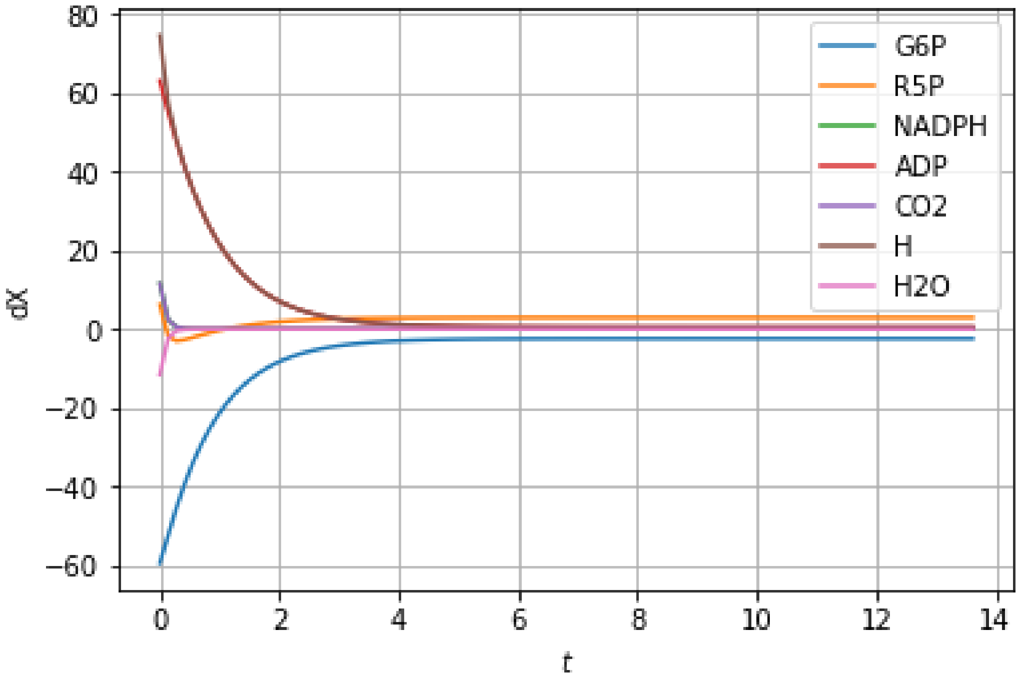
```
Path = all
```



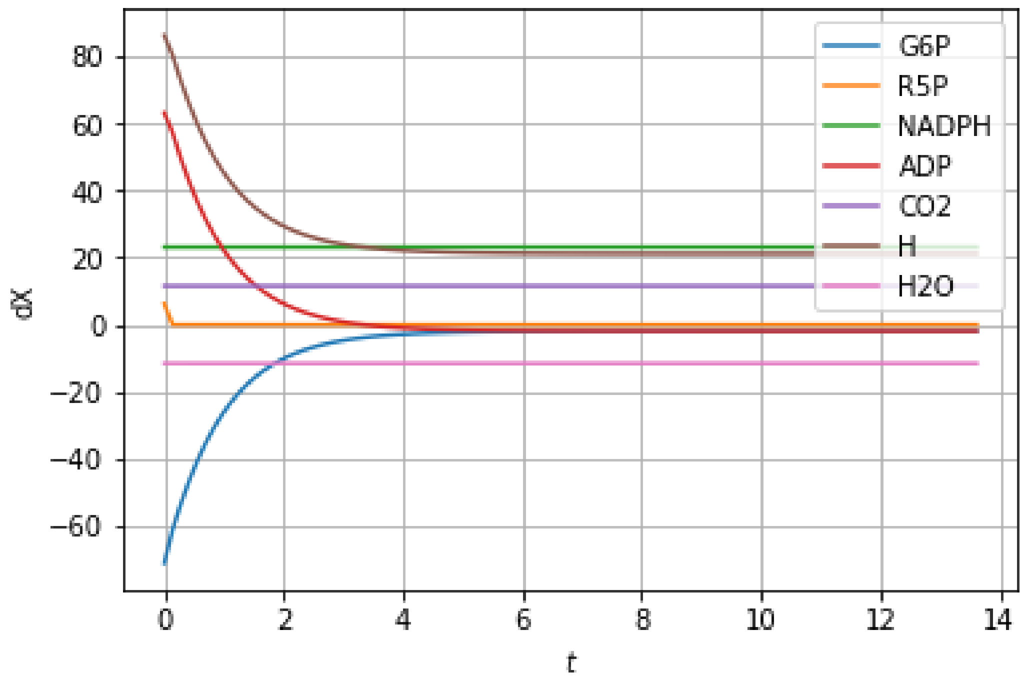
```
G6P:    -71.1   -71.1   -10.0   -100.0%
R5P:    6.2     6.2     0.9     8.7%
NADPH:  23.2    23.2    3.3     32.6%
ADP:    63.1    63.1    8.9     88.8%
CO2:    11.6    11.6    1.6     16.3%
H:      86.3    86.3    12.1    121.3%
H2O:    -11.6   -11.6   -1.6    -16.3%

 Path = both
```



```
G6P:    -11.6   -11.6   -1.0    -100.0%
R5P:    6.2     11.5    1.0     99.4%
NADPH:  23.2    23.2    2.0     200.0%
ADP:    63.1    0.0     0.0     0.4%
CO2:    11.6    11.6    1.0     100.0%
H:      86.3    23.2    2.0     200.4%
H2O:    -11.6   -11.6   -1.0    -100.0%

 Path = R5P
```



```
G6P:    -59.5   -2.4    -5.0    -100.0%
R5P:    6.2     2.9     6.0     120.0%
NADPH:  11.6    0.0     0.0     0.2%
ADP:    63.1    0.5     1.0     20.0%
CO2:    11.6    0.0     0.0     0.1%
H:      74.7    0.5     1.0     20.2%
H2O:    -11.6   -0.0    -0.0    -0.1%

 Path = NADPH
```



```
G6P:    -71.1   -1.9    -1.0    -100.0%
R5P:    6.2     0.0     0.0     0.0%
NADPH:  23.2    23.1    12.0    1200.0%
ADP:    63.1    -1.9    -1.0    -100.0%
CO2:    11.6    11.6    6.0     600.0%
H:      86.3    21.2    11.0    1100.0%
H2O:    -11.6   -11.6   -6.0    -600.0%
```

[24]:
```
## Plot ratios
name = ['i','ii','iii']
for sp in ['R5P','NADPH']:
    BigFont = 14
    plt.rcParams.update({'font.size': BigFont})
    for i,path in enumerate(['both','R5P','NADPH']):
        Ratio = RATIO[path]
        label = f'Path {name[i]}'
        plt.plot(t/t_0,Ratio[sp],label=label,linewidth=5)
    ylabel = r'$\rho_{'+sp+'}$'
    plt.ylabel(ylabel)
    plt.xlabel('$t/t_0$')
    plt.legend()
    plt.grid()
    plt.savefig(f'Figs/{sp}.pdf')
    plt.show()
```

[24]:
```
[<matplotlib.lines.Line2D at 0x7f9e03768550>]
```

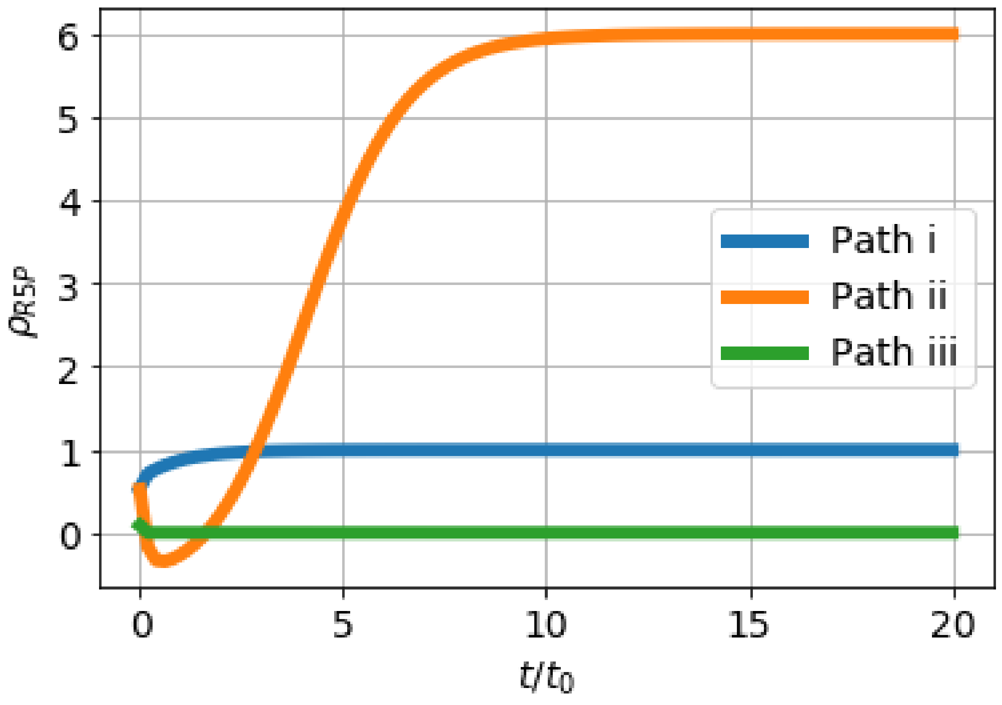
[24]:
```
[<matplotlib.lines.Line2D at 0x7f9e03768c10>]
```

```
[24]: [<matplotlib.lines.Line2D at 0x7f9e035d6b20>]
```

```
[24]: Text(0,0.5,'$\\rho_{R5P}$')
```

```
[24]: Text(0.5,0,'$t/t_0$')
```

```
[24]: <matplotlib.legend.Legend at 0x7f9e037684c0>
```



```
[24]: [<matplotlib.lines.Line2D at 0x7f9e03673fd0>]
```

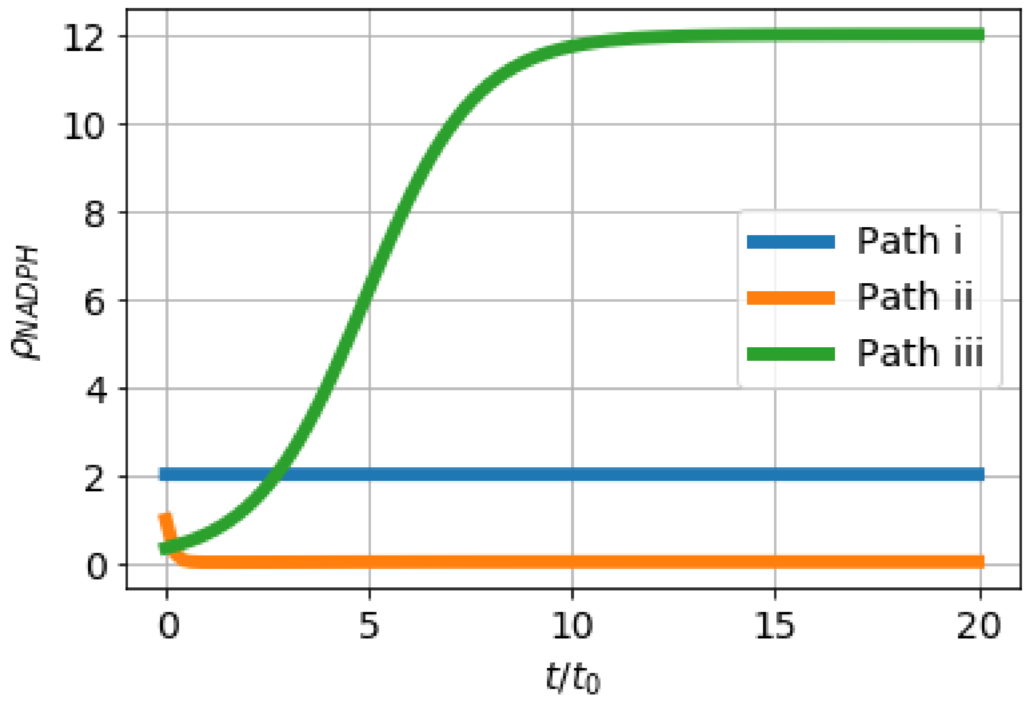
```
[24]: [<matplotlib.lines.Line2D at 0x7f9e035e3550>]
```

```
[24]: [<matplotlib.lines.Line2D at 0x7f9e03667100>]
```

```
[24]: Text(0,0.5,'$\\rho_{NADPH}$')
```

```
[24]: Text(0.5,0,'$t/t_0$')
```

```
[24]: <matplotlib.legend.Legend at 0x7f9e035e3df0>
```



```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

References

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

Peter J. Gawthrop, Peter Cudmore, and Edmund J. Crampin. Physically-plausible modelling of biomolecular systems: A simplified, energy-based model of the mitochondrial electron transport chain. *Journal of Theoretical Biology*, 493: 110223, 2020. ISSN 0022-5193. doi: 10.1016/j.jtbi.2020.110223.

J. Orth, R. Fleming, and B. Palsson. Reconstruction and use of microbial metabolic networks: the core escherichia coli metabolic model as an educational guide. *EcoSal Plus*, 2010. doi: 10.1128/ecosalplus.10.2.1.

Junyoung O. Park, Sara A. Rubin, Yi-Fan Xu, Daniel Amador-Noguez, Jing Fan, Tomer Shlomi, and Joshua D. Rabinowitz. Metabolite concentrations, fluxes and free energies imply efficient enzyme usage. *Nat Chem Biol*, 12(7):482-489, Jul 2016. ISSN 1552-4450. doi: 10.1038/nchembio.2077.