The Escherichia coli Core Model: Modular Energetic Bond Graph Analysis of Pentose Phosphate Pathways

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

January 15, 2021

Contents

Note: this is the EcoliPPP.ipynb notebook. The PDF version "The Escherichia coli Core Model: Modular Energetic Bond Graph Analysis of Pentose Phosphate Pathways" is available here.

1 Introduction

The Network Thermodynamics/Bond Graph approach of Oster et al. (1971, 1973), extended by Gawthrop and Crampin (2016, 2014, 2017), to modelling biomolecular systems developed independently from the stoichiometric approach Palsson (2006, 2011, 2015).

However, the conceptual point of intersection of the two approaches is the fact that the stoichiometric matrix is the modulus of the conceptual multiport transformer linking reactions to species. This was pointed out by Cellier and Greifeneder (2009). This means that the two approaches are complementary and each can build on the strengths of the other.

In particular, as discussed here, the Bond Graph approach adds energy to stoichiometry.

This notebook focuses on building modular models of metabolism and consequent pathway analysis based on the Escherichia coli Core Model Orth et al. (2010); in particular, the Glycolysis and Pentose Phosphate portion is extracted and analysed. Following the discussion in the textbook of Garrett and Grisham (2017), section 22.6d, various possible pathways are examined by choosing appropriate chemostats and flowstats. Gawthrop and Crampin (2018)

Assuming steady-state conditions, the corresponding pathway potentials Gawthrop (2017) are derived.

1.1 Import some python code

The bond graph analysis uses a number of Python 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

## BG stoichiometric utilities
import stoich as st

## 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

import copy

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

import importlib as imp

## Units etc
factor = 1
units = ['mV','kJ']

## Control output
quiet = True
computePhi = 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
    print(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)
    #Phi = -N.T@phi
```

```
    return phi
```

1.3 Deriving species potentials

To perform energetic analysis it is necessary to have values of the chemical potential of the species involved. One way of this is to use experimentally derived value of species potentials at standard conditions and then derive potentials corresponding to the concentrations of the species. Another approach used here, is to take experimental values of reaction potentials Φ Park et al. (2016) and derive a consistent set of species potentials ϕ using $\phi = -N^{\dagger}\Phi$ where N is the stoichiometric matrix of the reaction system and † denotes pseudo inverse.

[4]:
```
def setphi(s,S,phiS):
    ## Load up phi
    Species = S['species']
    species = s['species']
    phi = []
    for spec in species:
#         if spec in ['H_E']:
#             sp = 'H'
#         elif spec in ['O2']:
#             sp = 'H'
#         else:
        sp = spec
        ph = phiS[Species.index(sp)]
        #print(f'phi_{spec} = {ph}')
        phi.append(ph)

    phi = np.array(phi)

    ## Compute Phi
    NN = s['N']
    Phi = -NN.T@phi

    return Phi,phi
```

[5]:
```
def getPhi(s,Phi_hyd=0.5,phi_6PGL=None,quadprog=True):
    """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
```

2 Extract the model

2.1 Extract full ecoli core model from the CobraPy representation

[6]:
```
sm = Extract.extract(cobraname='textbook',Remove=['_C','__' ],
                     negReaction=['RPI','PGK','PGM','SUCOAS'], quiet=quiet)
print(sm['reaction'])
```

```
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
Multiplying reaction SUCOAS ( 69 ) by -1
['ACALD', 'ACALDT', 'ACKR', 'ACONTA', 'ACONTB', 'ACT2R', 'ADK1', 'AKGDH',
'AKGT2R', 'ALCD2X', 'ATPM', 'ATPS4R', 'BIOMASS_ECOLIORE', 'CO2T', 'CS', 'CYTBD',
'D_LACT2', 'ENO', 'ETOHT2R', 'FBA', 'FBP', 'FORT2', 'FORTI', 'FRD7', 'FRUPTS2',
'FUM', 'FUMT2_2', 'G6PDH2R', 'GAPD', 'GLCPTS', 'GLNS', 'GLNABC', 'GLUDY',
'GLUN', 'GLUSY', 'GLUT2R', 'GND', 'H2OT', 'ICDHYR', 'ICL', 'LDH_D', 'MALS',
'MALT2_2', 'MDH', 'ME1', 'ME2', 'NADH16', 'NADTRHD', 'NH4T', 'O2T', 'PDH',
'PFK', 'PFL', 'PGI', 'PGK', 'PGL', 'PGM', 'PIT2R', 'PPC', 'PPCK', 'PPS', 'PTAR',
'PYK', 'PYRT2', 'RPE', 'RPI', 'SUCCT2_2', 'SUCCT3', 'SUCDI', 'SUCOAS', 'TALA',
'THD2', 'TKT1', 'TKT2', 'TPI']
```

2.2 Extract Glycolysis, Pentose Phosphate Pathways and TCA (using PDH and PDH)

[7]:
```
name = 'GlyPPP_abg'
reaction = []

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

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

## Create submodel
sGlyPPP = Extract.choose(sm,reaction=reaction)
#Phi,phi,Phi_0,Phi_reac = getPhi(sGlyPPP,phi_6PGL=0.04)
Phi,phi,Phi_0,Phi_reac = getPhi(sGlyPPP)
sGlyPPP['name'] = name
stbg.model(sGlyPPP)


## Print all the phis
#print(phi)
species = sGlyPPP['species']
```

```
for i,ph in enumerate(phi):
    #print(species[i],ph)
    print(f'phi_{species[i]} = {int(round(ph*1000))} mV')

# Print all the phis
reac = sGlyPPP['reaction']
for i,ph in enumerate(Phi):
    print(f'Phi_{reac[i]} = {int(round(ph*1000))} mV')
```

```
Extracting 19 values of phi from 10 values of Phi
(19, 2)
Phi error = 0mV

phi_6PGC = 29 mV
phi_6PGL = 1 mV
phi_ADP = -27 mV
phi_ATP = 27 mV
phi_CO2 = -30 mV
phi_DHAP = -10 mV
phi_E4P = -27 mV
phi_F6P = -21 mV
phi_FDP = -8 mV
phi_G3P = -18 mV
phi_G6P = -5 mV
phi_H = -28 mV
phi_H2O = 1 mV
phi_NADP = 30 mV
phi_NADPH = -30 mV
phi_R5P = 5 mV
phi_RU5PD = 5 mV
phi_S7P = 24 mV
phi_XU5PD = 5 mV
Phi_PGI = 16 mV
Phi_PFK = 69 mV
Phi_FBA = 20 mV
Phi_TPI = 8 mV
Phi_G6PDH2R = 83 mV
Phi_PGL = 1 mV
Phi_GND = 115 mV
Phi_RPI = 0 mV
Phi_TKT2 = 16 mV
Phi_TALA = 54 mV
Phi_TKT1 = 4 mV
Phi_RPE = 1 mV
```

[8]:
```
## Create stoichiometry
import GlyPPP_abg
S = st.stoich(GlyPPP_abg.model(),quiet=quiet)
```

2.3 Display the extracted reactions

- () indicates reaction potential in Volts (J/coulomb)
- [] indicates reaction free energy in J/mol

See Gawthrop (2017) for a discussion of these two quantities.

[9]:
```
disp.Latex(st.sprintrl(sGlyPPP,chemformula=True,
                       Phi=Phi,showMu=showMu,units=['mV','kJ']))
```

[9]:

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

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

2.4 Code to analyse pathways defined by chemostats and flowstats

[10]:
```
## Analyse pathways defined by chemostats and flowstats
def ch(name):
    return '\\ch{'+name+'}'

def energetics(s,sp,phi):
    """Reaction energetics.
    """

    ## Phi for all reactions
    Phi = -s['N'].T@phi

    ##Phi for pathway
    ## I is the relevant indices of phi
    I = []
    for spec in sp['species']:
        i = s['species'].index(spec)
        I.append(i)
```

```
    Phip = -sp['N'].T@phi[I]

    return Phi,Phip

def pathway(bg,phi,chemostats,flowstats=[],computePhi=False,verbose=False):
    """ Analyse pathways
    """

    print('Chemostats:',sorted(chemostats))
    print('Flowstats:', sorted(flowstats))
    ## Stoichiometry
    ## Create stoichiometry from bond graph.
    s = st.stoich(bg,quiet=True)

    ## Stoichiometry with chemostats
    sc = st.statify(s,chemostats=chemostats,flowstats=flowstats)

    ## Pathway stoichiometry
    sp = st.path(s,sc)

    ## Print info
    if verbose:
        for stat in sorted(chemostats):
            print(ch(stat)+',')

    ## Energetics
    if computePhi:
        Phi,Phip = energetics(s,sp,phi)
        #print('Phi units: kJ/mol')
#         fac = -F/1000
#         units='~\si{\kilo\joule\per\mol}'
        units = '~\si{\volt}'
        print(st.sprintp(sc))
        disp.Latex(st.sprintrl(sp,chemformula=True,Phi=Phip,showMu=showMu))
        #return s,sc,sp,Phi*fac,Phip*fac,units
        return s,sc,sp,Phip
    else:
        print(st.sprintrl(sp,chemformula=True))
        Phip = 0
        return s,sc,sp,Phip

def Pathway(S,phi,chemostats,flowstats=[],computePhi=False,verbose=False):
    """ Analyse pathways
    """

    print('Chemostats:',sorted(chemostats))
    print('Flowstats:', sorted(flowstats))
    ## Stoichiometry
    ## Create stoichiometry from bond graph.
    #s = st.stoich(bg,quiet=True)
```

```
    s = copy.copy(S)
    ## Stoichiometry with chemostats
    sc = st.statify(s,chemostats=chemostats,flowstats=flowstats)

    ## Pathway stoichiometry
    sp = st.path(s,sc)

    ## Print info
    if verbose:
        for stat in sorted(chemostats):
            print(ch(stat)+',')

    ## Energetics
    if computePhi:
        Phi,Phip = energetics(s,sp,phi)
        #print('Phi units: kJ/mol')
#         fac = -F/1000
#         units='~\si{\kilo\joule\per\mol}'
        units = '~\si{\volt}'
        print(st.sprintp(sc))
        disp.Latex(st.sprintrl(sp,chemformula=True,Phi=Phip,showMu=showMu))
        #return s,sc,sp,Phi*fac,Phip*fac,units
        return s,sc,sp,Phip
    else:
        print(st.sprintrl(sp,chemformula=True))
        Phip = 0
        return s,sc,sp,Phip
```

3 Analyse Pentose Phosphate Pathway with Glycolysis

The pathways are isolated by using appropriate (zero-flow) flowstats. For compatibility with Garrett and Grisham (2017, § 18.2) the pathways start from G6P (Glucose 6-phosphate).

3.1 Common chemostats

[11]:
```
def Chemostats(start='G6P',end=None):
    chemostats = ['ADP','ATP','CO2','H','H2O','NADP','NADPH']
    chemostats += [start]
    if end is not None:
        chemostats += end
    return chemostats
```

3.2 R_5P and NADPH generation

- This pathway is isolated by setting PGI and TKT2 as flowstats and the product R_5P is added to the chemostat list.
- It is isolated from the TCA cycle by replacing the connecting reactions (PDH and PFL) by flowstats.

[12]:
```
imp.reload(st)
print('R5P and NADPH generation')
chemostats = Chemostats(start='G6P',end=['R5P'])
flowstats = ['PGI','TKT2']
#s,sc,sp,Phip,Phi,Phip,units =␣
 ↪Pathway(S,phi,chemostats,flowstats=flowstats,computePhi=computePhi)
s,sc,sp,Phip =␣
 ↪Pathway(S,phi,chemostats,flowstats=flowstats,computePhi=computePhi)
disp.Latex(st.
 ↪sprintrl(sp,chemformula=True,Phi=Phip,units=units,showMu=showMu))
```

[12]:
```
<module 'stoich' from
'/home/peterg/WORK/Research/SystemsBiology/lib/python/stoich.py'>

R5P and NADPH generation
Chemostats: ['ADP', 'ATP', 'CO2', 'G6P', 'H', 'H2O', 'NADP', 'NADPH', 'R5P']
Flowstats: ['PGI', 'TKT2']
3 pathways
0:  + PGI
1:  + G6PDH2R + PGL + GND + RPI
2:  + TKT2
```

[12]:

$$G_6P + H_2O + 2\,NADP \xrightleftharpoons{pr_1} CO_2 + 2\,H + 2\,NADPH + R_5P \quad (198\,mV)\ [-19.14\,kJ\,mol^{-1}]$$

- The pathway reaction P_1 corresponds to the R_5P and NADPH synthesis discussed in comment 1 of Garrett and Grisham (2017), p787.
- The positive reaction potential (negative reaction free energy) indicates that the reaction proceeds in the forward direction.
- It is isolated from the TCA cycle by replacing the connecting reactions (PDH and PFL) by flowstats.

3.3 R_5P generation

- This pathway is isolated by setting GAPD and G6PDH2R as flowstats and the product R_5P is added to the chemostat list.

[13]:
```
print('R5P generation')
chemostats = Chemostats(start='G6P',end=['R5P'])
flowstats = ['G6PDH2R']
s,sc,sp,Phip =␣
 ↪Pathway(S,phi,chemostats,flowstats=flowstats,computePhi=computePhi)
disp.Latex(st.
 ↪sprintrl(sp,chemformula=True,Phi=Phip,units=['mV','kJ'],showMu=showMu))
```

```
R5P generation
Chemostats: ['ADP', 'ATP', 'CO2', 'G6P', 'H', 'H2O', 'NADP', 'NADPH', 'R5P']
Flowstats: ['G6PDH2R']
2 pathways
0:  + G6PDH2R
```

```
1:  - 5 PGI - PFK - FBA - TPI - 4 RPI + 2 TKT2 + 2 TALA + 2 TKT1 + 4 RPE
```

[13]:

$$\mathrm{ADP} + \mathrm{H} + 6\,\mathrm{R_5P} \overset{\mathrm{pr_1}}{\rightleftharpoons} \mathrm{ATP} + 5\,\mathrm{G_6P} \qquad (-26\ \mathrm{mV})\ [2.55\ \mathrm{kJ\ mol^{-1}}]$$

- The pathway reaction $\mathrm{pr_1}$ corresponds to the $\mathrm{R_5P}$ synthesis discussed in comment 2 of Garrett and Grisham (2017), p787.
- The *negative* reaction potential (*positive* reaction free energy) indicates that the reaction proceeds in the *reverse* direction.

3.4 NADPH generation

- This pathway is isolated by setting GAPD as a flowstat.
- It is isolated from the TCA cycle by replacing the connecting reactions (PDH and PFL) by flowstats.

[14]:
```
print('NADPH generation')
chemostats = Chemostats(start='G6P')
flowstats = []
s,sc,sp,Phip = ␣
 ↪Pathway(S,phi,chemostats,flowstats=flowstats,computePhi=computePhi)
disp.Latex(st.
 ↪sprintrl(sp,chemformula=True,Phi=Phip,units=['mV','kJ'],showMu=showMu))
```

```
NADPH generation
Chemostats: ['ADP', 'ATP', 'CO2', 'G6P', 'H', 'H2O', 'NADP', 'NADPH']
Flowstats: []
1 pathways
0:  - 5 PGI - PFK - FBA - TPI + 6 G6PDH2R + 6 PGL + 6 GND + 2 RPI + 2 TKT2 + 2
TALA + 2 TKT1 + 4 RPE
```

[14]:

$$\mathrm{ADP} + \mathrm{G_6P} + 6\,\mathrm{H_2O} + 12\,\mathrm{NADP} \overset{\mathrm{pr_1}}{\rightleftharpoons} \mathrm{ATP} + 6\,\mathrm{CO_2} + 11\,\mathrm{H} + 12\,\mathrm{NADPH} \quad (1164\ \mathrm{mV})\ [-112.31\ \mathrm{kJ\ mol^{-1}}]$$

- The pathway reaction $\mathrm{pr_1}$ corresponds to the NADPH synthesis discussed in comment 3 of Garrett and Grisham (2017), p787.
- The positive reaction potential (negative reaction free energy) indicates that the reaction proceeds in the forward direction.

References

F.E. Cellier and J. Greifeneder. Modeling chemical reactions in modelica by use of chemo-bonds. In *Proceedings 7th Modelica Conference*, Como, Italy, September 2009.

Reginald H. Garrett and Charles M. Grisham. *Biochemistry*. Cengage Learning, Boston, MA, 6th edition, 2017.

P. J. Gawthrop. Bond graph modeling of chemiosmotic biomolecular energy transduction. *IEEE Transactions on NanoBioscience*, 16(3):177–188, April 2017. ISSN 1536-1241. doi: 10.1109/TNB.2017.2674683. Available at arXiv:1611.04264.

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

Peter J. Gawthrop and Edmund J. Crampin. Energy-based analysis of biochemical cycles using bond graphs. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, 470(2171):1–25, 2014. doi: 10.1098/rspa.2014.0459. Available at arXiv:1406.2447.

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

Peter J. Gawthrop and Edmund J. Crampin. Biomolecular system energetics. In *Proceedings of the 13th International Conference on Bond Graph Modeling (ICBGM'18)*, Bordeaux, 2018. Society for Computer Simulation. Available at arXiv:1803.09231.

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.

George Oster, Alan Perelson, and Aharon Katchalsky. Network thermodynamics. *Nature*, 234: 393–399, December 1971. doi: 10.1038/234393a0.

George F. Oster, Alan S. Perelson, and Aharon Katchalsky. Network thermodynamics: dynamic modelling of biophysical systems. *Quarterly Reviews of Biophysics*, 6(01):1–134, 1973. doi: 10.1017/S0033583500000081.

Bernhard Palsson. *Systems biology: properties of reconstructed networks*. Cambridge University Press, 2006. ISBN 0521859034.

Bernhard Palsson. *Systems Biology: Simulation of Dynamic Network States*. Cambridge University Press, 2011.

Bernhard Palsson. *Systems Biology: Constraint-Based Reconstruction and Analysis*. Cambridge University Press, Cambridge, 2015.

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.