Physically-Plausible Parameters

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

January 15, 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.

$$\text{minimise } \frac{1}{2}x^TPx + q^Tx \tag{1}$$

$$\text{subject to } Gx \leq h \tag{2}$$

$$\text{and } Ax = b \tag{3}$$

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]:
```python
## 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
```

2 Conversion factor

[4]:
```python
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]:
```
## Set up chemostats
chemostats = ['ADP','ATP','H','H2O','NADP','NADPH','CO2']
chemostats += ['G6P','G3P','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]:

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

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

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

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

```
\begin{align}
K &=
\left(\begin{matrix}1 & 0 & -2\\1 & 0 & 0\\1 & 0 & 0\\1 & 0 & 0\\0 & 1 & 2\\0 &
1 & 2\\0 & 1 & 2\\0 & 1 & 0\\0 & 0 & 1\\0 & 0 & 1\\0 & 0 & 1\\0 & 0 &
2\end{matrix}\right)
\end{align}
```

[8]:

$$K = \begin{pmatrix}1 & 0 & -2\\1 & 0 & 0\\1 & 0 & 0\\1 & 0 & 0\\0 & 1 & 2\\0 & 1 & 2\\0 & 1 & 2\\0 & 1 & 0\\0 & 0 & 1\\0 & 0 & 1\\0 & 0 & 1\\0 & 0 & 2\end{pmatrix} \tag{13}$$

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
(19, 2)
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]:

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

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

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)
        f0 = np.linalg.solve(Kp.T@Kp,Kp.T@f)
        for i,Reac in enumerate(Reaction):
            Flux[Reac] = f0[i][0]
        #print(f0)
        f_est = Kp@f0
        #print(Kp@f0-f)

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

    return Flux,f0,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]:
```python
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)))
    #print(f0)
    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 Show computed reaction flows

[15]:
```python
Reaction = ['PPP1','PPP2','PPP3']
#print(s['reaction'])
## Reaction flows
Flux = phiData.ParRubXu16_flux()
flux = Flux['Ecoli']

## Normalise flux wrt PGI = 100
flux_PGI = flux['PGI']
for reac in flux.keys():
    flux[reac] *= 100/flux_PGI

fluxp,f0,f_est,f,reaction = PathwayFlux(sc['K'],s['reaction'],Reaction,flux)
```

```
## Reaction constants
f_est = sc['K']@f0
parameter = reactionConstant(s,phi_est,f_est)

#f_est = sc['K']@f0
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')
```

```
Flux error = 3.10e-01


%% LaTeX table
\hline
Reaction &       $\Phi$ &        $\hat{\Phi}$ &  $f$ & $\hat{f}$ &
$\hat{\kappa}$\\
\hline
PGI &     16 &     16 &     100 & 99 & 2.6e+02 \\
PFK &     69 &     69 &     104 & 105 & 9.1e+01 \\
FBA &     20 &     20 &     106 & 105 & 2.7e+02 \\
TPI &     8 &      8 &      105 & 105 & 5.9e+02 \\
G6PDH2R &          -- &     83 &     -- & 19 & 7.8 \\
PGL &     -- &     1 &      -- & 19 & 4.9e+02 \\
GND &     115 &    115 &    19 & 19 & 2.1 \\
RPI &     0 &      0 &      13 & 13 & 7e+03 \\
TKT2 &    16 &     16 &     2 & 3 & 1.5e+01 \\
TALA &    54 &     54 &     -- & 3 & 2.8 \\
TKT1 &    4 &      4 &      5 & 3 & 1.5e+01 \\
RPE &     1 &      1 &      6 & 6 & 1.6e+02 \\
\hline
```

5.2 Show computed chemostat flows

[16]:
```
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 &    105 \\
ATP &    -105 \\
CO2 &    19 \\
G3P &    213 \\
G6P &    -119 \\
H &      144 \\
H2O &    -19 \\
NADP &   -39 \\
NADPH &  39 \\
R5P &    10 \\
\hline
```

5.3 Show pathway flows

[17]:
```
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 &   105 \\
PPP2 &   13 \\
PPP3 &   3 \\
\hline
```

6 Species constants

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

[18]:
```python
#imp.reload(phiData)

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

concentration = phiData.ParRubXu16_conc()
#concentration['H'] = 1e-7

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

print('\\hline')
```

```
%% LaTeX table
\hline
Species &        $\hat{\phi}~mV$ & $c$ & $\hat{K}$ \\
\hline
6PGC & 29 &      0.017 & 62.4 \\
ADP & -27 &      0.57 & 1.711 \\
ATP & 27 &       4.7 & 0.22 \\
CO2 & -30 &      7.6 & 0.1272 \\
DHAP & -10 &     1.6 & 0.6075 \\
E4P & -27 &      0.01 & 94.47 \\
F6P & -21 &      0.097 & 10.1 \\
FDP & -8 &       1.5 & 0.6528 \\
G3P & -18 &      0.14 & 6.967 \\
G6P & -5 &       0.68 & 1.474 \\
NADP & 30 &      0.028 & 36.29 \\
NADPH & -30 &    0.065 & 14.83 \\
R5P & 5 &        0.028 & 35.4 \\
RU5PD & 5 &      0.0053 & 190.8 \\
S7P & 24 &       0.018 & 56.58 \\
XU5PD & 5 &      0.03 & 33.6 \\
\hline
```

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.