# 1. Minimum Energy Metabolic Flux Distributions

In this lecture, we will:

1. Introduce Gibbs energy minimiza
   energy for multiple coupled react
2. Compute the reversibility of mult
   catalyzed reactions

Discussion problem:

- Compute the `low-energy` metab
  a metabolic network

📖 📚 Table of Contents



# 2. Direct Gibbs Energy Minimization of Multiple Chemical Reactions in Central Carbon Metabolism

Calculate the equilibrium extent of reaction and the equilibrium concentrations for the first five steps of glycolysis occurring in an *E. coli* MG-1655 cell-free extract using Direct Gibbs Energy Minimization (DGEM). The pathway in this example is here.

**Assumptions**

- The cell-free extract has a constant V = 30.0μL
- The cell-free extract acts like an *ideal* liquid solution
- The cell-free reaction is at a constant T, P
- The *default* metabolic settings used by eQuilibrator are valid for this system

## 2.1 Theory

The problem asks us to use the Direct Gibbs Energy Minimization (DGEM) approach. For multiple reactions, the Gibbs expression:

$$\hat{G} = \sum_{i=1}^{\mathcal{M}} \bar{G}_i n_i$$

becomes:

$$\frac{\left(\hat{G} - \sum_{i=1}^{\mathcal{M}} n_i^\circ G_i^\circ\right)}{RT} = \sum_{j=1}^{\mathcal{R}} \epsilon_j \left(\frac{\Delta G_j^\circ}{RT}\right) + \sum_{i=1}^{\mathcal{M}} n_i \ln \hat{a}_i$$

where the number of mol for species *i* is given by:

$$n_i = n_i^\circ + \sum_{r=1}^{\mathcal{R}} \sigma_{ir}\epsilon_r \qquad i = 1, 2, \ldots, \mathcal{M}$$

The quantity $\Delta G_j^\circ$ denotes the Gibbs energy of reaction for reaction *j* (units: kJ/mmol), and $\hat{a}_i$ denotes the ratio of fugacity for component *i*, which (after the assumption of an ideal solution) becomes: $\ln \hat{a}_i = x_i$ where $x_i$ denotes the mol fraction of component *i*.

To estimate the equilibrium extent *vector* we minimize Gibbs energy expression, subject to constraints. Our decision variables (what we are looking for) are the extents of reaction $\epsilon_i, i = 1, \ldots, \mathcal{R}$ subject to bounds on each extent $\epsilon_i \in [0, \star], \forall i$ and $n_i \geq 0, \forall i$.

We implement $\epsilon_i \in [0, \star]$ as a box constraint, and $n_i \geq 0$ using a Penalty Method.

## 2.2 Implementation

```
T = 37.0 + 273.15; # units: K
```

```
V = 30*(1/1e6); # units: L
```

```
R = 8.314*(1/1000)*(1/ΔG_sf); # units: kJ/nmol-K
```

```
begin
```

```
# setup problem parameters -
parameters_dict = Dict{String,Any}()

# conversion factor -
ΔG_sf = 1e9; # convert to mol to *mol

# what are my system dimensions?
ℳ = 8 # number of metabolites
ℛ = 5 # number of reactions

# ΔG_formation data -
G_formation_array = zeros(ℳ)
G_formation_array[1] = -409.4       # 1 gluc kJ/mol
G_formation_array[2] = -1304.7      # 2 gluc-6-p
kJ/mol
G_formation_array[3] = -2280.7      # 3 atp kJ/mol
G_formation_array[4] = -1405.9      # 4 adp kJ/mol
G_formation_array[5] = -1302.1      # 5 fruc-6-p
kJ/mol
G_formation_array[6] = -2193.6      # 6 fruc-1,6-
bis-p kJ/mol
G_formation_array[7] = -1097.2      # 7 dhap kJ/mol
G_formation_array[8] = -1091.5      # 8 ga3p kJ/mol
parameters_dict["G_formation_array"] = (1/
ΔG_sf)*G_formation_array # units: kJ/*mol

# what are my initial condtions?
n_initial_array = 1.0*ones(ℳ)
n_initial_array[1] = 35.9*(V)*(1e9/1e3)         #
1 gluc nmol
n_initial_array[3] = 2000.0                     #
3 atp nmol
parameters_dict["n_initial_array"] =
n_initial_array

# setup stoichiometric array -
S = [

    # ϵ₁ ϵ₂  ϵ₃  ϵ₄  ϵ₅
    -1.0 0.0 0.0 0.0 0.0    ; # 1 gluc
    1.0 -1.0 0.0 0.0 0.0    ; # 2 gluc-6-p
```

```
* Status: success

* Candidate solution
   Final objective value:     -1.785180e+04

* Found with
   Algorithm:     Fminbox with BFGS

* Convergence measures
   |x - x'|               = 4.93e-07 ≰ 0.0e+00
   |x - x'|/|x'|          = 2.61e-10 ≰ 0.0e+00
   |f(x) - f(x')|         = 0.00e+00 ≤ 0.0e+00
   |f(x) - f(x')|/|f(x')| = 0.00e+00 ≤ 0.0e+00
   |g(x)|                 = 9.58e-09 ≤ 1.0e-08

* Work counters
   Seconds run:   0  (vs limit Inf)
   Iterations:    3
   f(x) calls:    283
   ∇f(x) calls:   283
```

```
begin

    # setup bounds -
    L = zeros(ℛ)
    U = maximum(n_initial_array)*ones(ℛ)

    # set the initial -
    xinitial = 0.001*ones(ℛ)
    xinitial[1] = 0.1*maximum(U)

    # setup the objective function -
    OF_closed(p) = objective_function_closed(p,
    parameters_dict)

    # call the optimizer -
    opt_result_closed = optimize(OF_closed, L, U,
xinitial, Fminbox(BFGS()))
end
```

| Reaction | ΔG_rxn<br>kJ/mol-K | ϵ<br>nmol | ϵ-scale<br>A |
|---|---|---|---|
| glc + atp = g6p + adp | -20.5 | 1072.4 | 0.99572 |
| g6p = f6p | 2.6 | 952.37 | 0.8842 |
| f6p + atp = f16bp + adp | -16.7 | 909.215 | 0.8442 |
| f16bp = dhap + ga3p | 4.9 | 644.181 | 0.59812 |
| ga3p = dhap | -5.7 | 517.68 | 0.48066 |

```
with_terminal() do

    ϵ = Optim.minimizer(opt_result_closed)
```

```
reaction_string_array = [
    "glc + atp = g6p + adp"     ;
    "g6p = f6p"                 ;
    "f6p + atp = f16bp + adp"   ;
    "f16bp = dhap + ga3p"       ;
    "ga3p = dhap"               ;
]

# compute the dG_reaction -
G_formation_array =
parameters_dict["G_formation_array"]
ΔG_rxn = transpose(S)*G_formation_array

# compute the equlibrium constant -
n_final = n_initial_array + S*ϵ

# compute the final mol fraction -
ln_x_final = log.((1/sum(n_final)).*n_final)

# make the data table array -
data_table_array = Array{Any,2}(undef,ℛ,7)
for reaction_index = 1:ℛ
    data_table_array[reaction_index,1] =
   reaction_string_array[reaction_index]
    data_table_array[reaction_index,2] =
   ΔG_rxn[reaction_index]*(ΔG_sf)
    data_table_array[reaction_index,3] =
   ϵ[reaction_index]
    data_table_array[reaction_index,4] =
   ϵ[reaction_index]*(1/n_initial_array[1])

    # compute the Keq -
    tmp = dot(S[:,reaction_index],ln_x_final)
    K_eq = exp(tmp)

    data_table_array[reaction_index,5] = K_eq
    data_table_array[reaction_index,6] = exp(-
   ΔG_rxn[reaction_index]/(R*T))
    data_table_array[reaction_index,7] =
   sign(ΔG_rxn[reaction_index]/(R*T)) == 1 ? true
   : false
```

| Species | n_i<br>nmol | n_f<br>nmol | C_i<br>mM | C_f<br>mM |
|---|---|---|---|---|
| glucose | 1077.0 | 4.60179 | 35.9 | 0.153393 |
| g6p | 1.0 | 121.028 | 0.0333333 | 4.03427 |
| atp | 2000.0 | 18.3873 | 66.6667 | 0.612909 |
| adp | 1.0 | 1982.61 | 0.0333333 | 66.0871 |
| f6p | 1.0 | 44.1556 | 0.0333333 | 1.47185 |
| f16bp | 1.0 | 266.033 | 0.0333333 | 8.86778 |
| dhap | 1.0 | 1162.86 | 0.0333333 | 38.762 |
| ga3p | 1.0 | 127.501 | 0.0333333 | 4.25004 |

```
with_terminal() do

    ϵ = Optim.minimizer(opt_result_closed)
    S = parameters_dict["S"]
    n_initial_array =
    parameters_dict["n_initial_array"]
    n = n_initial_array + S*ϵ

    # setp table_data_array -
    table_data_array = Array{Any,2}(undef,ℳ,5)
    species_array =
    ["glucose","g6p","atp","adp","f6p","f16bp","dhap","g
    a3p"]
    for species_index = 1:ℳ
        table_data_array[species_index,1] =
        species_array[species_index]
        table_data_array[species_index,2] =
        n_initial_array[species_index]
        table_data_array[species_index,3] =
        n[species_index]
        table_data_array[species_index,4] =
        (1/V)*n_initial_array[species_index]*(1e3)*(1/
        ΔG_sf) # converts to mM
        table_data_array[species_index,5] =
        (1/V)*n[species_index]*(1e3)*(1/ΔG_sf) #
        converts to mM
    end

    # setup pretty table -
    # header row -
    path_table_header_row = (["Species","n_i","n_f",
    "C_i", "C_f"],["","nmol","nmol", "mM", "mM"]);

    # write the table -
    pretty_table(table_data_array;
    header=path_table_header_row)
end
```

# 3. Discussion problem (experimental)

Compute the metabolic flux distribution for upper glycolysis using Direct Gibbs Energy Minimization (DGEM) for the first five reactions of glycolysis. The reactions operate in an open logical control volume with a single input (s=1) and a single output (s=2). Let the rate of glucose input into the logical control volume be $\dot{n}_{1,1} = 1077$ nmol/time and the rate of ATP input $\dot{n}_{3,1} = 2000$ nmol/time. All other components enter the logical control volume at 1.0 nmol/time. All components can exit the logical control volume.

**Compute**

- The open extent of reaction $\dot{\epsilon}_i \; \forall i$ using a DGEM approach for an *unbounded* exit stream and unbounded extent ($\dot{\epsilon}_i \geq 0 \; \forall i$).
- The open extent of reaction $\dot{\epsilon}_i \; \forall i$ using a DGEM approach for a *bounded* exit stream and unbounded extent ($\dot{\epsilon}_i \geq 0 \; \forall i$). In particular, let's simulate the logical exit condition for DHAP $\dot{n_{7,2}} \leq 10$.

**Assumptions**

- The cell-free extract has a constant V = 30.0µL
- The cell-free extract acts like an *ideal* liquid solution
- The cell-free reaction is at a constant T, P
- The cell-free extract is well mixed
- The *default* metabolic settings used by eQuilibrator are valid for this system

## 3.1 Theory

The problem asks us to use the Direct Gibbs Energy Minimization (DGEM) approach to estimate metabolic flux in an `open system`. For multiple reactions in an `open problem`, the Gibbs expression is now given by:

$$\dot{G} = \sum_{i=1}^{\mathcal{M}} \bar{G}_i \dot{n}_{i,s^\star}$$

where $\dot{n}_i$ denotes the $\star$mol flow rate into/from the logical control volume (units: $\star$mol/time), $\bar{G}_i$ denotes the partial molar Gibbs energy for component $i$ (units: energy/$\star$mol), and $\dot{G}$ denotes the Gibbs energy in the logical control volume (units: energy/time). The `open` mol balance around species $i$ inside the *logical* control volume (assuming a single input s=1 and output steam s=2):

$$\dot{n}_{i,2} = \dot{n}_{i,1} + \sum_{j=1}^{\mathcal{R}} \sigma_{ij} \dot{\epsilon}_j \qquad i = 1, 2, \ldots, \mathcal{M}$$

These balances can be used as constraints to find the optimal open extent of reaction with the lowest total Gibbs energy for a particular stream. In particular, the total Gibbs energy for an open system in which we minimize the energy of the **exit stream** is given by:

$$\dot{G} = \sum_{i=1}^{\mathcal{M}} \bar{G}_i \dot{n}_{i,2}$$

where the partial molar Gibbs energy (assuming an ideal solution) is given by:

$$\bar{G}_i = G_i^\circ + RT \ln x_i \qquad i = 1, 2, \ldots, \mathcal{M}$$

and:

$$x_i = \frac{\dot{n}_i}{\sum_{j=1}^{\mathcal{M}} \dot{n}_j} \qquad i = 1, 2, \ldots, \mathcal{M}$$

To estimate the open extent *vector* we minimize the open Gibbs energy expression, subject to constraints. Our decision variables (what we are looking for) are the open extents of reaction $\dot{\epsilon}_i, i = 1, \ldots, \mathcal{R}$. In this case the constraints are bounds on each extent $\dot{\epsilon}_i \in [0, \star], \forall i$ and $n_{i,\star} \geq 0, \forall i$.

## 3.2 Implementation

```
begin

    # problem setup -
    open_parameters_dict = Dict{String,Any}()
    open_parameters_dict["G_formation_array"] =
    parameters_dict["G_formation_array"];

    # setup the reaction to pull ga3p -
    Sₒ = [

        # ϵ₁ ϵ₂  ϵ₃  ϵ₄  ϵ₅
        -1.0 0.0 0.0 0.0 0.0    ; # 1 gluc
        1.0 -1.0 0.0 0.0 0.0    ; # 2 gluc-6-p
        -1.0 0.0 -1.0 0.0 0.0   ; # 3 atp
        1.0 0.0 1.0 0.0 0.0     ; # 4 adp
        0.0 1.0 -1.0 0.0 0.0    ; # 5 fruc-6-p
        0.0 0.0 1.0 -1.0 0.0    ; # 6 fruc-1,6-bis-p
        0.0 0.0 0.0 1.0 1.0     ; # 7 dhap
        0.0 0.0 0.0 1.0 -1.0    ; # 8 ga3p
    ];
    open_parameters_dict["S"] = Sₒ
    (ℳₒ,ℛₒ) = size(Sₒ)

    # what are my initial condtions?
    n_dot_in_array = 0.01*ones(ℳₒ)
    n_dot_in_array[1] = 1077.0                  # 1
    gluc nmol/time
    n_dot_in_array[3] = 2000.0                  # 3
    atp nmol/time
    open_parameters_dict["n_dot_in_array"] =
    n_dot_in_array

    # what are the outflow terms -
    n_dot_out_upper_bound_array = 100000.0*ones(ℳₒ)

    # uncomment me to impose upper bound for DHAP -
    n_dot_out_upper_bound_array[7] = 5.0
    open_parameters_dict["n_dot_out_upper_bound_array"]
     = n_dot_out_upper_bound_array

    # show -
```

```
* Status: success

* Candidate solution
   Final objective value:     -5.014041e-03

* Found with
   Algorithm:     Fminbox with BFGS

* Convergence measures
   |x - x'|               = 0.00e+00 ≤ 0.0e+00
   |x - x'|/|x'|          = 0.00e+00 ≤ 0.0e+00
   |f(x) - f(x')|         = 0.00e+00 ≤ 0.0e+00
   |f(x) - f(x')|/|f(x')| = 0.00e+00 ≤ 0.0e+00
   |g(x)|                 = 7.12e-04 ≰ 1.0e-08

* Work counters
   Seconds run:   0  (vs limit Inf)
   Iterations:    9
   f(x) calls:    1948
   ∇f(x) calls:   1948
```

```
begin

    # setup bounds -
    Lₒ = zeros(ℛₒ)

    # uncomment me:
    Uₒ = 1000000.0*ones(ℛₒ)

    # set the initial extent -
    ϵₒ = 0.1*ones(ℛₒ)

    # setup the objective function -
    OF_open(p) = objective_function_open(p,
    open_parameters_dict)

    # uncomment me: call the optimizer -
    opt_result_open = optimize(OF_open, Lₒ, Uₒ, ϵₒ,
Fminbox(BFGS()))
end
```

| Reaction | ϵdot<br>nmol/time | flux<br>mM/time |
|---|---|---|
| glc + atp = g6p + adp | 163.582 | 5.45273 |
| g6p = f6p | 100.768 | 3.35895 |
| f6p + atp = f16bp + adp | 13.0959 | 0.436531 |
| f16bp = dhap + ga3p | 4.99004 | 0.166335 |
| ga3p = dhap | 4.42622e-44 | 1.47541e-45 |

```
with_terminal() do

    ϵ = Optim.minimizer(opt_result_open)
    reaction_string_array = [
        "glc + atp = g6p + adp"     ;
        "g6p = f6p"                 ;
        "f6p + atp = f16bp + adp"   ;
        "f16bp = dhap + ga3p"       ;
        "ga3p = dhap"               ;
    ]


    # make the data table array -
    data_table_array = Array{Any,2}(undef,ℛₒ,3)
    for reaction_index = 1:ℛₒ
        data_table_array[reaction_index,1] =
       reaction_string_array[reaction_index]
        data_table_array[reaction_index,2] =
       ϵ[reaction_index]
        data_table_array[reaction_index,3] =
       ϵ[reaction_index]*(1/V)*(1e3/1e9)
    end

    # setup pretty table -
    # header row -
    path_table_header_row =
    (["Reaction","ϵdot","flux"],["","nmol/time",
    "mM/time"]);

    # write the table -
    pretty_table(data_table_array;
    header=path_table_header_row)
end
```

| Species | n_dot_in nmol/time | n_dot_out nmol/time | $dn_i/dt$ nmol/time |
|---|---|---|---|
| glucose | 1077.0 | 913.418 | -163.582 |
| g6p | 0.01 | 62.8234 | 62.8134 |
| atp | 2000.0 | 1823.32 | -176.678 |
| adp | 0.01 | 176.688 | 176.678 |
| f6p | 0.01 | 87.6825 | 87.6725 |
| f16bp | 0.01 | 8.11589 | 8.10589 |
| dhap | 0.01 | 5.00004 | 4.99004 |
| ga3p | 0.01 | 5.00004 | 4.99004 |

```
with_terminal() do

    ϵ = Optim.minimizer(opt_result_open)
    S = open_parameters_dict["S"]
    n_initial_array =
    open_parameters_dict["n_dot_in_array"]
    n = n_initial_array + S*ϵ

    # setp table_data_array -
    table_data_array = Array{Any,2}(undef,𝓜,4)
    species_array =
    ["glucose","g6p","atp","adp","f6p","f16bp","dhap","g
    a3p"]
    for species_index = 1:𝓜
        table_data_array[species_index,1] =
       species_array[species_index]
        table_data_array[species_index,2] =
       n_initial_array[species_index]
        table_data_array[species_index,3] =
       n[species_index]
        table_data_array[species_index,4] =
       n[species_index] -
       n_initial_array[species_index]
    end

    # setup pretty table -
    # header row -
    path_table_header_row =
    (["Species","n_dot_in","n_dot_out","dnᵢ/dt"],
    ["","nmol/time","nmol/time","nmol/time"]);

    # write the table -
    pretty_table(table_data_array;
    header=path_table_header_row)
end
```

# 4. Summary and Conclusions

In this lecture we:

1. Introduced Gibbs energy minimization and partial molar Gibbs energy
2. Computed the reversibility of multiple coupled enzyme-catalyzed reactions
3. Used the Direct Gibbs Energy Minimization (DGEM) approach to estimate metabolic flux in an `open logical system`

# 5. Next time:

We'll discuss enzyme kinetics and the origin of the bounds conditions. In particular, we'll look at:

- The assumptions that underly Michaelis-Menten kinetics (and the derivation)
- The MWC and Sequential models for allosteric enzyme kinetics
- Developing our approach to modeling enzyme kinetics (it's going to be crazy awesome!)

# 6. Julia function library

```
objective_function_open (generic function with 1 method)
```

```
function objective_function_open(ϵ, parameters)

    # get data from the parameters -
    G_formation_array = parameters["G_formation_array"]
    S = parameters["S"]
    n_dot_in = parameters["n_dot_in_array"]
    UB_ndot_out =
    parameters["n_dot_out_upper_bound_array"]
    RT = R*T

    # compute the n_dot_out and mol fraction -
    tmp = n_dot_in + S*ϵ
    n_dot_out = max.(0.0, tmp) # hack: if we have a
    mol count less than 0, correct -
    n_total = sum(n_dot_out)
    x_array = (1/n_total)*n_dot_out
    activity_terms = log.(x_array)

    # compute the partial molar Gibbs energy --
    G_bar = G_formation_array .+ RT*(activity_terms)

    # compute the objective value -
    𝒪 = sum(n_dot_out.*G_bar)

    # setup non-negative penalty term array -
    penalty_terms_array_1 = Array{Float64,1}()
    for species_index ∈ 1:ℳ

        penalty_term = max(0,-1*tmp[species_index])^2
        push!(penalty_terms_array_1, penalty_term)
    end
    𝒫₁ = sum(penalty_terms_array_1);

    # setup penality term array -
    penalty_terms_array_2 = Array{Float64,1}()
    for species_index = 1:ℳ

        # compute the tmp term -
```

```
objective_function_closed (generic function with 1 method)
```

```
function objective_function_closed(ϵ,parameters)

    # get data from the parameters -
    G_formation_array = parameters["G_formation_array"]
    S = parameters["S"]
    n_initial_array = parameters["n_initial_array"]
    RT = R*T

    # compute the ΔG/RT terms -
    ΔG_terms = (1/RT)*transpose(S)*G_formation_array
    term_1 = sum(ΔG_terms.*ϵ)

    # compute mols -
    tmp = n_initial_array + S*ϵ
    n_array = max.(0.0,tmp) # hack: if we have a mol
   count less than 0, correct -
    n_total = sum(n_array)
    x_array = (1/n_total)*n_array
    activity_terms = log.(x_array)
    term_2 = sum(n_array.*activity_terms)

    # setup non-negative penalty term array -
    penalty_terms_array_1 = Array{Float64,1}()
    for species_index ∈ 1:𝓜

        penalty_term = max(0,-1*tmp[species_index])^2
        push!(penalty_terms_array_1, penalty_term)
    end
    𝒫₁ = sum(penalty_terms_array_1);

    # return -
    return (term_1 + term_2) + 10*𝒫₁
end
```

```
begin

    # import some packages -
    using PlutoUI
    using PrettyTables
    using Optim
    using Plots
    using LinearAlgebra

    # setup paths -
    const _PATH_TO_NOTEBOOK = pwd()
    const _PATH_TO_DATA =
    joinpath(_PATH_TO_NOTEBOOK,"data")
    const _PATH_TO_FIGS =
    joinpath(_PATH_TO_NOTEBOOK,"figs")
    const _PATH_TO_SRC =
    joinpath(_PATH_TO_NOTEBOOK,"src")

    # return -
    nothing
end
```

```
TableOfContents(title="📚 Table of Contents",
indent=true, depth=5, aside=true)
```

```
html"""
<style>
main {
    max-width: 860px;
    width: 70%;
    margin: auto;
    font-family: "Roboto, monospace";
}

a {
    color: blue;
    text-decoration: none;
}
</style>"""
```

```
html"""
<script>
    // initialize -
    var section = 0;
    var subsection = 0;
    var subsubsection = 0;
    var headers = document.querySelectorAll('h3, h5,
    h6');

    // main loop -
    for (var i=0; i < headers.length; i++) {

        var header = headers[i];
        var text = header.innerText;
        var original = header.getAttribute("text-
    original");
        if (original === null) {

            // Save original header text
            header.setAttribute("text-original", text);
        } else {

            // Replace with original text before
           adding section number
            text = header.getAttribute("text-
    original");
        }

        var numbering = "";
        switch (header.tagName) {
            case 'H3':
                section += 1;
                numbering = section + ".";
                subsection = 0;
                break;
            case 'H5':
                subsection += 1;
                numbering = section + "." + subsection;
                break;
            case 'H6':
```