1. Reversibilty of a Metabolic Reaction

In this lecture, we will

1. Introduce a general model for the flux bounds constraints in a constraint-based calculation
2. Introduce direct Gibbs energy minimization using the partial molar Gibbs energy
3. Compute the reversibility of a single enzyme-catalyzed reaction

```
md"""
### Reversibilty of a Metabolic Reaction

In this lecture, we will
1. Introduce a general model for the flux bounds
constraints in a constraint-based calculation
1. Introduce direct Gibbs energy minimization using the
partial molar Gibbs energy
1. Compute the reversibility of a single enzyme-catalyzed
reaction
"""
```

2. A model for flux bounds

The flux bounds are important constraints in flux balance analysis calculations and the convex decomposition of the stoichiometric array. Beyond their role in the flux estimation problem, the flux bounds are *integrative*, i.e., these constraints integrate many types of genetic and biochemical information into the flux estimation problem.

Flux bounds constrain the values that each reaction in a metabolic network can take. A general model for these bounds is given by:

$$-\delta_{j}\left[V_{max,j}^{\circ}\left(\frac{e}{e^{\circ}}\right)\theta_{j}(\dots)f_{j}(\dots)\right]\leq v_{j}\leq V_{max,j}^{\circ}\left(\frac{e}{e^{\circ}}\right)\theta_{j}(\dots)f_{j}(\dots)$$

where $V_{max,j}^{\circ}$ denotes the maximum reaction velocity (units: flux) computed at some characteristic enzyme abundance (units: concentration), the ratio e/e° is a correction for enzyme abundance (units: dimensioness), $\theta_{j}(\dots)\in[0,1]$ is the fraction of maximial enzyme activity (a function or measurement producing units: dimensionless), and $f_{j}(\dots)$ is a function describing the substrate dependence of the reaction rate j (units: dimensionless). Both $\theta_{j}(\dots)$ and $f_{j}(\dots)$ could have associated parameters, e.g., saturation or binding constants, etc. Finally, the quanity $\delta_{j}\in\{0,1\}$ is a *binary* variable:

- If reaction j is **reversible** $\delta_{j}=1$ or,
- If reaction j is **irreversible** $\delta_{j}=0$

Today, let's focus on approaches for computing the binary direction parameter δ_{j}. However, before we consider the reversibility, let's quickly discuss the other terms.

2.1 A glimpse into the future

In addition to thermodynamics, the flux bounds integrate two levels of biological control:

- **fast** control mechanisms (allosteric) modulate enzyme activity (the biochemical state influences the catalytic rate), and
- **slow** control mechanisms (gene expression) modulate the abundance of system enzymes

The $\theta_j(\dots)$ terms describe the **fast** control mechansisms, while the **slow** control mechanisms are buried in the enzyme abundance e.

Some classics references that we'll discuss (in the future sometime):

- MONOD J, WYMAN J, CHANGEUX JP. ON THE NATURE OF ALLOSTERIC TRANSITIONS: A PLAUSIBLE MODEL. J Mol Biol. 1965 May;12:88-118. doi: 10.1016/s0022-2836(65)80285-6. PMID: 14343300.
- Covert MW, Palsson BO. Constraints-based models: regulation of gene expression reduces the steady-state solution space. J Theor Biol. 2003 Apr 7;221(3):309-25. doi: 10.1006/jtbi.2003.3071. PMID: 12642111.

2.2 Direct Gibbs energy calculation for a single reaction in a closed system

We can compute the reversibility of biochemical reactions by computing the equilibrium (or steady-state in the case of open systems) reaction extent. To do this, we minimize the Gibbs energy for the reaction mixture by searching over the reaction extent ϵ (units: mol) subject to bounds on the extent.

For a single reaction with $\mathcal{M}$ chemical components, the Gibbs energy can be written as the sum of the partial molar Gibbs energies (what we call G-bar) at constant T,P:

$$\hat{G}=\sum_{i=1}^{\mathcal{M}}\bar{G}_{i}n_{i}$$

where $\hat{G}$ denotes the total Gibbs energy for the reaction mixture (units:

kJ), $\bar{G}_i$ denotes the partial molar Gibbs energy for chemical component i (units: kJ/mol) and n_i denotes the number of mols of chemical component i. From classical thermodynamics we know that the partial molar Gibbs energy for chemical species i is given by:

$$\bar{G}_i = G_i^\circ + RT\ln\hat{a}_i$$

where G_i° denotes the Gibbs energy for pure component i at a reference state, and $\hat{a}_i$ denotes the ratio of fugacities (mixture at reaction T,P/pure component i at reference conditions). We can subsitute $\bar{G}_i$ into the expression for $\hat{G}$, and after some algebraic magic, we get:

$$\frac{1}{RT}\left(\hat{G}-\sum_{i=1}^{\mathcal{M}}n_i^\circ G_i^\circ\right) = \epsilon_1\frac{\Delta G^\circ}{RT}+\sum_{i=1}^{\mathcal{M}}n_i\ln\hat{a}_i$$

The first term on the right-hand side is the extent of reaction times the scaled Gibbs energy of reaction; we can think of this term as how much of the Gibbs energy of reaction we recover as the reaction proceeds to the right. The second term describes how the overall Gibbs energy changes as the composition changes (at a fixed T, P). In particular, we know that:

$$n_i = n_i^\circ + \sigma_{i1}\epsilon_1 \qquad i = 1,2,\dots,\mathcal{M}$$

and (for an ideal liquid reaction mixture) we know that $\hat{a}_i = x_i$ where:

$$x_i = \frac{n_i}{\sum_{j=1}^{\mathcal{M}}n_j} \qquad i = 1,2,\dots,\mathcal{M}$$

Thus, we need to search for ϵ_1 (subject to bounds on permissible values of the extent) such that the total Gibbs energy $\hat{G}$ is at a minimum. Once we have the equilibrium extent of reaction, we can compute the equilibrium constant (for an ideal liquid phase reaction at moderate pressures):

$$K_{eq} = \prod_{i=1}^{\mathcal{M}}x_i^{\sigma_{i1}}$$

2.3 Example reaction: phosphoglucose isomerase (PGI)

Phosphoglucose isomerase (PGI) is the second step in glycolysis.

```
# ideal gas cosntant R -
R = (8.314)*(1/1000)*(1/1000) # units: kJ/mmol-K
```

```
# temperature T -
T = 25+273.15   # units: K
```

```
begin

    # number of species and reactions -
    ℳ = 2
    ℛ = 1

    # setup the initial number of mols -
    initial_mol_array = zeros(ℳ,1)
    initial_mol_array[1,1] = 22.0   # units: mmol G6P = 1
    initial_mol_array[2,1] = 0.0    # units: mmol F6P = 2

    # G of formation -
    # we get these numbers from: eQuilibrator
    G_formation_array = zeros(ℳ,1)
    G_formation_array[1,1] = -1304.7*(1/1000)   # units: 
   kJ/mmol G6P
    G_formation_array[2,1] = -1302.1*(1/1000)   # units: 
   kJ/mmol F6P

    # stoichiometric coefficients -
    stoichiometric_array = zeros(ℳ,ℛ)
    stoichiometric_array[1,1] = -1
    stoichiometric_array[2,1] = 1

    # package -
    parameters_dict = Dict{String,Any}()
    parameters_dict["initial_mol_array"] = initial_mol_array
    parameters_dict["G_formation_array"] = G_formation_array
    parameters_dict["stoichiometric_array"] = 
   stoichiometric_array

    # show -
    nothing
end
```

```
function objective_function(epsilon,parameters)

    # get stuff from the parameters dict -
    initial_mol_array = parameters["initial_mol_array"]
    G_formation_array = parameters["G_formation_array"]
    S = parameters["stoichiometric_array"]

    # what extent are we looking at?
    ϵ = sum(initial_mol_array)*epsilon[1]
    RT = R*T

    # compute the ΔG/RT term -
    ΔG_initial_term = ((1/(RT))*sum(S.*G_formation_array))*ϵ

    # compute the current number of mols -
    n_array = initial_mol_array + S*ϵ

    # compute the mol total -
    n_total = sum(n_array)

    # compute the mol fractions for each component -
    ln_x_array = log.((1/n_total)*n_array)

    # compute the second "reaction" term -
    reaction_term = sum(n_array.*ln_x_array)

    # put all together -
    energy_total = ΔG_initial_term + reaction_term

    # return -
    return energy_total
end
```

2.3.1 Method 1: Direct search of the Gibbs energy landscape

```
begin

    # initialize -
    number_of_steps = 200
    gibbs_energy_array = Array{Float64,1}()
    epsilon_range =
   range(0.00001,stop=0.99999,length=number_of_steps) |>
   collect

    # compute -
    for ϵ in epsilon_range

        gibbs_energy_value =
       objective_function([ϵ],parameters_dict);
        push!(gibbs_energy_array,gibbs_energy_value)
    end
end
```

```
begin

    # what is the min value of the G energy and the extent?
    min_G_ind = argmin(gibbs_energy_array)
    min_ϵ_value = epsilon_range[min_G_ind]
    min_G_value = gibbs_energy_array[min_G_ind]

    with_terminal() do
        # show -
        println("Min extent (G,ϵ) = ($(min_G_value) AU,
       $(min_ϵ_value) AU)")
    end
end
```

```
begin
    plot(epsilon_range,
   gibbs_energy_array,lw=3,legend=:topleft, label="Scaled
   Gibbs energy")
    xlabel!("Scaled extent of reaction ϵ [AU]",fontsize=18)
    ylabel!("Scaled Gibbs Energy [AU]",fontsize=18)
end
```

2.3.2 Method 2: Constrained nonlinear optimization problem

Let's recast the Gibbs energy estimation as a (nonlinear) *constrained* optimization problem and solve this problem using the Optim.jl package.

```julia
begin

    # Use the answer from Method 2 as a starting point
    xinitial = [0.65]

    # setup bounds -
    L = 0.00001
    U = 0.99999

    # setup the objective function -
    OF(p) = objective_function(p, parameters_dict)

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

```julia
begin
    bgfs_soln = Optim.minimizer(opt_result)[1]
    with_terminal() do
        println("Optim found ϵ (scaled) = $(bgfs_soln) AU")
    end
end
```

```julia
begin

    # what is the equilibrium extent?
    ϵ_scaled = bgfs_soln;
    ϵ = sum(initial_mol_array)*ϵ_scaled

    # compute the equlibrium constant -
    n_final = initial_mol_array + stoichiometric_array*ϵ

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

    # compute the Keq -
    tmp = dot(stoichiometric_array,ln_x_final)
    K_eq = exp(tmp)

    with_terminal() do
        println("Estimated Keq = $(K_eq)")
    end
end
```

```julia
n_final
```

2.4 Rule of thumb for reversibility

The reversibility parameter can be computed in one of several possible ways. For example, one method in the literature is to use the sign of Gibbs reaction energy:

- if $sgn(\Delta G^{\circ}) < 0$ then $\delta = 0$ (irreversible)
- if $sgn(\Delta G^{\circ}) > 0$ then $\delta = 1$ (reversible)

Alternatively, the value of δ can be assigned based upon a cutoff on the equilibrium constant:

- if $K_{eq} > \star$ then $\delta = 0$ (irreversible)
- if $K_{eq} \leq \star$ then $\delta = 1$ (reversible)

where the metabolic engineer specifies the value $\star$.

3. The eQuilibrator application programming interface (API)

The eQuilibrator application programming interface is a tool for thermodynamic calculations in biological reaction networks. It was developed by the Milo lab at the Weizmann Institute in Rehovot, Israel.

- Beber ME, Gollub MG, Mozaffari D, Shebek KM, Flamholz AI, Milo R, Noor E. eQuilibrator 3.0: a database solution for thermodynamic constant estimation. Nucleic Acids Res. 2022 Jan 7;50(D1): D603-D609. doi: 10.1093/nar/gkab1106. PMID: 34850162; PMCID: PMC8728285.

The eQuilibrator.jl package is a Julia wrapper around eQuilibrator (which is written in Python).

4. Summary and conclusions

In this lecture we:

1. Introduced a general model for the flux bounds constraints in a constraint-based calculation
2. Introduced direct Gibbs energy minimization using the partial molar Gibbs energy
3. Computed the reversibility of a single enzyme-catalyzed reaction

5. Next time

Could we use Direct Gibbs energy calculations to compute the reversibility of many coupled reactions (or perhaps the flux itself)?

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

```julia
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
```

```
html"""
<style>
main {
    max-width: 760px;
    width: 75%;
    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':
```