Notes on Monte Carlo simulations of proteins

May 19, 2016

1 Optimizing the reference energies

1.1 The maximum likelihood solution

We use Monte Carlo to generate a Markov chain of states [1, 2], such that the states are populated according to a Boltzmann distribution. One possible elementary move is a "mutation": we modify the sidechain type $t \to t'$ at a chosen position i in the folded protein, assigning a particular rotamer r' to the new sidechain. At the same time, we perform the reverse mutation in the unfolded protein, $t' \to t$. The corresponding energy change has the form:

$$\Delta E = \Delta E^f - \Delta E^u = \big(E^f(...t'_i, r'_i...) - E^f(...t_i, r_i...)\big) - (E^u(t'_i) - E^u(t_i)) \tag{1}$$

ΔE measures the stability change due to the mutation.

For a particular sequence S, the unfolded state energy has the form:

$$E^u_S = \sum_{i \in S} E^r(t_i). \tag{2}$$

The type-dependent quantities $E^r(t) \equiv E^r_t$ are essential parameters in the simulation model, referred to as "reference energies". Our goal here is to choose them empirically so that the simulation produces amino acid frequencies that match a set of target values, for example experimental values in the Pfam database. Specifically, we will choose them so as to maximize the probability or likelihood of the target sequences.

Let S be a particular sequence. Its Boltzmann probability is

$$p(S) = \frac{1}{Z} \exp(-\beta \Delta G_S), \tag{3}$$

where $\Delta G_S = G^f_S - E^u_S$ is the folding free energy of S, G^f_S is the free energy of the folded form, and Z is a normalizing constant (the partition function). We then have

$$kT \ln p(S) = \sum_{i \in S} E^r(t_i) - G^f_S - kT \ln Z = \sum_{t \in \mathrm{aa}} n_S(t) E^r_t - G^f_S - kT \ln Z, \tag{4}$$

where the sum on the right is over the amino acid types and $n_S(t)$ is the number of amino acids of type t within the sequence S.

We now consider a set $\mathcal{S}$ of N target sequences S; we denote $\mathcal{L}$ the probability of the entire set, which we refer to as their likelihood [3]. We have

$$kT \ln \mathcal{L} = \sum_S \sum_{t \in \mathrm{aa}} n_S(t) E_t^r - \sum_S G_S^f - NkT \ln Z = \sum_{t \in \mathrm{aa}} N(t) E_t^r - \sum_S G_S^f - NkT \ln Z, \quad (5)$$

where $N(t)$ is the number of amino acids of type t in the whole dataset $\mathcal{S}$. The normalization factor or partition function Z is a sum over all possible sequences R:

$$Z = \sum_R \exp(-\beta \Delta G_R) = \sum_R \exp(-\beta \Delta G_R^f) \Pi_{t \in aa} \exp(\beta n_R(t) E_t^r) \quad (6)$$

In view of maximizing $\mathcal{L}$, we consider the derivative of Z with respect to one of the E_t^r:

$$\frac{\partial Z}{\partial E_t^r} = \sum_R \exp(-\beta \Delta G_R^f) \Pi_{s \in aa} \exp(\beta n_R(s) E_s^r) \beta n_R(t) \quad (7)$$

We then have

$$\frac{kT}{Z} \frac{\partial Z}{\partial E_t^r} = \frac{\sum_R n_R(t) \exp(-\beta \Delta G_R)}{\sum_R \exp(-\beta \Delta G_R)} = \langle n(t) \rangle. \quad (8)$$

The quantity on the right is the Boltzmann average of the number $n(t)$ of amino acids t over all possible sequences. In practice, this is the average population of t we would obtain in a long MC simulation. We note that, as usual in statistical mechanics [4], the derivative of $\ln Z$ with respect to one quantity (E_t^r) is equal to the ensemble average of the conjugate quantity ($n_S(t)$).

A necessary condition to maximize $\ln \mathcal{L}$ is that its derivatives with respect to the E_t^r should all be zero. We see that

$$\frac{1}{N} \frac{\partial}{\partial E_t^r} \ln \mathcal{L} = \frac{1}{N} \sum_S n_S(t) - \langle n(t) \rangle = \frac{N(t)}{N} - \langle n(t) \rangle \quad (9)$$

so that

$$\mathcal{L} \text{ maximum} \Longrightarrow \frac{N(t)}{N} = \langle n(t) \rangle, \ \ \forall t \in \mathrm{aa} \quad (10)$$

Thus, to maximize $\mathcal{L}$, we should choose the $\{E_t^r\}$ so that a long simulation gives the same amino acid frequencies as the target database.

1.2 Searching for the maximum

To approach the maximum likelihood, starting from a current guess $\{E_t^r\}$, we should step along the gradient of $\ln \mathcal{L}$, for example using a rule such as [3]

$$E_t^r(n+1) = E_t^r(n) + \delta E \left(n_t^{\exp} - \langle n(t) \rangle_n\right) \tag{11}$$

Here, $n_t^{\exp} = N(t)/N$ is the mean population of type t in the target database; $\langle\rangle_n$ indicates an average over a simulation done using the current reference energies $\{E_t^r\}^{(n)}$, and δE is an empirical constant with the dimension of an energy. This rule differs from the logarithmic increment rule used previously [5, 6].

Optimization methods usually need to calculate the gradient of the cost function but also the function itself. Here, our cost function is $\mathcal{C} = \ln \mathcal{L}$. We can easily compute its gradient (see above) but $\mathcal{C}$ and $\mathcal{L}$ themselves are much harder (impossible) to compute. One idea is to use an auxiliary cost function, such as

$$C = \sum_{t \in \mathrm{aa}} (n_t^{\exp} - \langle n(t) \rangle_n)^2 \tag{12}$$

We might use the $\mathcal{C}$ gradients to increment our $\{E_t^r\}$ values, and C to evaluate convergence. For example, starting from $\{E_t^r\}^{(n)}$, we could run proteus simulations with $\{E_t^r\}^{(n+1)}$ values obtained from Eq. (11), trying 3 or 4 δE values. Based on these 3-4 results, we would then fit $C(\delta E)$ to a simple function (cubic spline) and deduce the best δE and the best $\{E_t^r\}^{(n+1)}$ values. Alternatively, we could just apply Eq. (11) with some plausible δE guess.

References

[1] Frenkel, D., and Smit, B. *Understanding molecular simulation, Chapter 3.* Academic Press, New York, 1996.

[2] Grimmett, G. R., and Stirzaker, D. R. *Probability and random processes.* Oxford University Press, 2001.

[3] Kleinman, C. L., Rodrigue, N., Bonnard, C., Philippe, H., and Lartillot, N. A maximum likelihood framework for protein design. *BMC Bioinf. 7* (2006), Art. 326.

[4] Fowler, R. H., and Guggenheim, E. A. *Statistical Thermodynamics.* Cambridge University Press, 1939.

[5] Schmidt am Busch, M., Lopes, A., Mignon, D., and Simonson, T. Computational protein design: software implementation, parameter optimization, and performance of a simple model. *J. Comput. Chem. 29* (2008), 1092–1102.

[6] Simonson, T., Gaillard, T., Mignon, D., Schmidt am Busch, M., Lopes, A., Amara, N., Polydorides, S., Sedano, A., Druart, K., and Archontis, G. Computational protein design: the Proteus software and selected applications. *J. Comput. Chem. 34* (2013), 2472–2484.