Webnucleo Technical Report: Computational Details Behind libnuceq

Bradley S. Meyer, Tianhong Yu

December 13, 2010

This technical report describes some details of calculations in the libnuceq module.

1 Introduction

libnuceq is a library of C codes for computing nuclear statistical equilibria relevant to nucleosynthesis. It is built on top of libxml, the GNOME C xml toolkit, gsl, the GNU Scientific library, and the Webnucleo.org modules wn_matrix, libnucnet, and libstatmech. Users can compute abundances in arbitrary statistical equilibria, which are specified by an XPath expression. A well-documented API allows users to incorporate libnuceq into their own codes, and examples in the libnuceq distribution demonstrate the API.

2 Abundances

The starting point for libnuceq calculations is the definition of Y_i, the abundance per nucleon of nuclear species i. This is defined as

$$Y_i = \frac{n_i}{\rho N_A} \tag{1}$$

where n_i is the number density of species i, ρ is the mass density, and N_A is Avogadro's number. We relate this to the chemical potential of species i (less its rest mass energy) by

$$Y_i = Y_{Qi} \exp\left\{\frac{\mu_i'}{kT} + f_{corr,i}\right\}. \tag{2}$$

In Eq. (2), μ_i' is the chemical potential of species i less the rest mass energy, k is Boltzmann's constant, T is the temperature, and $f_{corr,i}$, a term that allows for deviations of the abundance from this expression. Y_{Qi} is the quantum abundance per nucleon of species i. It is the abundance of species i per nucleon that would obtain if there were one particle of species i in a cube with side equal to the thermal de Broglie wavelength of species i.

In arbitrary nuclear statistical equilibria, there is a relation between μ_i, the chemical potential of species i and that of the neutrons μ_n and protons μ_p:

$$\mu_i = \Lambda_i + Z_i\mu_p + (A_i - Z_i)\,\mu_n \tag{3}$$

where Λ_i is a quantity or function we call the *prefactor*, Z_i is the atomic number of species i and A_i is its mass number. From Eqs. (2) and (3), we find

$$Y_i = Y_{Qi}\exp\left\{\Lambda_i + Z_i\frac{\mu_p'}{kT} + \left(A_i - Z_i\right)\frac{\mu_n'}{kT} + \frac{B_i}{kT} - \left(Z_i f_{corr,p} + (A_i - Z_i)\,f_{corr,n} - f_{corr,i}\right)\right\}. \tag{4}$$

In this equation, the binding energy B_i is

$$B_i = Z_i m_p c^2 + (A_i - Z_i)\,m_n c^2 - m_i c^2, \tag{5}$$

where $m_p c^2$, $m_n c^2$, and $m_i c^2$ are the rest mass energies of the proton, neutron, and species i, respectively.

3 Calculation of the Equilibria

To solve for equilibria, we must satisfy certain abundance constraints. The first, and the one always relevant under conditions in which the nucleon rest mass energy is much greater than kT, is that nucleon number is conserved:

$$\sum_i A_i Y_i = 1 \tag{6}$$

where the sum runs on all nuclear species. The remaining constraints depend on the particular equilibrium considered.

1. The least constrained equilibrium is weak nuclear statistical equilibrium (WSE). Here $\Lambda_i = 0$ and, if we assume zero chemical potential for all neutrinos,

   $$\mu_p + \mu_e = \mu_n \tag{7}$$

   This allows us to write

   $$\mu_e' = \mu_n' - \mu_p' + \left(m_n c^2 - m_p c^2 - m_e c^2\right) \tag{8}$$

   Since we generally use atomic rather than nuclear masses, this becomes, upon neglect of the binding energy of the electron in the H atom,

   $$\frac{\mu_e'}{kT} = \frac{\mu_n'}{kT} - \frac{\mu_p'}{kT} + \frac{\left(m_n c^2 - m_{^1\mathrm{H}} c^2\right)}{kT} \tag{9}$$

   The additional constraint on WSE is that the abundances must satisfy charge neutrality:

   $$\sum_i Z_i Y_i = Y_e \tag{10}$$

where Y_e is the net number of electrons per nucleon and the sum again runs over nuclear species. We note that for WSE, Y_e is not fixed; rather, it is a function of T and μ_e/kT. The consequence is that we must simultaneously find the roots μ_n/kT and μ_p/kT of the equations

$$f_1 = \sum_i A_i Y_i - 1 \tag{11}$$

and

$$f_2 = \sum_i Z_i Y_i - Y_e \tag{12}$$

where the abundances are computed from Eq. (4) with $\Lambda_i = 0$ and Y_e is computed from the electron chemical potential in Eq. (9).

To solve Eqs. (11) and (12) with libnuceq, one first creates an equilibrium with Libnuceq__new(). The input to this routine is a Libnucnet__Nuc structure, which contains the relevant nuclear data. Libnuceq__new() returns a pointer to an equilibrium structure. To solve for the WSE at a particular temperature and density, one calls Libnuceq__computeEquilibrium(), which takes as input the pointer to the equilibrium and the temperature and density at which to compute the abundances.

libnuceq solves Eqs. (11) and (12) as nested one-dimensional root problems using the Brent solver gsl_root_fsolver_brent in the GNU Scientific Library. We choose nested 1-d roots because it is possible to bracket the roots and thus guarantee a solution. Two-d methods are faster but require a good initial guess. We have opted for robustness over speed. Once the equilibrium has been found, the user may access the results using API routines to get abundances for species or chemical potentials for neutrons, protons, or electrons.

The default routine to compute Y_e from μ_e'/kT is that for fully relativistic, non-interacting electrons. Should the user wish to employ a different equation of state for the electrons, he or she can do so by writing a Libstatmech__Fermion__Function and/or a Libstatmech__Fermion__Integrand for the electron number density. Once these are written, the user then sets them with Libnuceq__updateUserElectronNumberDensity(), which has the prototype

```
Libnuceq__updateUserElectronNumberDensity(
  Libnuceq *self,
  Libstatmech__Fermion__Function my_function,
  Libstatmech__Fermion__Integrand my_integrand,
  void *p_function_data,
  void *p_integrand_data
);
```

Here *self* is a pointer to the equilibrium that will use the number density function my_function and the number density integrand my_integrand. p_function_data and p_integrand_data are pointers to user-defined extra data for the function and the integrand. The default calculation for the electron number density is that for

default libstatmech calculations (non-interacting fully relativistic electrons). If either the function or integrand is set, it will be used in place of the default. To restore the default, the user simply calls Libnuceq__updateUserElectronNumberDensity() with NULL for the function, integrand, and data.

2. The next equilibrium is regular nuclear statistical equilibrium (NSE). Here we assume weak reactions are slow so that the electron fraction does not vary. To compute NSE, then, one again solves Eqs. (11) and (12) but for specified Y_e. In libnuceq, one does this by calling Libnuceq__setYe() before computing the equilibrium. This routine takes as input the pointer to the equilibrium and the value of the particular Y_e at which to compute the NSE. libnuceq then uses the user-specified value of Y_e in Eq. (12). To clear the Y_e constraint (and thereby restore the equilibrium to a WSE), one calls the routine Libnuceq__clearYe(). After the equilibrium has been computed, the user may call API routines to get abundances and chemical potentials.

3. A quasi-statistical equilibrium (QSE) is one for which there is an extra constraint on some subset of nuclei. The most common QSE occurs when the number of heavy nuclei becomes a fixed number because the three-body reactions that assemble them from ^{4}He nuclei become slow (e.g., [1]). The chemical potential for heavy nuclei thus have a uniform shift from their usual NSE relation such that $\Lambda_i = \mu_h/kT$ for all heavy nuclei i, where μ_h is a chemical potential for the heavy nuclei as a whole.

   To compute a QSE with libnuceq, one defines the relevant "cluster", that is, the subset of nuclei with the same μ_h. These nuclei are in equilibrium under the exchange of neutrons and protons. To create equilibrium cluster, the user first creates an equilibrium as in §1 and 2 and then creates a cluster with the routine Libnuceq__newCluster(), which has the prototype

```
Libnuceq__Cluster *
Libnuceq__newCluster(
  Libnuceq *self,
  const char *s_cluster_xpath
);
```

   Here *self* is the equilibrium that will include the cluster and s_cluster_xpath is an XPath expression that defines the cluster using Libnucnet__Nuc variables. For example, one could define the QSE cluster of all heavy nuclei to be carbon isotopes and above; thus, one could include this cluster in the equilibrium pointed to by *p_my_equilibrium* by calling

```
p_cluster =
  Libnuceq__newCluster(
    p_my_equilibrium,
    "[z >= 6]"
  );
```

This routine creates the cluster within *p_my_equilibrium* and returns a pointer to it. The pointer to the cluster may subsequently be retrieved by calling Libnuceq__getCluster(), which gets the cluster by the defining XPath expression. For example, one would retrieve the heavy nuclei cluster above by calling:

```
p_cluster =
  Libnuceq__getCluster(
    p_my_equilibrium,
    "[z >= 6]"
  );
```

Once a cluster is defined, the user then sets the constraint on the function by calling Libnuceq__Cluster__updateConstraint(). The default constraint on a cluster is that the abundances of species within the cluster sum up to a particular value. Thus, for example, if the sum of heavy nuclei is $Y_h = 0.01$, one would call

```
Libnuceq__Cluster__updateConstraint( p_cluster, 0.01 );
```

When we solve for the equilibrium, libnuceq will simultaneously solve Eqs. (11), (12), and

$$f_3 = \sum_{i \in C} Y_i - Y_h \tag{13}$$

for the roots μ'_n/kT, μ'_p/kT and μ_h/kT. In Eq. (13) the sum extends only over species contained in cluster C. Again, once the equilibrium has been computed abundances and chemical potentials, including the chemical potential of the cluster, may be retrieved with API routines.

It is possible to define multiple clusters–one simply calls Libnuceq__newCluster() for each cluster and sets the constraint on each. It is important to note that clusters should not overlap, that is, a species should not belong to more than one cluster. Also, clusters should not include neutrons and protons.

4. A restricted NSE is one for which the NSE extends only over a subset of nuclei. The cluster (subset) C that does not include the neutrons and protons, then, has a $\Lambda_i = A_i\mu_r/kT$ and a cluster constraint equation

$$f_3 = \sum_{i \in C} A_i Y_i - X_r \tag{14}$$

where X_r is the mass fraction of species contained within C. Here both the prefactor Λ_i and the constraint function Eq. (14) differ from the defaults. To specify a different prefactor, the user writes a Libnuceq__Cluster__prefactorFunction, which has the prototype

```
double
my_prefactor_function(
  Libnuceq__Cluster *p_cluster,
  Libnuceq__Species *p_species,
  void *p_my_prefactor_data
);
```

where *p_cluster* is the cluster to which the prefactor function is applied, p_species is the particular nuclear species *i*, and *p_my_prefactor_data* is a pointer to user-defined extra data to the function. The user can then write a Libnuceq__Cluster__constraint_function, which has the prototype

```
double
my_constraint_function(
  Libnuceq__Species *p_species,
  void *p_my_constraint_data
);
```

The names my_prefactor_function and my_constraint_function are for illustration– the user may choose different names that make sense to him or her.

Once the prefactor and constraint functions are written, the user then sets them for an equilibrium with the API routines Libnuceq__Cluster__updatePrefactorFunction() and Libnuceq__Cluster__updateConstraintFunction(). For the examples above, the calls would be, for a cluster *p_my_cluster*,

```
Libnuceq__Cluster__updatePrefactorFunction(
  p_my_cluster,
  Libnuceq__Cluster__prefactorFunction my_prefactor_function,
  p_my_prefactor_data
);
```

and

```
Libnuceq__Cluster__updateConstraintFunction(
  p_my_cluster,
  Libnuceq__Cluster__constraint_function my_constraint_function,
  p_my_constraint_data
);
```

Now when the equilibrium is computed, libnuceq calculates the prefactor and constraint from the user's functions.

Examples in the libnuceq distribution illustrate how to compute equilibria and to set constraints.

4 Setting Correction Factors

libnuceq allows users to correct abundances for deviations ($f_{corr,i}$) away from the simple relation that $Y_i = Y_{Qi}\exp(\mu'/kT)$. The correction is computed by a user-defined Libnucnet__Species__nseCorrectionFactorFunction. The prototype for this function is (see libnucnet documentation for further details)

```
double
my_corr_function(
  Libnucnet__Species *p_species,
  double d_t9,
  double d_rho,
  double d_ye,
  void *p_nse_corr_data
);
```

Here *d_t9* is the temperature in 10^9 K, *d_rho* is the mass density in g/cc, *d_ye* is the electron fraction Y_e, and *p_nse_corr_data* is a pointer to user-defined extra data for the function. The user writes this function and then sets it for a particular equilibrium with Libnuceq__setNseCorrectionFactorFunction(), which has the prototype

```
void
Libnuceq__setNseCorrectionFactorFunction(
  Libnuceq *p_my_equilibrium,
  Libnucnet__Species__nseCorrectionFactorFunction my_corr_function,
  void *p_nse_corr_data
);
```

Now when libnuceq computes the equilibrium, it will compute $f_{corr,i}$ for each species from the user's correction factor function (in the above example, my_corr_function). Note that libnuceq will call the correction factor function with the temperature and density set by the user in calling Libnuceq__computeEquilibrium() and the current Y_e for the equilibrium (whether set by the user with Libnuceq__setYe() or computed from μ_e'/kT). To restore the default (no correction factor function), the user calls Libnuceq__clearNseCorrectionFactorFunction(). Examples in the libnuceq distribution illustrate how to do this.

References

[1] B. S. MEYER, T. D. KRISHNAN, AND D. D. CLAYTON, *Theory of quasi-equilibrium nucleosynthesis and applications to matter expanding from high temperature and density*, ApJ, 498 (1998), pp. 808–830.