THERMUS V2.0
User Guide

Spencer Wheaton

August 2005

0.1 Introduction

With the appropriate choice of ensemble, the statistical-thermal model has proved extremely successful in describing the hadron multiplicities observed in relativistic collisions of both heavy ions and elementary particles, over a wide range of energies. This success motivated the development of THERMUS [1, 2], a thermal model analysis package of C++ classes and functions, for incorporation into the object-oriented ROOT framework [3]. All THERMUS C++ classes inherit from the ROOT base class `TObject`. This allows them to be fully integrated into the interactive ROOT environment, allowing all of the ROOT functionality in a statistical-thermal model analysis. Other publicly available codes performing thermal analyses include SHARE [4] and THERMINATOR [5].

Anyone is free to use THERMUS. However, in the event of work leading to publication, the authors request that THERMUS be cited:

'THERMUS – A Thermal Model Package for ROOT',
S. Wheaton and J. Cleymans, hep-ph/0407174

Since several of the constraining functions in THERMUS use 'Numerical Recipes in C' code (which is under copyright), it is required that THERMUS users have their own copies of this software. Then, with ROOT already installed on your system, follow these steps:

- Download the THERMUS source (in the form of a zipped tar-file) from:

  http://hep.phy.uct.ac.za/THERMUS/

- Unzip and untar the downloaded file.

- Set an environment variable 'THERMUS' to point at the top-level directory containing the THERMUS code.

- Copy the following 'Numerical Recipes in C' functions to `$(THERMUS)/nrc`:

```
broydn.c  rsolv.c
fdjac.c   fmin.c
lnsrch.c  nrutil.c
nrutil.h  qrdcmp.c
qrupdt.c  rotate.c
zbrent.c
```

- Use the makefiles in `$(THERMUS)/functions`, `$(THERMUS)/nrc` and `$(THERMUS)/main` to build the `libFunctions.so`, `libNRCFunctions.so` and `libTHERMUS.so` shared object files (run `make all` in each of these directories).
- Finally, open a ROOT session, load the libraries and begin:

```
[ ]$ root
  ****************************************
  *                                      *
  *        W E L C O M E  to  R O O T     *
  *                                      *
  *   Version   3.10/02  16 February 2004  *
  *                                      *
  *  You are welcome to visit our Web site  *
  *          http://root.cern.ch          *
  *                                      *
  ****************************************

FreeType Engine v2.1.3 used to render TrueType fonts.
Compiled for linux with thread support.

CINT/ROOT C/C++ Interpreter version 5.15.115, Dec 9 2003
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [ ] gSystem->Load("./lib/libFunctions.so");
root [ ] gSystem->Load("./lib/libNRCFunctions.so");
root [ ] gSystem->Load("./lib/libTHERMUS.so");
```

At present, three distinct thermal model formalisms are implemented in THERMUS: the grand-canonical ensemble, in which baryon number (B), strangeness (S), and charge (Q) are conserved on average (the charm quantum number C is also accommodated in this class, although charmed particles are yet to be included in the THERMUS particle list, and the constraining functions involving the charm density are yet to be written); a strangeness-canonical ensemble, in which strangeness is exactly conserved, while B and Q are treated grand-canonically; and, finally, a canonical ensemble, in which B, S and Q are all treated canonically.

Currently, THERMUS performs only chemical analyses. In other words, no kinetic freeze-out analysis or momentum spectra calculations are per-

formed. It is our aim to include such functionality in later versions of THERMUS.

As input to the various thermal model formalisms one needs first a set of particles to be considered thermalised. When combined with a set of thermal parameters, all primordial densities (i.e. number density as well as energy and entropy density and pressure) are calculable. Once the particle decays are known, sensible comparisons can be made with experimentally measured yields.

In THERMUS, the following units are used for the parameters:

Parameter	Unit
Temperature (T)	GeV
Chemical Potential (μ)	GeV
Radius	fm

Quantities frequently output by THERMUS are in the following units:

Quantity	Unit
Number Densities (n)	fm^{-3}
Energy Density (e)	$\mathrm{GeV.fm}^{-3}$
Entropy Density (s)	fm^{-3}
Pressure (P)	$\mathrm{GeV.fm}^{-3}$
Volume (V)	fm^{3}

In the sections to follow, we explain the basic structure and functionality of THERMUS by introducing the major THERMUS classes in a bottom-up approach. We begin with a look at the `TTMParticle` object.[1]

[1]It is a requirement that all ROOT classnames begin with a 'T'. THERMUS classnames begin with 'TTM' for easy identification.

0.2 The `TTMParticle` Class

The properties of a particle applicable to the statistical-thermal model are grouped in the basic `TTMParticle` object:

```
********* LISTING FOR PARTICLE Delta(1600)0 *********

        ID = 32114

        Deg. = 4

        STAT = 1

        Mass            =    1.6 GeV
        Width           =   0.35 GeV
        Threshold       = 1.07454 GeV

        Hard sphere radius = 0

        B = 1
        S = 0           |S| = 0
        Q = 0
        Charm = 0               |C| = 0
        Beauty = 0
        Top = 0

        UNSTABLE

        Decay Channels:

        Summary of Decays:

*************************************************
```

Besides the particle name, 'Delta(1600)0', its Monte Carlo ID is also stored. This provides a far more convenient means of referencing the particle. The particle's decay status is also noted. In this case, the $\Delta(1600)^0$ is considered unstable.

Currently, only the default constructor is written. Particle properties are thus input using the 'setters'.

0.2.1 Inputting and Accessing Particle Decays

The `TTMParticle` class allows also for the storage of a particle's decays. These can be entered from file. As an example, consider the decay file of the $\Delta(1600)^0$:

```
11.67   2112    111
5.83    2212    -211
29.33   2214    -211
3.67    2114    111
22.     1114    211
8.33    2112    113
4.17    2212    -213
15.     12112   111
7.5     12212   -211
```

Each line in the decay file corresponds to a decay channel. The first column lists the branching ratio of the channel, while the subsequent tab-separated integers represent the Monte Carlo ID's of the daughters (each line (channel) can contain any number of daughters). The decay channel list of a `TTMParticle` object is populated with `TTMDecayChannel` objects by the `SetDecayChannels(char* file)` function, with the decay file the argument:

```
root [ ] part->SetDecayChannels("./particles/Delta\(1600\)0_decay.txt")
root [ ] part->List()

        ********* LISTING FOR PARTICLE Delta(1600)0 *********

                ID = 32114

                Deg. = 4
                STAT = 1

                Mass            =    1.6 GeV
                Width           =   0.35 GeV
                Threshold       = 1.07454 GeV

                Hard sphere radius = 0

                B = 1
                S = 0           |S| = 0
                Q = 0
                Charm = 0               |C| = 0
                Beauty = 0
```

```
Top = 0

UNSTABLE

Decay Channels:
BRatio: 0.1167          Daughters:      2112    111
BRatio: 0.0583          Daughters:      2212    -211
BRatio: 0.2933          Daughters:      2214    -211
BRatio: 0.0367          Daughters:      2114    111
BRatio: 0.22            Daughters:      1114    211
BRatio: 0.0833          Daughters:      2112    113
BRatio: 0.0417          Daughters:      2212    -213
BRatio: 0.15            Daughters:      12112   111
BRatio: 0.075           Daughters:      12212   -211

 Summary of Decays:
2112            20%
111             30.34%
2212            10%
-211            42.66%
2214            29.33%
2114            3.67%
1114            22%
211             22%
113             8.33%
-213            4.17%
12112           15%
12212           7.5%
*********************************************
```

In addition to the list of decay channels, a summary list of `TTMDecay` objects is generated in which each daughter appears only once, together with its total decay fraction. This summary list is automatically generated from the decay channel list when the `SetDecayChannels` function is called.

An existing `TList` can be set as the decay channel list of the particle, using the `SetDecayChannels(TList* x)` function. This function calls `UpdateDecaySummary`, thereby automatically ensuring consistency between the decay channel and decay summary lists.

The function `SetDecayChannelEfficiency` sets the reconstruction efficiency of the specified decay channel to the specified percentage. Again, a consistent decay summary list is generated.

Access to the `TTMDecayChannel` objects in the decay channel list is achieved through the `GetDecayChannel` method. If the extracted decay channel is subsequently altered, `UpdateDecaySummary` must be called to ensure consistency of the summary list.

0.2.2 The Destructor

Once the `TTMParticle` destructor is called, all heap-based `TTMDecayChannel` and `TTMDecay` objects in the decay lists are deleted.

0.3 The `TTMParticleSet` Class

The thermalised fireballs considered in statistical-thermal models typically contain approximately 350 different hadron and hadronic resonance species. To facilitate fast retrieval of particle properties, the `TTMParticle` objects of all constituents are stored in a hash table in a `TTMParticleSet` object. Other data members of this `TTMParticleSet` class include the filename used to instantiate the object and the number of particle species. Access to the entries in the hash table is through the particle Monte Carlo ID's. The numerical ID of each particle is converted into a string and stored as the `fName` data member of its associated `TTMParticle` object. This is required, since, in ROOT, access to objects stored in container classes is through `fName`.

0.3.1 Instantiating a `TTMParticleSet` Object

In addition to the default constructor, the following constructors exist:

```
TTMParticleSet *set = new TTMParticleSet(char *file);
TTMParticleSet *set = new TTMParticleSet(TDatabasePDG *pdg);
```

The first constructor instantiates a `TTMParticleSet` object and inputs the particle properties contained in the specified text file. As an example of such a file, `/$THERMUS/particles/PartList_PPB2002.txt` contains a list of all mesons (up to the $K_4^*(2045)$) and baryons (up to the Ω^-) listed in the July 2002 Particle Physics Booklet [6] (195 entries). Only particles need be

included, since the anti-particle properties are directly related to those of the corresponding particle. The required file format is as follows:

```
0       Delta(1600)0    32114   4       +1      1.60000      0       1
0      0      0.35000   1.07454 (npi0)
```

- stability flag (1 for stable, 0 for unstable)
- particle name
- Monte Carlo particle ID (used for all referencing)
- spin degeneracy
- statistics (+1 for Fermi-Dirac, -1 for Bose-Einstein, 0 for Boltzmann)
- mass in GeV
- strangeness
- baryon number
- charge
- absolute strangeness content $|S|$
- width in GeV
- threshold in GeV
- string recording the decay channel from which the threshold is calculated if the particle's width is non-zero

All further particle properties have to be set with the relevant 'setters' (e.g. the charm, absolute charm content and hard-sphere radius). By default, all properties not listed in the particle list file are assumed to be zero.

Figure 1 shows the distribution of resonances (both particle and anti-particle) derived from `/$THERMUS/particles/PartList_PPB2002.txt`. As collider energies increase, so does the need to include also the higher mass resonances. Although the `TTMParticle` class allows for the properties of charmed particles, these particles are not included in the default THERMUS particle list. If required, these particles have to be input by the user. The same applies to the hadrons composed of b and t quarks.

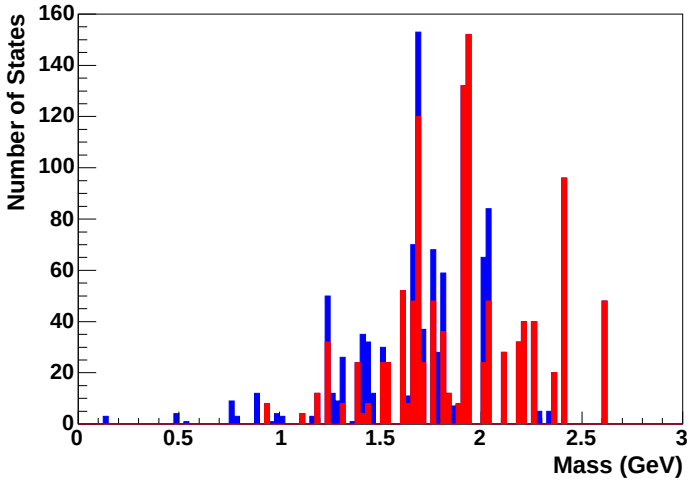

Figure 1: The mass distribution of the resonances included in `/PartList_PPB2002.txt` (blue: mesons, red: baryons).

It is also possible to use a `TDatabasePDG` object to instantiate a particle set[2]. `TDatabasePDG` objects also read in particle information from text files. The default file is `/$ROOTSYS/etc/pdg_table.txt` and is based on the parameters used in PYTHIA6.

```
root [ ] TDatabasePDG *pdg = new TDatabasePDG()
root [ ] pdg->ReadPDGTable()
root [ ] pdg->GetParticle(211)->Print()

pi+  211    Mass:   0.1396 Width (GeV): Stable      Charge:   3.0
 Channel Code BranchingRatio Nd   .............Daughters.............
    0     0     9.99877e-01   2          mu+(-13)      nu_mu(14)
    1     0     1.23000e-04   2           e+(-11)       nu_e(12)

root [ ] TTMParticleSet set(pdg)
root [ ] set.GetParticle(211)->List()

         ********* LISTING FOR PARTICLE pi+ *********

                ID = 211
```

[2]In order to have access to `TDatabasePDG` and related classes, one must first load `/$ROOTSYS/lib/libEG.so`

```
        Deg. = 1

        STAT = -1

        Mass            = 0.13957 GeV
        Width           =      0 GeV
        Threshold       =      0 GeV

        Hard sphere radius = 0

        B = 0
        S = 0           |S| = 0
        Q = 1
        Charm = 0               |C| = 0
        Beauty = 0
        Top = 0

        STABLE
*******************************************
```

The constructor `TTMParticleSet(TDatabasePDG *pdg)` extracts only those particles in the specified `TDatabasePDG` object in particle classes 'Meson', 'CharmedMeson', 'HiddenCharmMeson', 'B-Meson', 'Baryon', 'CharmedBaryon' and 'B-Baryon', as specified in `/$ROOTSYS/etc/pdg_table.txt`, and includes them in the hadron set. Anti-particles must be included in the `TDatabasePDG` object, as they are not automatically generated in this constructor of the `TTMParticleSet` class.

The default file read into the `TDatabasePDG` object, however, is incomplete; the charm, degeneracy, threshold, strangeness, $|S|$, beauty and topness of the particle are not included. Although the `TDatabasePDG::ReadPDGTable` function and default file allow for isospin, I_3, spin, flavor and tracking code to be entered too, the default file does not contain these values. Furthermore, all particles are made stable by default. Therefore, at present, using the `TDatabasePDG` class to instantiate a `TTMParticleSet` class should be avoided, at least until `pdg_table.txt` is improved.

0.3.2 Inputting Decays

Once a particle set has been defined, the decays to the stable particles in the set can be determined. Firstly, let us instantiate a `TTMParticleSet` object and list its stable constituents:

```
root [ ] TTMParticleSet set("./particles/PartList_PPB2002.txt")
root [ ] set.ListStableParticles()


         ************** STABLE PARTICLES **************
                anti-Lambda
                Sigma-
                Omega
                pi0
                Ksi0
                K+
                n
                Sigma+
                anti-Sigma-
                anti-Omega
                K0S
                anti-Ksi0
                Ksi-
                anti-K+
                anti-n
                anti-Sigma+
                pi+
                anti-Ksi-
                p
                anti-pi+
                anti-p
                Lambda
                K0L
         ************************************************
```

This list of stable particles can be modified by adjusting the stability flags of the `TTMParticle` objects included in the `TTMParticleSet` object.

Decays can be input using the `InputDecays(char* dir)` method. Running this function populates the decay lists of all unstable particles in the set, using the decay files listed in the directory specified in the argument. If a file is not found, then the corresponding particle is set to stable. For each typically unstable particle in `/$THERMUS/particles/PartList_PPB2002.txt`, there exists a file in `/$THERMUS/particles` listing its decays. The filename is derived from the particle's name (e.g. `Delta(1600)0__decay.txt` for the $\Delta(1600)^0$). There are presently 195 such files, with entries based on the Particle Physics Booklet of July 2002 [6]. The decays of the corresponding anti-particles are automatically generated, while a private recursive function,

`GenerateBRatios`, is invoked to ensure that only stable particles feature in the decay summary lists.

```
root [ ] set.InputDecays("./particles/")
root [ ] TTMParticle *part = set.GetParticle(32114)
root [ ] part->List()

         ********* LISTING FOR PARTICLE Delta(1600)0 *********

                 ID = 32114

                 Deg. = 4

                 STAT = 1

                 Mass           =    1.6 GeV
                 Width          =   0.35 GeV
                 Threshold      = 1.07454 GeV

                 Hard sphere radius = 0

                 B = 1
                 S = 0          |S| = 0
                 Q = 0
                 Charm = 0              |C| = 0
                 Beauty = 0
                 Top = 0

                 UNSTABLE

                 Decay Channels:
                 BRatio: 0.1167         Daughters:      2112    111
                 BRatio: 0.0583         Daughters:      2212    -211
                 BRatio: 0.2933         Daughters:      2214    -211
                 BRatio: 0.0367         Daughters:      2114    111
                 BRatio: 0.22           Daughters:      1114    211
                 BRatio: 0.0833         Daughters:      2112    113
                 BRatio: 0.0417         Daughters:      2212    -213
                 BRatio: 0.15           Daughters:      12112   111
                 BRatio: 0.075          Daughters:      12212   -211

                 Summary of Decays:
                2112            65.3932%
                111             67.4244%
```

```
        2212            42.4443%
        -211            90.4787%
        211             48.1469%
*****************************************
```

For particle sets based on `TDatabasePDG` objects, decay lists should be populated through the function `InputDecays(TDatabasePDG *)`. This function, however, does not automatically generate the anti-particle decays from those of the particle. Instead, the anti-particle decay list is used. Since the decay list may include electromagnetic and weak decays to particles other than the hadrons stored in the `TTMParticleSet` object, each channel is first checked to ensure that it contains only particles listed in the set. If not, the channel is excluded from the hadron's decay list used by THERMUS. As mentioned earlier, care should be taken when using `TDatabasePDG` objects based on the default file, as it is incomplete.

An extremely useful function is `ListParents(Int_t id)`, which lists all of the parents of the particle with Monte Carlo ID `id`. This function uses `GetParents(TList *parents, Int_t id)`, which populates the list passed with the decays to particle `id`. Note that these parents are not necessarily 'direct parents'; the decays may involve unstable intermediates.

0.3.3 Customising the Set

The `AddParticle` and `RemoveParticle` functions allow customisation of particle sets. Particle and anti-particle are treated symmetrically in the case of the former; if a particle is added, then its corresponding anti-particle is also added. This is not the case for the `RemoveParticle` function, however, where particle and anti-particle have to be removed separately.

Mass-cuts can be performed using `MassCut(Double_t x)` to exclude all hadrons with masses greater than the argument (expressed in GeV). Decays then have to be re-inserted, to exclude the influence of the newly-excluded hadrons from the decay lists.

The function `SetDecayEfficiency` allows the reconstruction efficiency of the decays from a specified parent to the specified daughter to be set. Changes are reflected only in the decay summary list of the parent (i.e. not the decay channel list). Note that running `UpdateDecaySummary` or `GenerateBRatios` will remove any such changes, by creating again a summary list consistent with the channel list.

In addition to these operations, users can input their own particle sets by compiling their own particle lists and decay files.

0.3.4 The Destructor

When the destructor is called, the heap-based `TTMParticle` entries in the hash table are deleted.

0.4 The `TTMParameter` Class

This class groups all relevant information for parameters in the statistical-thermal model. Data members include:

`fName`	- the parameter name,
`fValue`	- the parameter value,
`fError`	- the parameter error,
`fFlag`	- a flag signalling the type of parameter (constrain, fit, fixed, or uninitialised),
`fStatus`	- a string reflecting the intended treatment or action taken.

In addition to these data members, the following, relevant to fit-type parameters, are also included:

`fStart`	- the starting value in a fit,
`fMin`	- the lower bound of the fit-range,
`fMax`	- the upper bound of the fit-range,
`fStep`	- the step-size.

The constructor,

```
TTMParameter *p = new TTMParameter(  TString name, Double_t value,
                                     Double_t error),
```

and `SetParameter(TString name, Double_t value, Double_t error)` function set the parameter to fixed-type, by default. The parameter-type can be modified using the `Constrain`, `Fit` or `Fix` methods.

0.5 The TTMParameterSet Class

The TTMParameterSet class is the base class for all thermal parameter set classes. Each derived class contains its own TTMParameter array, with size determined by the requirements of the ensemble. The base class contains a pointer to the first element of this array. In addition, it stores the constraint information.

All derived classes must contain the function GetRadius. In this way, TTMParameterSet is able to define a function, GetVolume, which returns the volume required to convert densities into total fireball quantities (the volume returned by this function is in units of fm^3).

TTMParameterSetBSQ (applicable to a grand-canonical approach), TTMParameterSetBQ (applicable to a strangeness-canonical approach) and TTMParameterSetCanBSQ (applicable to a fully B, S and Q canonical approach) are the derived classes coded at present.

0.5.1 TTMParameterSetBSQ

This derived class, applicable to the grand-canonical ensemble, contains the parameters:

T	μ_B	μ_S	μ_Q	μ_C	γ_S	γ_C	R

where R is the fireball radius, assuming a spherical fireball (i.e. $V = 4/3\pi R^3$). In addition, the $B/2Q$ ratio and charm and strangeness density of the system are stored here. In the constructor, all errors are defaulted to zero, as is R, μ_C, S/V, C/V and $B/2Q$, while γ_C is defaulted to unity.

Each parameter has a 'getter' (e.g. GetTPar), which returns a pointer to the requested TTMParameter object. In this class, μ_S and μ_Q can be set to constrain-type using ConstrainMuS and ConstrainMuQ, where the arguments are the required strangeness density and $B/2Q$ ratio, respectively. No such function exists for μ_C, since constraining functions are not yet coded for the charm density. Each parameter of this class can be set to fit-type, using functions such as FitT (where the fit parameters have reasonable default values), or fixed-type, using functions such as FixMuB.

0.5.2 `TTMParameterSetBQ`

This derived class, applicable to the strangeness-canonical ensemble (strangeness exactly conserved and B and Q treated grand-canonically), has the parameters:

T	μ_B	μ_Q	γ_S	R_c	R

where R_c is the canonical or correlation radius; the radius inside which strangeness is exactly conserved. The fireball radius R, on the other hand, is used to convert densities into total fireball quantities. In addition, the required $B/2Q$ ratio is also stored, as well as the strangeness required inside the correlation volume (which must be an integer).

In addition to the same 'getters' and 'setters' as the previous derived class, it is possible to set μ_Q to constrain-type by specifying the $B/2Q$ ratio in the argument of `ConstrainMuQ`. The strangeness required inside the canonical volume is set through the `SetS` method. This value is defaulted to zero. The function `ConserveSGlobally` fixes the canonical radius, R_c, to the fireball radius, R. As in the case of the `TTMParameterSetBSQ` class, there also exist functions to set each parameter to fit or fixed-type.

0.5.3 `TTMParameterSetCanBSQ`

This set, applicable to the canonical ensemble with exact conservation of B, S and Q, contains the parameters:

T	B	S	Q	γ_S	R

Since all conservation is exact, there are no chemical potentials to satisfy constraints. Again, the same 'getters', 'setters' and functions to set each parameter to fit or fixed-type exist, as in the case of the previously discussed `TTMParameterSet` derived classes.

0.5.4 Example

As an example, let us define a `TTMParameterSetBQ` object. By default, all parameters are initially of fixed-type. Suppose we wish to fit T and μ_B, and use μ_Q to constrain the $B/2Q$ ratio in the model to that in Pb+Pb collisions:

```
root [ ] TTMParameterSetBQ parBQ(0.160,0.2,-0.01,0.8,6.,6.)
root [ ] parBQ.FitT(0.160)
root [ ] parBQ.FitMuB(0.2)
root [ ] parBQ.ConstrainMuQ(1.2683)
root [ ] parBQ.List()
  ********************* Thermal Parameters **********************

                Strangeness inside Canonical Volume = 0

       T         =          0.16               (to be FITTED)
                                                start: 0.16
                                                range: 0.05 -- 0.18
                                                step:  0.001

     muB         =           0.2               (to be FITTED)
                                                start: 0.2
                                                range: 0 -- 0.5
                                                step:  0.001

     muQ         =         -0.01               (to be CONSTRAINED)

                                                B/2Q: 1.2683

  gammas         =           0.8               (FIXED)

Can. radius      =             6               (FIXED)

  radius         =             6               (FIXED)

                          Parameters unconstrained

  ***************************************************************
```

Note the default parameters for the T and μ_B fits. Obviously, no constraining or fitting can take place yet; we have simply signalled our intent to take these actions at some later stage.

0.6 The `TTMThermalParticle` Class

By combining a `TTMParticle` and `TTMParameterSet` object, a thermal particle can be created. The `TTMThermalParticle` class is the base class from which thermal particle classes relevant to the three currently implemented

thermal model formalisms, `TTMThermalParticleBSQ`, `TTMThermalParticleBQ` and `TTMThermalParticleCanBSQ`, are derived. Since no particle set is specified, the total fireball properties cannot be determined. Thus, in the grand-canonical approach, the constraints cannot yet be imposed to determine the values of the chemical potentials of constrain-type, while, in the strangeness-canonical and canonical formalisms, the canonical correction factors cannot yet be calculated. Instead, at this stage, the chemical potentials and/or correction factors must be specified.

Use is made of the fact that, in the Boltzmann approximation, e, n and P, in the canonical and strangeness-canonical ensembles, are simply the grand-canonical values, with the chemical potential(s) corresponding to the canonically-treated quantum number(s) set to zero, multiplied by a particle-specific correction factor. This allows the functions for calculating e, n and P in the Boltzmann approximation to be included in the base class, which then also contains the correction factor as a data member (by definition, this correction factor is 1 in the grand-canonical ensemble).

Both functions including and excluding resonance width, Γ, are coded (e.g. `DensityBoltzmannNoWidth` and `EnergyBoltzmannWidth`). When width is included, a Breit-Wigner distribution is integrated over between the limits $[\max(m - 2\Gamma, m_{\rm threshold}), m + 2\Gamma]$.

0.6.1 `TTMThermalParticleBSQ`

This class, relevant to the grand-canonical treatment of B, S and Q, has constructor:

```
TTMThermalParticleBSQ(TTMParticle *part, TTMParameterSetBSQ *parm);
```

In addition to the functions for calculating e, n and P in the Boltzmann approximation, defined in the base class, functions implementing quantum statistics for these quantities exist in this derived class (e.g. `EnergyQStatNoWidth` and `PressureQStatWidth`). Additional member functions of this class calculate the entropy using either Boltzmann or quantum statistics, with or without width.

In the functions calculating the thermal quantities assuming quantum statistics, it is first checked that the integrals converge for the bosons (i.e.

there is no Bose-Einstein condensation). The check is performed by the `ParametersAllowed` method. A warning is issued if there are problems and zero is returned.

This class also accommodates charm, since the associated parameter set includes μ_C and γ_C, while the associated particle may have non-zero charm.

Note: The chemical potentials μ_S and μ_Q are not automatically constrained in this class.

0.6.2 `TTMThermalParticleBQ`

This class, relevant to the strangeness-canonical ensemble, has constructor:

```
TTMThermalParticleBQ(  TTMParticle *part, TTMParameterSetBQ *parm,
                       Double_t corr);
```

At present, this class is only applied in the Boltzmann approximation. Under this assumption, n, e and P are given by the grand-canonical result, with μ_S set to zero, up to a multiplicative correction factor. Since the total entropy does not split into the sum of particle entropies, no entropy calculation is made in this class.

Note: The chemical potential μ_Q is not automatically constrained and the canonical correction factor must be specified.

0.6.3 `TTMThermalParticleCanBSQ`

This class, relevant to the fully canonical treatment of B, S and Q, has constructor:

```
TTMThermalParticleCanBSQ(  TTMParticle *part,
                           TTMParameterSetCanBSQ *parm,
                           Double_t corr);
```

At present, as in the case of `TTMThermalParticleBQ`, this class is only applied in the Boltzmann approximation. Also, since the total entropy again does not split into the sum of particle entropies, no entropy calculation is made here.

Note: The canonical correction factor must be specified.

0.6.4 Example

Let us make a thermal particle, within the strangeness-canonical ensemble, from the $\Delta(1600)^0$ and the parameter set previously defined. Since this particle has zero strangeness, a correction factor of 1 is passed as the third argument of the constructor:

```
root [ ] TTMThermalParticleBQ therm_delta(part,&parBQ,1.)
root [ ] therm_delta.DensityBoltzmannNoWidth()
(Double_t)8.15072671710089913e-04
root [ ] therm_delta.EnergyBoltzmannWidth()
(Double_t)2.29185316377137748e-03
```

0.7 The `TTMThermalModel` Class

Once a parameter and particle set have been specified, these can be combined into a thermal model. `TTMThermalModel` is the base class from which the `TTMThermalModelBSQ`, `TTMThermalModelBQ` and `TTMThermalModelCanBSQ` classes are derived. A string descriptor is included as a data member of the base class to identify the type of model. This is used, for example, to handle the fact that the number of parameters in the associated parameter sets is different, depending on the model type.

All derived classes define functions to calculate the primordial particle, energy and entropy densities, as well as the pressure. These thermal quantities are stored in a hash table of `TTMDensObj` objects. Again, access is through the particle ID's. In addition to the individual particles' thermal quantities, the total primordial fireball strangeness, baryon, charge, charm, energy, entropy, and particle densities, pressure, and Wròblewski factor (see Section 0.7.11) are included as data members.

At this level, the constraints on any chemical potentials of constrain-type can be imposed, and the correction factors in canonical treatments can be determined. Also, as soon as the primordial particle densities are known, the decay contributions can be calculated.

0.7.1 Calculating Particle Densities

Running `GenerateParticleDens` clears the current entries in the density hash table of the `TTMThermalModel` object, automatically constrains the

chemical potentials (where applicable), calculates the canonical correction factors (where applicable), and then populates the density hash table with a `TTMDensObj` object for each particle in the associated set. The decay contributions to each stable particle are also calculated, so that the density hash table contains both primordial and decay particle density contributions, provided of course that the decays have been entered in the associated `TTMParticleSet` object. In addition, the Wròblewski factor and total strangeness, baryon, charge, charm and particle densities in the fireball are calculated.

Note: The summary decay lists of the associated `TTMParticleSet` object are used to calculate the decay contributions. Hence, only stable particles have decay contributions reflected in the hash table. Unstable particles that are themselves fed by higher-lying resonances, do not receive a decay contribution.

Each derived class contains the private function `PrimPartDens`, which calculates only the primordial particle densities and, hence, the canonical correction factors, where applicable. In the case of the grand-canonical and strangeness-canonical ensembles, this function calculates the densities without automatically constraining the chemical potentials of constrain-type first. The constraining is handled by `GenerateParticleDens`, which calls external friend functions, which, in turn, call `PrimPartDens`. In the purely canonical ensemble, `GenerateParticleDens` simply calls `PrimPartDens`. In this way, there is uniformity between the derived classes. Since there is no constraining to be done, there is no real need for a separate function in the canonical case.

0.7.2 Calculating Energy and Entropy Densities and Pressure

`GenerateEnergyDens`, `GenerateEntropyDens` and `GeneratePressure` iterate through the existing density hash table and calculate and insert, respectively, the primordial energy density, entropy density and pressure of each particle in the set. In addition, they calculate the total primordial energy density, entropy density and pressure in the fireball, respectively. These functions require that the density hash table already be in existence. In other words, `GenerateParticleDens` must already have been run. If the parameters have subsequently changed, then this function must be run yet again to recalculate the correction factors or re-constrain the parameters, as required.

0.7.3 Bose-Einstein Condensation

When quantum statistics are taken into account (e.g. in `TTMThermalModelBSQ` or for the non-strange particles in `TTMThermalModelBQ`), certain choices of parameters lead to diverging integrals for the bosons (Bose-Einstein condensation). In these classes, a check, based on `TTMThermalParticleBSQ::ParametersAllowed`, is included to ensure that the parameters do not lead to problems. Including also the possibility of incomplete strangeness and/or charm saturation (i.e. $\gamma_S \neq 1$ and/or $\gamma_C \neq 1$), Bose-Einstein condensation is avoided, provided that,

$$e^{(m_i-\mu_i)/T} > \gamma_S^{|S_i|}\gamma_C^{|C_i|}, \tag{1}$$

for each boson. If this condition is failed to be met for any of the bosons in the set, a warning is issued and the densities are not calculated.

0.7.4 Accessing the Thermal Densities

The entries in the density hash table are accessed using the particle Monte Carlo ID's. The function `GetDensities(Int_t ID)` returns the `TTMDensObj` object containing the thermal quantities of the particle with the specified ID. The primordial particle, energy, and entropy densities, pressure, and decay density are extracted from this object using the `GetPrimDensity`, `GetPrimEnergy`, `GetPrimEntropy`, `GetPrimPressure`, and `GetDecayDensity` functions of the `TTMDensObj` class, respectively. The sum of the primordial and decay particle densities is returned by `TTMDensObj::GetFinalDensity`. `TTMDensObj::List` outputs to screen all thermal densities stored in a `TTMDensObj` object.

`ListStableDensities` lists the densities (primordial and decay contributions) of all those particles considered stable in the particle set associated with the model. Access to the total fireball densities is through separate 'getters' defined in the `TTMThermalModel` base class (e.g. `GetStrange`, `GetBaryon` etc.).

0.7.5 Further Functions

`GenerateDecayPartDens` and `GenerateDecayPartDens(Int_t id)` (both defined in the base class) calculate decay contributions to stable particles. The

former iterates through the density hash table and calculates the decay contributions to all those particles considered stable in the set. The latter calculates just the contribution to the stable particle with ID `id`. In both cases, the primordial densities must be calculated first. In fact, `GenerateParticleDens` automatically calls `GenerateDecayPartDens`, so that this function does not have to be run separately under ordinary circumstances. However, if one is interested in investigating the effect of decays, while keeping the parameters (and hence the primordial densities) fixed, then running these functions is best (the hash table will not be repeatedly cleared and repopulated with the same primordial densities).

`ListDecayContributions(Int_t d_id)` lists the contributions (in percentage and absolute terms) of decays to the daughter with ID `d_id`. The primordial and decay densities must already appear in the density hash table (i.e. run `GenerateParticleDens` first). `ListDecayContribution(Int_t p_id, Int_t d_id)` lists the contribution of the decay from the specified parent (with ID `p_id`) to the specified daughter (with ID `d_id`). The percentages listed by each of these functions are those of the individual decays to the total decay density.

0.7.6 `TTMThermalModelBSQ`

This class has constructor:

```
TTMThermalModelBSQ(  TTMParticleSet *particles,
                     TTMParameterSetBSQ *parameters,
                     Bool_t qstats = true, Bool_t width = true)
```

In the grand-canonical ensemble, quantum statistics can be employed and, hence, there is a flag specifying whether to use Fermi-Dirac and Bose-Einstein statistics or Boltzmann statistics. As can be seen, the constructor, by default, includes both the effect of quantum statistics and resonance width. The flags controlling their inclusion are set using the `SetQStats` and `SetWidth` functions, respectively. The functions that calculate the particle, energy, and entropy densities, and pressure then use the corresponding functions in the `TTMThermalParticleBSQ` class to calculate these quantities in the required way. The statistics data member (`fStat`) of each `TTMParticle` included in the associated set can be used to fine-tune the inclusion of quantum statistics; with the quantum statistics flag switched on, Boltzmann statistics are

still used for those particles with `fStat=0`.

In this ensemble, at this stage, both μ_S and μ_Q can be constrained (either separately or simultaneously). In order to accomplish this, the μ_S and/or μ_Q parameters in the associated `TTMParameterSetBSQ` object must be set to constrain-type.

It is also possible to constrain μ_B by the primordial ratio E/N (the average energy per hadron), $n_b + n_{\bar{b}}$ (the total primordial baryon plus anti-baryon density), or s/T^3 (the primordial, temperature-normalised entropy density). This is accomplished by the `ConstrainEoverN`, `ConstrainTotalBaryonDensity` and `ConstrainSoverT3` methods, respectively. Running these functions will adjust μ_B such that E/N, $n_b + n_{\bar{b}}$ or s/T^3, respectively, has the required value, regardless of the parameter type of μ_B.

This class also accommodates charm, since the associated parameter set includes μ_C and γ_C, while the associated particle set may contain charmed particles. However, no constraining functions have yet been written for the charm content within this ensemble.

Excluded Volume Effects

Within the grand-canonical ensemble, it is possible to include excluded volume effects. Their inclusion is controlled by the `fExclVolCorrection` flag, false by default, which is set through the `SetExcludedVolume` function. When included, these corrections are calculated on calling `GenerateParticleDens`, based on the hard-sphere radii stored in the `TTMParticle` objects of the associated particle set.

0.7.7 `TTMThermalModelBQ`

This class contains the following additional data members:

`flnZtot`	- log of the total partition function,
`flnZ0`	- log of the non-strange component of the partition function,
`fExactMuS`	- equivalent strangeness chemical potential,
`fCorrP1`	- canonical correction for S = +1 particles,
`fCorrP2`	- canonical correction for S = +2 particles,
`fCorrP3`	- canonical correction for S = +3 particles,

`fCorrM1` - canonical correction for S = -1 particles,
`fCorrM2` - canonical correction for S = -2 particles,
`fCorrM3` - canonical correction for S = -3 particles,

and has constructor:

```
TTMThermalModelBQ(  TTMParticleSet *particles,
                    TTMParameterSetBQ *parameters,
                    Bool_t width = true)
```

Although this ensemble is only applied in the Boltzmann approximation for $S \neq 0$ hadrons, it is possible to apply quantum statistics to the $S = 0$ hadrons. This is achieved through the `SetNonStrangeQStats` function. By default, quantum statistics is included for the non-strange hadrons by the constructors. Resonance width can be included for all hadrons, and is achieved through the `SetWidth` function. The constructors, by default, apply resonance width. The functions that calculate the particle, energy, and entropy densities, and pressure then use the corresponding functions in the `TTMThermalParticle` classes to calculate these quantities in the required way.

`GenerateParticleDens` populates the density hash table with particle densities, including the canonical correction factors, which are also stored in the appropriate data members. The equivalent strangeness chemical potential is calculated from the canonical correction factor for $S = +1$ particles. In the limit of large VT^3, this approaches the value of μ_S in the equivalent grand-canonical treatment.

Running `GenerateEntropyDens` populates each `TTMDensObj` object in the hash table with only that part of the total entropy that can be unambiguously attributed to that particular particle. There is a term in the total entropy that cannot be split; this is added to the total entropy at the end, but not included in the individual entropies (i.e. summing up the entropy contributions of each particle will not give the total entropy).

At this stage, in this formalism, μ_Q can be constrained (this is automatically realised if this parameter is set to constrain-type), while the correlation radius (R_c) can be set to the fireball radius (R) by applying the function `ConserveSGlobally` to the associated `TTMParameterSetBQ` object.

It is also possible to constrain μ_B by the primordial ratio E/N (the average energy per hadron), $n_b + n_{\bar{b}}$ (the total primordial baryon plus anti-baryon

density), or s/T^3 (the primordial, temperature-normalised entropy density) using the `ConstrainEoverN`, `ConstrainTotalBaryonDensity` or `ConstrainSoverT3` methods, respectively. Running these functions will adjust μ_B such that E/N, $n_b + n_{\bar{b}}$ or s/T^3, respectively, has the required value, regardless of the parameter type of μ_B.

The strangeness content within the correlation volume, as calculated by THERMUS in the strangeness-canonical ensemble requiring strangeness neutrality, is shown in Figure 2, for $\mu_B = 0$ and $\mu_B = 0.240$ GeV (in both cases, $\mu_Q = 0$ and $\gamma_S = 1$). As one can see, the strangeness constraint is satisfied far better in the case of the vanishing baryon chemical potential. One notices, too, that the constraint worsens as the temperature and/or correlation radius increase.

0.7.8 `TTMThermalModelCanBSQ`

This class has constructor:

```
TTMThermalModelCanBSQ(  TTMParticleSet *particles,
                        TTMParameterSetCanBSQ *parameters,
                        Bool_t width = true);
```

and contains, amongst others, the following data members:

`flnZtot`	- log of the total canonical partition function,
`fMuB,fMuS,fMuQ`	- equivalent chemical potentials,
`fCorrpip`	- correction for π^+-like particles,
`fCorrpim`	- correction for π^--like particles,
`fCorrkm`	- correction for K^--like particles,
`fCorrkp`	- correction for K^+-like particles,
`fCorrk0`	- correction for K^0-like particles,
`fCorrak0`	- correction for $\bar{K}^0$-like particles,
`fCorrproton`	- correction for p-like particles,
`fCorraproton`	- correction for $\bar{p}$-like particles,
`fCorrneutron`	- correction for n-like particles,
`fCorraneutron`	- correction for $\bar{n}$-like particles,
`fCorrlambda`	- correction for Λ-like particles,
`fCorralambda`	- correction for $\bar{\Lambda}$-like particles,
`fCorrsigmap`	- correction for Σ^+-like particles,
`fCorrasigmap`	- correction for $\bar{\Sigma}^-$-like particles,
`fCorrsigmam`	- correction for Σ^--like particles,

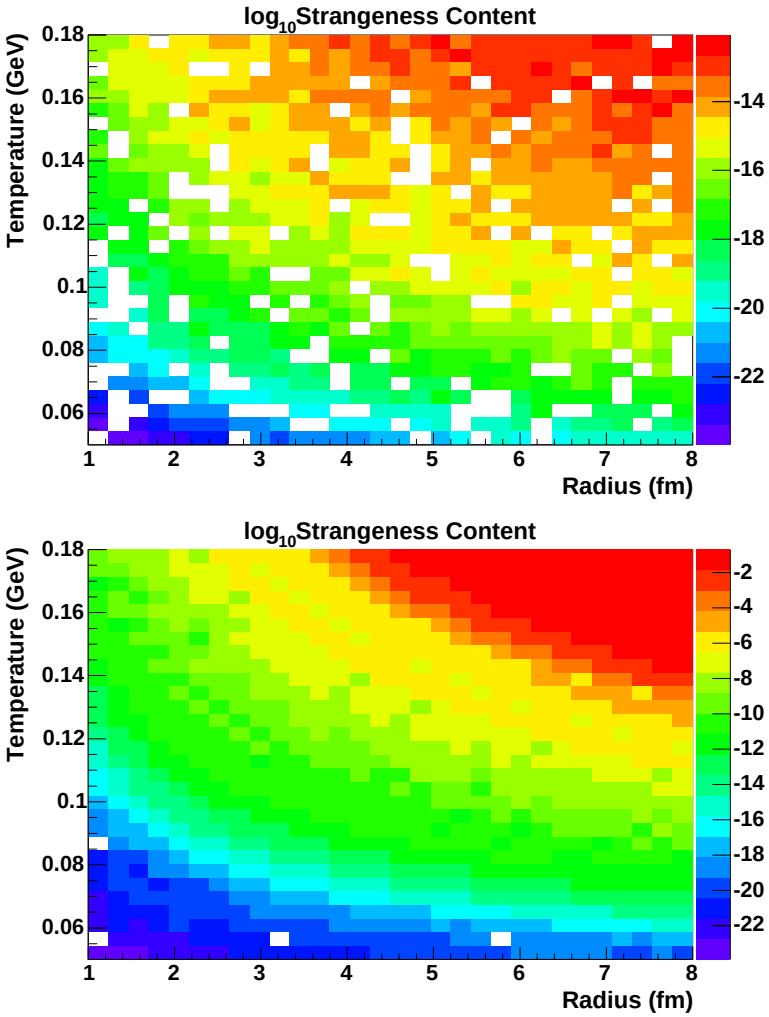

Figure 2: The strangeness content in the correlation volume, as a function of temperature and correlation radius, achieved by THERMUS in the strangeness-canonical ensemble imposing strangeness neutrality, with $\mu_Q = 0$ and $\gamma_S = 1$, for $\mu_B = 0$ (top) and $\mu_B = 0.240$ GeV (bottom).

`fCorrasigmam` - correction for $\bar{\Sigma}^+$-like particles,
`fCorrdeltam` - correction for Δ^--like particles,
`fCorradeltam` - correction for $\bar{\Delta}^+$-like particles,
`fCorrdeltapp` - correction for Δ^{++}-like particles,
`fCorradeltapp` - correction for $\bar{\Delta}^{--}$-like particles,
`fCorrksim` - correction for Ξ^--like particles,
`fCorraksim` - correction for $\bar{\Xi}^+$-like particles,
`fCorrksi0` - correction for Ξ^0-like particles,
`fCorraksi0` - correction for $\bar{\Xi}^0$-like particles,
`fCorromega` - correction for Ω^--like particles,
`fCorraomega` - correction for $\bar{\Omega}^+$-like particles.

Since this ensemble is only applied in the Boltzmann approximation, there is no flag for quantum statistics. However, resonance width can be included. This is achieved through the `SetWidth` function. The constructor, by default, applies resonance width. The functions that calculate the particle, energy, and entropy densities, and pressure then use the corresponding functions in the `TTMThermalParticle` classes to calculate these quantities in the required way.

`GenerateParticleDens` calls `PrimPartDens`, which calculates the particle densities, including the canonical correction factors, which are then also stored in the relevant data members accessible through the `GetCorrFactor` method. The integrands featuring in the evaluation of the partition function and correction factors can be viewed after calling `PopulateZHistograms`. This function populates the array passed as argument with histograms showing these integrands as a function of the integration variables ϕ_S and ϕ_Q. Since these histograms are created off of the heap, they must be cleaned up afterwards.

Running `GenerateEntropyDens` populates each `TTMDensObj` object in the density hash table with only that part of the total entropy that can be unambiguously attributed to that particular particle. There is a term in the total entropy that cannot be split; this is added to the total entropy at the end, but not included in the individual entropies (i.e. summing up the entropy contributions of each particle will not give the total entropy).

Figures 3 and 4 show, as a function of temperature T and fireball radius R, the degree to which the baryon, strangeness and charge contents in the model agree with that required, for $B = 2$, $S = 0$ and $Q = 2$ (as in pp collisions) and $B = 24$, $S = 0$ and $Q = 12$ (as in fully-central $C + C$ collisions),

respectively. As is evident from the figures, the constraints on the quantum contents of the system are well enforced, particularly at low temperatures and for small systems.

0.7.9 Example

As an example, we consider the strangeness-canonical ensemble, based on the particle set and strangeness-canonical parameter set previously defined. After instantiating the object, we populate the hash table with primordial and decay particle densities:

```
root [ ] TTMThermalModelBQ modBQ(&set,&parBQ)
root [ ] modBQ.GenerateParticleDens()
root [ ] parBQ.List()

  ********************* Thermal Parameters **********************

                Strangeness inside Canonical Volume = 0

       T         =          0.16               (to be FITTED)
                                                start: 0.16
                                                range: 0.05 -- 0.18
                                                step:  0.001

     muB         =           0.2               (to be FITTED)
                                                start: 0.2
                                                range: 0 -- 0.5
                                                step:  0.001

     muQ         =       -0.00636409           (*CONSTRAINED*)

                                                B/2Q: 1.2683

  gammas         =           0.8               (FIXED)

Can. radius      =             6               (FIXED)

  radius         =             6               (FIXED)

                          B/2Q Successfully Constrained

  ***************************************************************
```

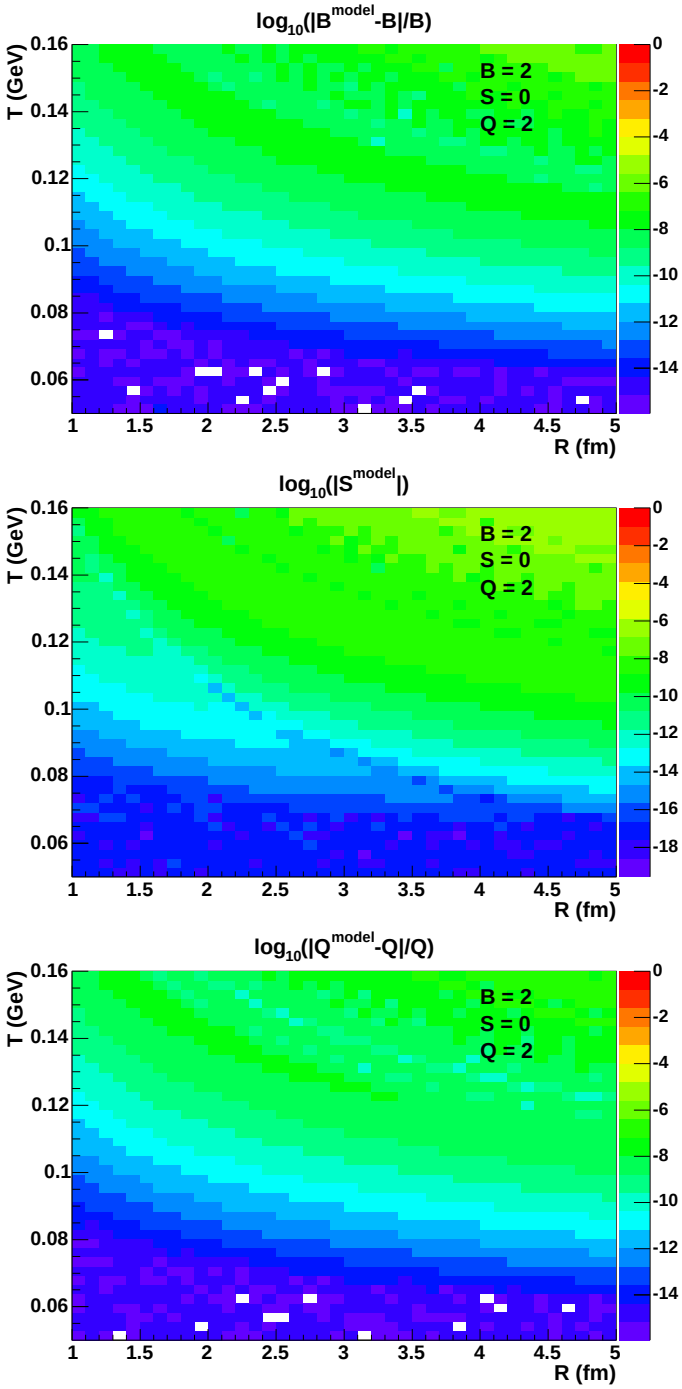

Figure 3: The accuracy, as a function of the fireball radius R and temperature T, with which THERMUS constrains the baryon, strangeness and charge contents, with $B = 2$, $S = 0$ and $Q = 2$, assuming $\gamma_S = 1$.

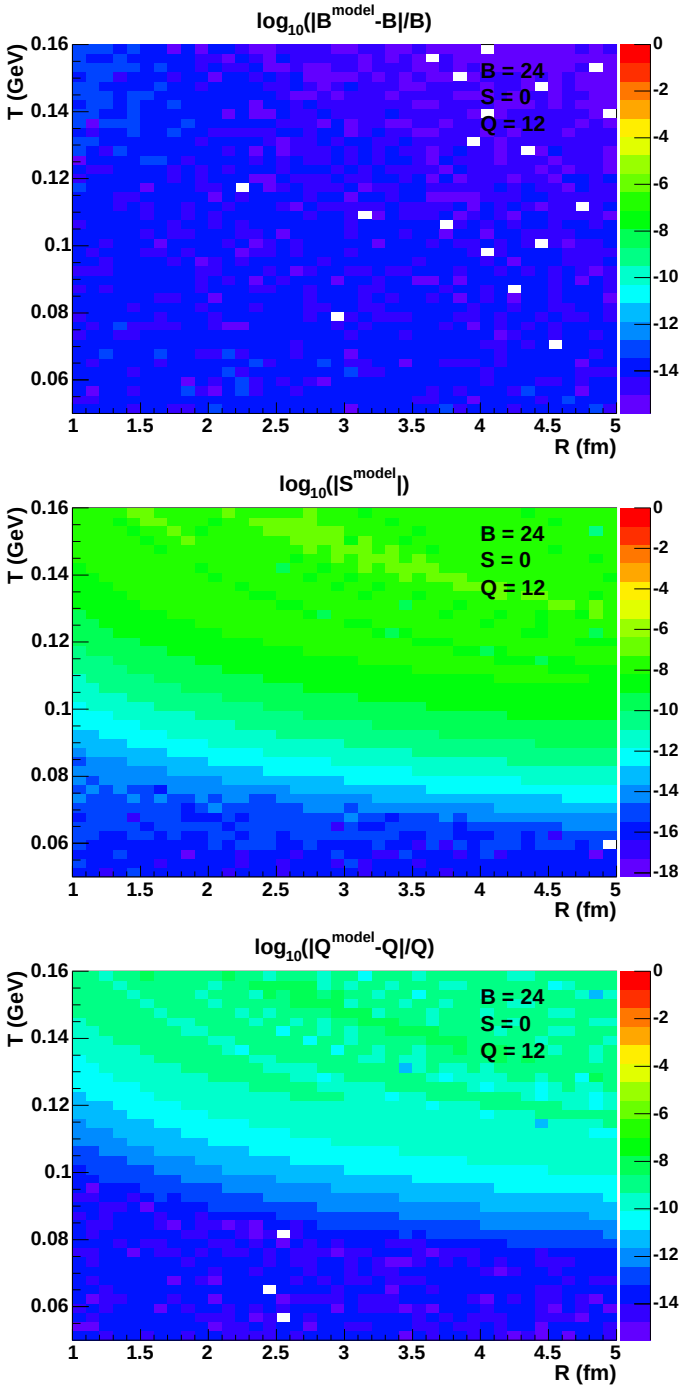

Figure 4: The accuracy, as a function of the fireball radius R and temperature T, with which THERMUS constrains the baryon, strangeness and charge contents, with $B = 24$, $S = 0$ and $Q = 12$, assuming $\gamma_S = 1$.

One notices that the constraint on μ_Q is now automatically imposed.

The energy and entropy densities and pressure can be calculated once `GenerateParticleDens` has been run:

```
root [ ] modBQ.GenerateEnergyDens()
root [ ] modBQ.GenerateEntropyDens()
root [ ] modBQ.GeneratePressure()
root [ ] modBQ.ListInfo()

  ****************************************************************
  ********************** Thermal Model Info **********************

         Particle set:
                ./particles/PartList_PPB2002.txt

         Quantum statistics for S=0 hadrons
         Boltzmann statistics for strange hadrons
         Resonance width included


  ********************** Thermal Parameters **********************

                 Strangeness inside Canonical Volume = 0

       T         =          0.16               (to be FITTED)
                                                start: 0.16
                                                range: 0.05 -- 0.18
                                                step:  0.001

     muB         =           0.2               (to be FITTED)
                                                start: 0.2
                                                range: 0 -- 0.5
                                                step:  0.001

     muQ         =       -0.00636409           (*CONSTRAINED*)

                                                B/2Q: 1.2683

  gammas         =           0.8               (FIXED)

Can. radius      =             6               (FIXED)
```

```
radius         =             6               (FIXED)

                     B/2Q Successfully Constrained

****************************************************************

******************** Thermal Quantities ***********************

       S required in canonical volume:          0

       S in canonical volume (model) = -0.00319327

       B/2Q     =     1.2683  (constraint :  1.2683)

       lambda_s = 0.437324

       Primordial Densities:
                              n = 0.452036
                              e = 0.463067
                              s = 3.21852

       ************** STABLE PARTICLES **************
              anti-Lambda
              Sigma-
              Omega
              pi0
              Ksi0
              K+
              n
              Sigma+
              anti-Sigma-
              anti-Omega
              K0S
              anti-Ksi0
              Ksi-
              anti-K+
              anti-n
              anti-Sigma+
              pi+
              anti-Ksi-
              p
              anti-pi+
              anti-p
```

```
            Lambda
            K0L
        *******************************************

  ****************************************************************
  ****************************************************************
```

One sees listed the properties of the fireball (S inside the canonical volume, $B/2Q$, and the Wròblewski factor (λ_S), as well as the total particle, energy and entropy primordial densities). Now, suppose that we are interested in the thermal densities of the $\Delta(1600)^0$ and π^+:

```
root [ ] TTMDensObj *delta_dens = modBQ.GetDensities(32114)
root [ ] delta_dens->List()

  **** Densities for Particle 32114 ****
         n_prim = 0.00138306
         n_decay = 0
         e_prim = 0.0022912
         s_prim = 0.0139745
         p_prim = 0.000221328

root [ ] TTMDensObj *piplus_dens = modBQ.GetDensities(211)
root [ ] piplus_dens->List()

  **** Densities for Particle 211 ****
         n_prim = 0.0488139
         n_decay = 0.114392
         e_prim = 0.0247039
         s_prim = 0.20276
         p_prim = 0.00742708
```

One notices that the π^+ has a decay density contribution, while the $\Delta(1600)^0$ does not. This is because, unlike the $\Delta(1600)^0$, the π^+ is considered stable.

0.7.10 Imposing of Constraints

The 'Numerical Recipes in C' function applying the Broyden globally convergent secant method of solving nonlinear systems of equations is employed in THERMUS to constrain parameters. The input to the Broyden method is a vector of functions for which roots are sought. Typically, in the thermal model, solutions to the following equations are required (either separately or

simultaneously):

$$\begin{aligned}
\left(\frac{B/V}{2Q/V}\right)^{model}_{primordial} - \left(\frac{B}{2Q}\right)^{colliding\ system} &= 0, \\
S^{model}_{primordial} - S^{colliding\ system} &= 0, \\
\left(\frac{E/V}{N/V}\right)^{model}_{primordial} - \left(\frac{E}{N}\right)^{required} &= 0.
\end{aligned}$$

Although, as written, these equations are correct, the quantities $B/2Q$, S and E/N are typically of different orders of magnitude. Since the Broyden method in 'Numerical Recipes in C' defines just one tolerance level for function convergence (`TOLF`), it is important to 'normalise' each equation:

$$\begin{aligned}
\left\{\left(\frac{B/V}{2Q/V}\right)^{model}_{primordial} - \left(\frac{B}{2Q}\right)^{colliding\ system}\right\} \Big/ \left(\frac{B}{2Q}\right)^{colliding\ system} &= 0, \\
\left\{S^{model}_{primordial} - S^{colliding\ system}\right\} / S^{colliding\ system} &= 0, \\
\left\{\left(\frac{E/V}{N/V}\right)^{model}_{primordial} - \left(\frac{E}{N}\right)^{required}\right\} \Big/ \left(\frac{E}{N}\right)^{required} &= 0.
\end{aligned}$$

This is the most democratic way of treating all constraints equally. However, this method obviously fails in the event of one of the denominators being zero. For the equations considered above, this is only likely in the case of the strangeness constraint, where the initial strangeness content is typically zero. In this case, where the strangeness carried by the positively strange particles S_+ is balanced by the strangeness carried by the negatively strange particles S_-, we write as our function to be satisfied,

$$(S/V)^{model}_{primordial} \Big/ \left(|S_+|^{model}_{primordial}/V + |S_-|^{model}_{primordial}/V\right) = 0.$$

In this way, the constraints can be satisfied to equal relative degrees, and equally well fractionally at each point in the parameter space. In addition to the constraints listed above, THERMUS also allows for the constraining of the total baryon plus anti-baryon density and the temperature-normalised entropy density, s/T^3.

0.7.11 Calculation of the Wròblewski Factor

The Wròblewski factor [7] is defined as,

$$\lambda_S \;=\; \frac{2 < s\bar{s} >}{< u\bar{u} > + < d\bar{d} >},$$

where $< u\bar{u} > + < d\bar{d} >$ is the sum of newly-produced $u\bar{u}$ and $d\bar{d}$ pairs, while all $s\bar{s}$ pairs are newly-produced if $S = 0$ in the initial state.

In THERMUS, λ_S is calculated in the following way:

- Using the primordial particle densities and the strangeness content of each particle listed in the particle hash table, the $s+\bar{s}$ and $u+d+\bar{u}+\bar{d}$ densities are determined.

- Assuming $S = 0$, $\#s = \#\bar{s}$, and so the density of newly-produced $s\bar{s}$ pairs is simply $(s + \bar{s})/2$.

- From baryon number conservation, the net baryon content in the system, n_B, originates from the initial state. Thus, $3 \times n_B$ must correspond to the density of $u + d$ quarks brought in by the colliding nuclei. This is subtracted from the total $u + d + \bar{u} + \bar{d}$ density to yield the density of newly-produced non-strange light quarks.

- Since $\#s = \#\bar{s}$ and, amongst newly-produced non-strange light quarks, $u+d = \bar{u}+\bar{d}$, further assuming that $\mu_Q = 0$ implies that $u = \bar{u} = d = \bar{d}$. This allows the density of $u\bar{u}$ and $d\bar{d}$ pairs to be easily determined.

0.8 Thermal Fit Preliminaries

Often a single experiment releases yields and ratios that contain different feed-down corrections. Each yield or ratio then has a different decay chain associated with it. Since `TTMThermalModel` objects allow for just one associated particle set, they do not allow sufficient flexibility for performing thermal fits to experimental data. Instead, `TTMThermalFit` classes had to be developed. Before we discuss these classes, let us look at the `TTMYield` object, which forms an essential part of the `TTMThermalFit` class.

0.8.1 TTMYield

Information relating to both yields and ratios of yields can be stored in `TTMYield` objects. These objects contain the following data members:

`fName`	- the name of the yield or ratio,
`fID1`	- the ID of the yield or numerator ID in the case of a ratio,
`fID2`	- denominator ID in the case of a ratio (0 for a yield),
`fFit`	- true if the yield or ratio is to be included in a fit (else predicted),
`fSet1`	- particle set relevant to yield or numerator in case of ratio,
`fSet2`	- particle set relevant to denominator in case of ratio (0 for yield),
`fExpValue`	- the experimental value,
`fExpError`	- the experimental error,
`fModelValue`	- the model value,
`fModelError`	- the model error.

This class has the following constructor,

```
TTMYield(  TString name, Double_t exp_val, Double_t exp_err,
           Int_t id1, Int_t id2 = 0, Bool_t fit = true)
```

By default, `TTMYield` objects are set for inclusion in fits. The functions `Fit` and `Predict` control the fit-status of a `TTMYield` object. Particle sets (decay chains) are assigned using the `SetPartSet` method.

The functions `GetStdDev` and `GetQuadDev` return the number of standard and quadratic deviations between model and experimental values, respectively, i.e.,

$$\text{(Model Value} - \text{Exp. Value)/Exp. Error,}$$

and,

$$\text{(Model Value} - \text{Exp. Value)/Model Value,}$$

respectively, while `List` outputs the contents of a `TTMYield` object to screen. Access to all remaining data members is through the relevant 'getters' and 'setters'.

0.9 The TTMThermalFit Class

This is the base class from which the TTMThermalFitBSQ, TTMThermalFitBQ and TTMThermalFitCanBSQ classes are derived. Each TTMThermalFit object contains:

- a particle set, the so-called base set, which contains all of the constituents of the hadron gas, as well as the default decay chain to be used,
- a parameter set,
- a list of TTMYield objects containing yields and/or ratios of interest,
- data members storing the total χ^2 and quadratic deviation, and
- a TMinuit fit object.

A string descriptor is also included in the base class to identify the type of model on which the fit is based. This is used, for example, to determine the number of parameters in the associated parameter sets.

Each derived class defines a private function, GenerateThermalModel, which creates (off the heap) a thermal model object, based on the base particle set and parameter set of the TTMThermalFit object, with the specific quantum statistics/resonance width/excluded volume requirements, where applicable.

0.9.1 Populating and Customising the List of Yields of Interest

The list of yields and/or ratios of interest can be input from file using the function InputExpYields, provided that the file has the following format:

```
333     Exp_A    0.02    0.01
-211    211     Exp_B  0.990   0.100
-211    211     Exp_C  0.960   0.177
321     -321    Exp_C  1.152   0.239
```

where the first line corresponds to a yield, and has format:

```
Yield ID /t Descriptor string /t Exp.  Value /t Exp.  Error/n
```

while the remaining lines correspond to ratios, and have format:

```
Numerator ID /t Denominator ID /t Descriptor string /t Exp.  Value
/t Exp.  Error/n
```

A `TTMYield` object is created off the heap for each line in the file, with a name derived from the ID's and the descriptor. This name is determined by the private function `GetName`, which uses the base particle set to convert the particle ID's into particle names and appends the descriptor. In addition to all of the Monte Carlo particle ID's in the associated base particle set, the following THERMUS-defined identifiers are also allowed:

- ID = 1: N_{part},
- ID = 2: h^-,
- ID = 3: h^+.

By default, each `TTMYield` object inserted in the list of yields of interest by the method `InputExpYields` is assigned the base particle set and set for inclusion in fits.

A `TTMYield` object can also be added to the list using `AddYield`. Such yields should, however, have names that are consistent with those added by the `InputExpYields` method; the `GetName` function should be used to ensure this consistency. Only yields with unique names can be added to the list, since it is this name which allows retrieval of the `TTMYield` objects from the list. If a yield with the same name already exists in the list, a warning is issued. The inclusion of descriptors ensures that `TTMYield` objects can always be given unique names.

`RemoveYield(Int_t id1,Int_t id2,TString descr)` removes from the list and deletes the yield with the name derived from the specified ID's and descriptor by `GetName`. The `GetYield(Int_t id1,Int_t id2,TString descr)` method returns the required yield.

0.9.2 Generating Model Values

Values for each of the yields of interest listed in a `TTMThermalFit` object are calculated by the function `GenerateYields`. This method uses the current parameter values and assigned particle sets to calculate these model values.

`GenerateYields` firstly calculates the primordial particle densities of all constituents listed in the base particle set. This it does by creating the relevant `TTMThermalModel` object from the base particle set and the parameters, and then calling `GenerateParticleDens`. In this way, the density hash table of the newly-formed `TTMThermalModel` object is populated with primordial densities, as well as decay contributions, according to the base particle set (recall that `GenerateParticleDens` automatically calculates decay contributions in addition to primordial ones). `GenerateYields` then iterates through the list of `TTMYield` objects, calculating their specific decay contributions. New model values are then inserted into these `TTMYield` objects. In addition, the total χ^2 and quadratic deviation are calculated, based solely on the `TTMYield` objects which are of fit-type. `ListYields` lists all `TTMYield` objects in the list.

0.9.3 Performing a Fit

The `FitData(Int_t flag)` method initiates a fit to all experimental yields or ratios in the `TTMYield` list which are of fit-type. With `flag=0`, a χ^2 fit is performed, while `flag=1` leads to a quadratic deviation fit. In both cases, `fit_function` is called. This function determines which parameters of the associated parameter set are to be fit, and performs the required fit using the ROOT `TMinuit` fit class. On completion, the list of `TTMYield` objects contains the model values, while the parameter set reflects the best-fit parameters. Model values are calculated by the `GenerateYields` method. For each `TTMYield` object in the list, a model value is calculated– even those that have been chosen to be excluded from the actual fit. In this way, model predictions can be determined at the same time as a fit is performed. `ListMinuitInfo` lists all information relating to the `TMinuit` object, following a fit.

0.9.4 `TTMThermalFitBSQ`

The constructor,

```
TTMThermalFitBSQ(  TTMParticleSet *set, TTMParameterSetBSQ *par,
                   char *file)
```

instantiates an object with the specified base particle set and parameter set and inputs the yields listed in the specified file in the `TTMYield` list.

The specifics of the fit, i.e. the treatment of quantum statistics, resonance width and excluded volume corrections, are handled through the `SetQStats`, `SetWidth` and `SetExclVol` methods, respectively. By default, both resonance width and quantum statistics are included, while excluded volume corrections are excluded.

0.9.5 `TTMThermalFitBQ`

The constructor,

```
TTMThermalFitBQ(  TTMParticleSet *set, TTMParameterSetBQ *par,
                  char *file)
```

instantiates an object with the specified base particle set and parameter set and inputs the yields listed in the specified file in the `TTMYield` list.

The specifics of the fit, i.e. the treatment of resonance width and quantum statistics for the non-strange hadrons, are handled through the `SetWidth` and `SetNonStrangeQStats` methods, respectively. By default, both resonance width and quantum statistics for the non-strange hadrons are included.

0.9.6 `TTMThermalFitCanBSQ`

The constructor,

```
TTMThermalFitCanBSQ(  TTMParticleSet *set, TTMParameterSetCanBSQ *par,
                      char *file)
```

instantiates an object with the specified base particle set and parameter set and inputs the yields listed in the specified file in the `TTMYield` list.

The inclusion of resonance width in the fit is handled through the `SetWidth` method. By default, resonance width is included.

0.9.7 Example

As an example, consider a fit to fictitious particle ratios measured in Au+Au collisions at some energy. We will assume a grand-canonical ensemble, with the parameters T, μ_B and μ_S fitted, and μ_Q fixed to zero. In the grand-canonical ensemble, ratios are independent of the fireball radius (this is not true in the canonical ensemble). For this reason, there is no need to specify the treatment of the radius. Furthermore, we will ignore the effects of resonance width and quantum statistics.

We begin by instantiating a particle set object, based on the particle list distributed with THERMUS. After inputting the particle decays, a parameter set is defined:

```
root [ ] TTMParticleSet set("./particles/PartList_PPB2002.txt")
root [ ] set.InputDecays("./particles/")
root [ ] TTMParameterSetBSQ par(0.160,0.05,0.,0.,1.)
root [ ] par.List()
  ********** Thermal Parameters **********

       T         =          0.16            (FIXED)

     muB         =          0.05            (FIXED)

     muS         =             0            (FIXED)

     muQ         =             0            (FIXED)

  gammas         =             1            (FIXED)

  radius         =             0            (FIXED)

     muC         =             0            (FIXED)

  gammac         =             1            (FIXED)

                      Parameters unconstrained

  ************************************************************
```

One notices that all parameters are, by default, of fixed-type.

Next, we change the parameters T, μ_B and μ_S to fit-type, supplying sensible starting values as the arguments to the appropriate functions. For

all other properties of the fit (step size, fit range etc.), we accept the default values:

```
root [ ] par.FitT(0.160)
root [ ] par.FitMuB(0.05)
root [ ] par.FitMuS(0.)
root [ ] par.List()

  ********************* Thermal Parameters **********************

       T          =          0.16             (to be FITTED)
                                                start: 0.16
                                                range: 0.05 -- 0.18
                                                step:  0.001

     muB          =          0.05             (to be FITTED)
                                                start: 0.05
                                                range: 0 -- 0.5
                                                step:  0.001

     muS          =             0             (to be FITTED)
                                                start: 0
                                                range: 0 -- 0.5
                                                step:  0.001

     muQ          =             0             (FIXED)

  gammas          =             1             (FIXED)

  radius          =             0             (FIXED)

     muC          =             0             (FIXED)

  gammac          =             1             (FIXED)

                        Parameters unconstrained

  ***************************************************************
```

Next, we prepare a file ('ExpData.txt') containing the experimental data:

```
      -211    211     Exp_A   0.990   0.100
      -211    211     Exp_B   0.960   0.177
      -211    211     Exp_D   1.000   0.022
```

```
321     -321    Exp_B   1.152   0.239
321     -321    Exp_D   1.098   0.111
321     -321    Exp_C   1.108   0.022
-2212   2212    Exp_A   0.650   0.092
-2212   2212    Exp_B   0.679   0.148
-2212   2212    Exp_D   0.600   0.072
-2212   2212    Exp_C   0.714   0.050
-3122   3122    Exp_B   0.734   0.210
-3122   3122    Exp_C   0.720   0.024
-3312   3312    Exp_C   0.878   0.054
-3334   3334    Exp_C   1.062   0.410
```

As one can see, there are multiple occurrences of the same particle–anti-particle combination. This is why additional descriptors are required. In this case, the descriptors list the particular experiment responsible for the measurement. In other situations, the descriptors may describe whether feed-down corrections have been employed or some other relevant detail that, together with the ID's, uniquely identifies each yield or ratio.

We are now in a position to create a `TTMThermalFitBSQ` object based on the newly-instantiated parameter and particle sets and the data file. Since quantum statistics and resonance width are included by default, we have to explicitly turn these settings off:

```
root [ ] TTMThermalFitBSQ fit(&set,&par,"ExpData.txt")
root [ ] fit.SetQStats(kFALSE)
root [ ] fit.SetWidth(kFALSE)
root [ ] fit.ListYields()

*******************************

  anti-pi+/pi+ Exp_A:
                       FIT YIELD
                       Experiment:     0.99     +-    0.1
  anti-pi+/pi+ Exp_B:
                       FIT YIELD
                       Experiment:     0.96     +-  0.177
  anti-pi+/pi+ Exp_D:
                       FIT YIELD
                       Experiment:        1     +-  0.022
    K+/anti-K+ Exp_B:
                       FIT YIELD
                       Experiment:    1.152     +-  0.239
```

```
    K+/anti-K+ Exp_D:
                        FIT YIELD
                        Experiment:    1.098     +-  0.111
    K+/anti-K+ Exp_C:
                        FIT YIELD
                        Experiment:    1.108     +-  0.022
      anti-p/p Exp_A:
                        FIT YIELD
                        Experiment:     0.65     +-  0.092
      anti-p/p Exp_B:
                        FIT YIELD
                        Experiment:    0.679     +-  0.148
      anti-p/p Exp_D:
                        FIT YIELD
                        Experiment:      0.6     +-  0.072
      anti-p/p Exp_C:
                        FIT YIELD
                        Experiment:    0.714     +-   0.05
anti-Lambda/Lambda Exp_B:
                        FIT YIELD
                        Experiment:    0.734     +-   0.21
anti-Lambda/Lambda Exp_C:
                        FIT YIELD
                        Experiment:     0.72     +-  0.024
anti-Ksi-/Ksi- Exp_C:
                        FIT YIELD
                        Experiment:    0.878     +-  0.054
anti-Omega/Omega Exp_C:
                        FIT YIELD
                        Experiment:    1.062     +-   0.41
  ******************************************************************
```

One can see that all ratios are set for inclusion in the fit (i.e. each is a 'FIT YIELD'). By default, all ratios are assigned the same decay chain as the base particle set of the thermal fit object. This can be changed, if required, by assigning a specific particle set to the numerator and denominator of a ratio, using the `TTMYield::SetPartSet` method.

Next, let us simply generate the model values corresponding to each of the `TTMYield` objects in the list, based on the current parameters. Part of the output of `ListYields` is shown here:

```
root [ ] fit.GenerateYields()
root [ ] fit.ListYields()
*******************************

                                -
                                -
                                -

    K+/anti-K+ Exp_B:
                        FIT YIELD
                        Experiment:    1.152     +-  0.239
                        Model:      0.986478     +-      0
                        Std.Dev.: -0.692562  Quad.Dev.: -0.167791

    K+/anti-K+ Exp_D:
                        FIT YIELD
                        Experiment:    1.098     +-  0.111
                        Model:      0.986478     +-      0
                        Std.Dev.: -1.00471  Quad.Dev.: -0.113051

    K+/anti-K+ Exp_C:
                        FIT YIELD
                        Experiment:    1.108     +-  0.022
                        Model:      0.986478     +-      0
                        Std.Dev.: -5.52375  Quad.Dev.: -0.123188

      anti-p/p Exp_A:
                        FIT YIELD
                        Experiment:     0.65     +-  0.092
                        Model:      0.535261     +-      0
                        Std.Dev.: -1.24716  Quad.Dev.: -0.21436

                                -
                                -
                                -
```

Each experimental measurement now has a corresponding model value, shown together with its χ^2 and quadratic deviation. The total χ^2 and quadratic deviation are also easily obtained:

```
root [ ] fit.GetChiSquare()
(Double_t)1.50228928916394807e+02
root [ ] fit.GetQuadDev()
(Double_t)1.93739598445467975e+00
```

Suppose, for some reason, that we wish to exclude Experiment D's K^+/K^- ratio from the future fit:

```
root [ ] fit.GetYield(321,-321,"Exp_D")->Predict()
root [ ] fit.GenerateYields()
root [ ] fit.GetChiSquare()
(Double_t)1.49219493574695520e+02
root [ ] fit.GetQuadDev()
(Double_t)1.92461541998607411e+00
```

One sees that the total χ^2 and quadratic deviation are modified (the predicted ratio is excluded from their calculation). This ratio is still included in the listing though:

```
root [ ] fit.ListYields()
*******************************

                                -
                                -
                                -

    K+/anti-K+ Exp_B:
                        FIT YIELD
                        Experiment:    1.152     +-  0.239
                        Model:      0.986478     +-      0
                        Std.Dev.: -0.692562  Quad.Dev.: -0.167791

    K+/anti-K+ Exp_D:
                        PREDICTED YIELD
                        Experiment:    1.098     +-  0.111
                        Model:      0.986478     +-      0
                        Std.Dev.: -1.00471  Quad.Dev.: -0.113051

    K+/anti-K+ Exp_C:
                        FIT YIELD
                        Experiment:    1.108     +-  0.022
                        Model:      0.986478     +-      0
                        Std.Dev.: -5.52375  Quad.Dev.: -0.123188

      anti-p/p Exp_A:
                        FIT YIELD
                        Experiment:     0.65     +-  0.092
                        Model:      0.535261     +-      0
                        Std.Dev.: -1.24716  Quad.Dev.: -0.21436
```

```
                                -
                                -
                                -
```

Finally, we perform a χ^2 fit:

```
root [ ] fit.FitData(0)

                                -
                                -
                                -

 *************************** FITTING ***************************

                        T =    0.159361        (** FITTING **)
                      muB =   0.0345583        (** FITTING **)
                      muS =   0.0099537        (** FITTING **)
                      muQ =           0                (FIXED)
                   gammas =           1                (FIXED)
                   radius =           0                (FIXED)
                      muC =           0                (FIXED)
                   gammac =           1                (FIXED)

              Parameters unconstrained

 ************************************************************

       ****************** ChiSquare = 3.63571 *****************

              S/V     =      0.00215014
              B/2Q    =      0.841173

       New Minimum!

                        T =    0.159361        (** FITTING **)
                      muB =   0.0345583        (** FITTING **)
                      muS =   0.0099537        (** FITTING **)
                      muQ =           0                (FIXED)
                   gammas =           1                (FIXED)
                   radius =           0                (FIXED)
                      muC =           0                (FIXED)
                   gammac =           1                (FIXED)

              Parameters unconstrained
```

```
  ****************************************************************

                                    -
                                    -
                                    -
```

Once completed, the associated parameter set contains the best-fit values for the fit parameters:

```
root [ ] fit.GetParameterSet()->List()
  ********************** Thermal Parameters **********************

       T       =       0.159      +-   0.118264      (FITTED!)
                                                      start: 0.16
                                                      range: 0.05 -- 0.18
                                                      step:  0.001

     muB       =     0.0344416    +-   0.0251108     (FITTED!)
                                                      start: 0.05
                                                      range: 0 -- 0.5
                                                      step:  0.001

     muS       =     0.00991061   +-   0.00956142    (FITTED!)
                                                      start: 0
                                                      range: 0 -- 0.5
                                                      step:  0.001

     muQ       =           0                          (FIXED)

  gammas       =           1                          (FIXED)

  radius       =           0                          (FIXED)

     muC       =           0                          (FIXED)

  gammac       =           1                          (FIXED)

                        Parameters unconstrained

  ****************************************************************
```

All other details of the fit are output to screen by the `ListMinuitInfo` function:

```
root [ ] fit.ListMinuitInfo()
                        FCN = 3.63561
                        EDM = 6.99938e-06
                        Errdef = 1
                Full accurate covariance matrix calculated
 FCN=3.63561 FROM MIGRAD    STATUS=CONVERGED     167 CALLS         168 TOTAL
                     EDM=6.99938e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   INTERNAL      INTERNAL
  NO.   NAME      VALUE            ERROR          STEP SIZE       VALUE
   1  T            1.59000e-01   1.18264e-01   2.20944e-04   7.43567e-01
   2  muB          3.44416e-02   2.51108e-02   1.69163e-05  -1.03966e+00
   3  muS          9.91061e-03   9.56142e-03   1.86392e-05  -1.28828e+00
 EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  3    ERR DEF=1
  9.504e-03  2.451e-03  9.225e-04
  2.451e-03  6.390e-04  2.404e-04
  9.225e-04  2.404e-04  9.200e-05
 PARAMETER  CORRELATION COEFFICIENTS
       NO.  GLOBAL      1      2      3
        1  0.99474   1.000  0.995  0.986
        2  0.99675   0.995  1.000  0.992
        3  0.99167   0.986  0.992  1.000
```

Bibliography

[1] S. Wheaton and J. Cleymans, hep-ph/0407174.

[2] S. Wheaton and J. Cleymans, J. Phys. G: Nucl. Part. Phys. **31** (2005) S1069.

[3] R. Brun and F. Rademakers, Nucl. Inst. & Meth. in Phys. Res. A **389** (1997) 81.
See also http://root.cern.ch/.

[4] G. Torrieri, S. Steinke, W. Broniowski, W. Florkowski, J. Letessier, J. Rafelski, nucl-th/0404083.

[5] A. Kisiel, T. Tałuć, W. Broniowksi and W. Florkowski, nucl-th/0504047.

[6] K. Hagiwara *et al.*, Phys. Rev. D **66** (2002) 010001.

[7] K. Wròblewski, Acta Physica Polonica **B16** (1985) 379.