Creating a Maxwell-Boltzmann particle distribution

Stanley Humphries, Ph.D.

Field Precision LLC
E mail: techinfo@fieldp.com
Internet: https://www.fieldp.com

A common task in computer calculations of collective physics is the generation of input particles with a desired probability distribution in energy, angle or position. This article discusses some easy methods to accomplish the task using a software module to manipulate tabular functions. As an example, we'll consider creating a Maxwell-Boltzmann distribution in kinetic energy U.

As a preliminary, let's review some probability theory. A distribution of particles with respect to a quantity like energy is represented as a *probability density*. The probability density $p(x)$ is defined as follows: the probability of observing a value of x in the interval $x_0 - \Delta x/2$ to $x_0 + \Delta x/2$ is approximately

$$p(x_0)\Delta x \tag{1}$$

The *marginal probability* is given by

$$P(x) = \int_{x_{min}}^{x} p(x')dx'. \tag{2}$$

Figure 1 shows the relationship of the two functions. If $p(x)$ is a normalized function, then the marginal probability has the range $0.0 \leq P(x) \leq 1.0$. The function gives the probability of observing a value of x between x_{min} and a specified maximum value. The marginal probability has another interpretation illustrated by the shaded region of Fig. 1. The interval along the $P(x)$ axis represents a fraction of the particles (5% in the figure). These particles are distributed along an interval of the x axis with a length inversely proportional to the slope of $P(x)$, or $1/p(x)$. The particle density along x is low where $p(x)$ is small and high where $p(x)$ is large. In other words, a uniform distribution along the vertical axis maps into a non-uniform distribution along the horizontal axis with density proportional to $p(x)$.

The observation suggests the following procedure to create N model particles with probability density $p(x)$. First, find N uniformly-distributed values of the variable ζ in the interval $0.0 \leq \zeta \leq 1.0$. Second, use the inverse of the marginal probability function to create N particles with x values

$$x = P^{-1}(\zeta). \tag{3}$$

There are potential problems applying the procedure in a code:

- The probability density $p(x)$ may not be normalized or integrable.
- There may be no closed form to express the inverse of the marginal probability.
- The probability density may extend to infinity.

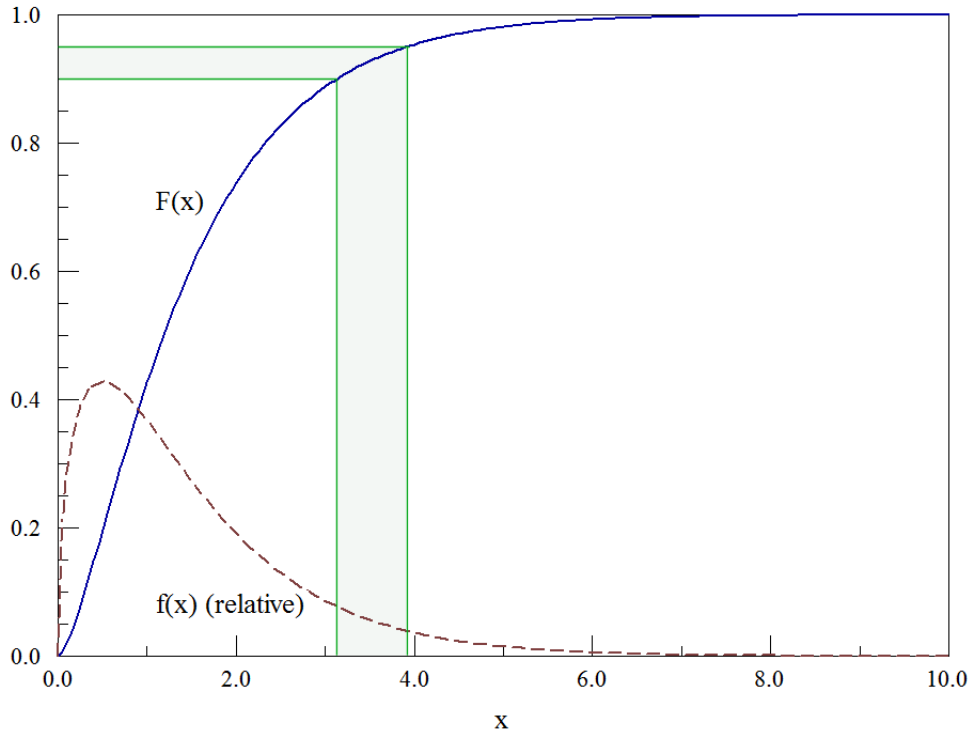

Figure 1: Relationship of the probability density and the marginal probability for a Maxwell-Boltzmann energy distribution with $x = U/kT$.

All three items apply to the Maxwell-Boltzmann distribution. The first two issues can be resolved by using a numerical approach that does not involve closed-form expressions. To resolve the third issue, it is necessary to truncate distributions at a finite value.

Tabular functions are a useful tool for generating particle distributions. My introduction to the concept was in the program **SciMath** developed by Robert Kares. It was easy-to-use software from the DOS era that was the precursor to programs like **Mathematica** and **MathCad**. A *tabular function* in **SciMath** was a set of discrete values of an independent variable x_i and a dependent variable $y_i(x_i)$. By using a variety of numerical techniques, the program made interpolations and calculated integrals and derivatives. To the user, tabular functions could be employed as though they were continuous functions. I wrote a module to define tabular function variables and to perform operations almost two decades ago and have applied it to innumerable applications since then. There is a complete description of the module at the end of this article. The source code is available by request.

Let's see how to apply tabular functions to create a Maxwell-Boltzmann energy distribution. The probability distribution of kinetic energy scales as

$$p(x) \sim \sqrt{x}\ \exp(-x), \tag{4}$$

where $x = U/kT$ for the temperature T in °K. My general approach is to do one-time setup calculations with a spreadsheet and to incorporate repetitive operations in the code. The first column in my spreadsheet contained 200 values of x over the range $0.0 \le x \le 10.0$. The second column contained non-normalized values of $p(x)$ from Eq. 4. The third column was the definite integral of $p(x)$ using the trapezoid rule. Finally, the fourth column was equal to the third column divided by the maximum value of the integral:

$$P(x) = \frac{\int_0^x\ p(x')dx'}{\int_0^{10}\ p(x')dx'}. \tag{5}$$

This quantity represents the normalized marginal probability for a truncated Maxwell-Boltzmann distribution.

The spreadsheet values can be transferred to a code with data statements or via a text file. They are assembled into a tabular function with the `PUTXY` subroutine. The trick to defining the inverse function is to use the $P(x)$ values as the independent variable and the x values as the dependent variable. This is one of the outstanding features of tabular functions. An inverse function is formed simply by exchanging the independent and dependent values. This is possible as long as the function is single-valued (as in Fig. 1).

For each model particle, the code generates a random value of ζ in the range $0.0 \le \zeta \le 1.0$. It then employs the `VALUE` subroutine to determine x from the P^{-1} table. This value is multiplied by kT to determine the kinetic

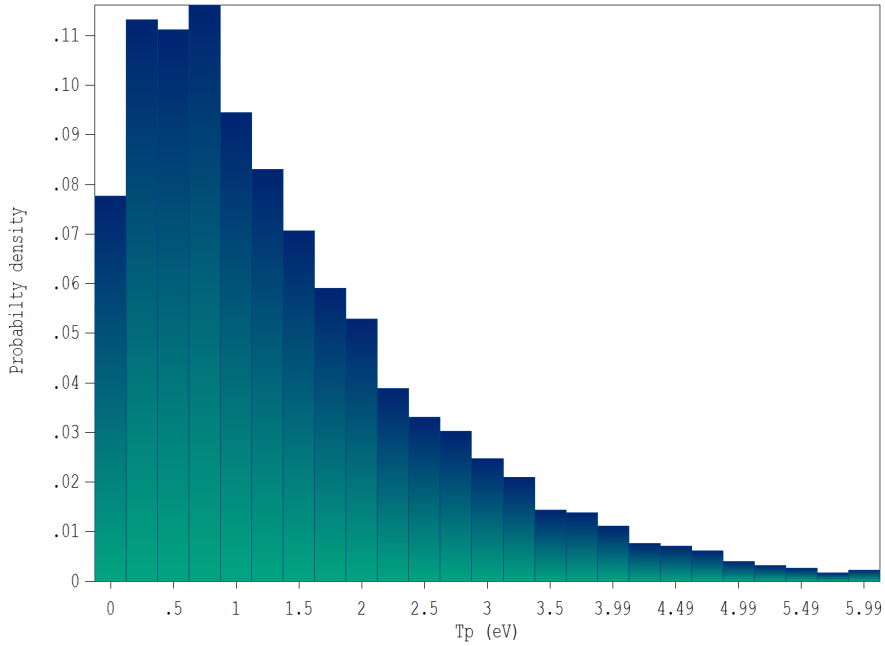

Figure 2: Initial distribution of particles at the cathode. Kinetic energy for $kT = 1.0$ eV.

energy U. Figure 2 shows the result for 10,000 model electrons with $kT_e = 1.0$ eV. The distribution in kinetic energy follows Eq. 4. To test the validity of the procedure, we can calculate the average kinetic energy of the electron distribution. The value for the distribution of Fig. 2 is 1.485 eV. The small difference from the theoretical value of $3kT/2 = 1.500$ eV is a consequence of truncation of the distribution at high energy.

A tabular function is an array of double precision numbers that contains a set of x and $y(x)$ values (maximum of 256) plus information about the structure of the table. The calling program defines a table with statements of the form

```
INTEGER(4),PARAMETER :: NTabSize = 770
INTEGER(4),PARAMETER :: NTabMax = 32
DOUBLE PRECISION MuTab(1:NTabSize,1:NTabMax)
```

The statements are taken from the **Magnum** program, which can include up to 32 tables of relative magnetic permeability μ_r as a function of $\mu_0 H$. A set of values xq and yq is added to a table with a statement of the form

```
CALL PutXY(MuTab(1,NumTable),xq,yq)
```

The module handles stack operations and indices internally. A tabular function has the following structure within the module:

```
FName(1) - FName(256)     XValues
FName(259) - FName(512)   YValues
FName(513) - FName(768)   YDPValues
FName(769)  DBLE(NEnt)
   Number of x-y pairs, negative if table has not been
   sorted and if the cubic spline coefficients have not
   been calculated
FName(770)  DBLE(NError)
```

The YDP values are used for cubic spline interpolations. The module includes the following interface functions and subroutines:

```
SUBROUTINE PutXY(FName,XValue,YValue)
  Adds an x-y set to the tabular function FName. XValue and
  YValue are double precision numbers. Increments NEnt and
  sets it equal to a negative number.
  NError = 1 if the table is full

DOUBLE PRECISION FUNCTION GetX(FName,N)
  Returns the Nth x value of the tabular function.
  NError = 1 and GetX = 0.0 if N < 1 or N > NEnt

DOUBLE PRECISION FUNCTION GetY(FName,N)
  Returns the Nth y value of the tabular function.
  NError = 1 and GetX = 0.0 if N < 1 or N > NEnt

SUBROUTINE Define(FName)
  Sets all x,y and y" values to 0.0, and NEnt = 0
  NError = 0

SUBROUTINE ScrDump(FName,NUnit,LinFlag)
  Diagnostic routine to write contents of FName to
  file unit NUnit. Cubic spline interpolation if
  LinFlag is FALSE, otherwise linear interpolation
```

```
INTEGER FUNCTION NElem(FName)
  Returns the number of x-y values stored in the table
  FName

DOUBLE PRECISION FUNCTION XTMin(FName)
  Returns the minimum value of x stored in FName.
  NError = 1 and XMin = 0.0 if NEnt = 0

DOUBLE PRECISION FUNCTION XMax(FName)
  Returns the maximum value of x stored in FName.
  NError = 1 and XMax = 0.0 if NEnt = 0

DOUBLE PRECISION FUNCTION YMin(FName)
  Returns the minimum value of y stored in FName.
  NError = 1 and YMin = 0.0 if NEnt = 0

DOUBLE PRECISION FUNCTION YMax(FName)
   Returns the maximum value of y stored in FName.
   NError = 1 and YMax = 0.0 if NEnt = 0

DOUBLE PRECISION FUNCTION Value(FName,x)
   Returns the value of y at the position x in the
   tabular function FName, cubic spline interpolation
   NError = 1 and Value = 0.0 if NEnt = 0
   NError = 2 and Value = 0.0 if x out of range
   NError = 3 and Value = 0.0 if interpolation error

DOUBLE PRECISION FUNCTION LValue(FName,x)
   Returns the value of y at the position x in the
   tabular function FName, linear interpolation
   NError = 1 and Value = 0.0 if NEnt = 0
   NError = 2 and Value = 0.0 if x out of range
   NError = 3 and Value = 0.0 if interpolation error

INTEGER FUNCTION NCode
  (Returns last NError)

SUBROUTINE Deriv(FName)
  Converts the tabular function to its derivative
  NError = 1 if NEnt = 0
  NError = 2 if interpolation error
```

```
SUBROUTINE Integral(FName)
    Converts the tabular function to its integral
  NError = 1 if NEnt = 0
  NError = 2 if interpolation error
  Checks that y is a monotonically increasing or decreasing
  value of x. If so, inverts the function. The function is
  unchanged if there is an error
  NError = 1 if NEnt = 0
  NError = 2 if non-monotonic function

SUBROUTINE Invert(FName)
  Checks that y is a monotonically increasing of decreasing
  value of x. If so, inverts the function. The function is
  unchanged if there is an error
  NError = 1 if NEnt = 0
  NError = 2 if non-monotonic function

SUBROUTINE Copy(FName,FName2)
  Copies tabular function FName to FName2
```

To request the module source code, contact us at techinfo@fieldp.com.