HOT Thermal Database version 2.4

A MATLAB/OCTAVE Toolbox

Chris Martin
Virginia Active Combustion Control Group
Virginia Tech

Table of Contents

i. Forward

In the spirit of collegial cooperation, we at the Virginia Active Combustion Control Group of the Department of Mechanical Engineering at Virginia Tech are pleased to share the first public release of the HOT thermal database package for Matlab and Octave. We do so hoping that engineers, scientists, and students alike will find the software as great an aid as we for the routine accurate calculation of thermodynamic data across a variety of species.

I started working on code that later evolved into HOT in 2004 when I began my graduate work at Virginia Tech. Since then it has involved a hand full of other programmers and been used by a small group of researchers with whom I have had close contact. I am personally eager to hear comments and suggestions from a broader group of users. Because our focus is on combustion, the selection of included species has been focused on those present in combustion systems; thus we are eager to include data submitted or suggested by other users.

Sincerely,

Chris Martin
Virginia Active Combustion Control Group
Virginia Tech

ii. License

1.0 Overview

1.1 Applications

The HOT thermal database package is designed to store, recall, and perform computations on thermodynamic data in Matlab or Octave for a variety of species at a variety of temperatures and pressures. This is a system developed primarily for combustion modeling, but has already been used for a variety of applications in modeling thermodynamic cycles. Ultimately, every step of the software design placed transparency and flexibility at a premium. This is true to the point that the addition of new data (such as viscosity) will not disturb the operation of the old code. In fact, the existing tools can be instructed to look for new data fields in the hope that users will add their own data as it becomes necessary. At present, the package includes entropy, enthalpy, specific heat, molecular weight, and enthalpy of formation. Enthalpy and specific heat are referenced as functions of temperature only while entropy is referenced as a function of both pressure and temperature.

We can offer **ABSOLUTELY NO WARANTY WHATSOEVER** as to the reliability or accuracy of the code or data. Don't go using this code to control nuclear power plants or to design attack subs without checking your numbers. In the future, we intend to exhaustively validate the code against the original JANAF tables.

1.2 Thermal background

Here's just a little primer on the sorts of calculations the code performs.

1.2.1 Mass vs. Molar

There is an everlasting dialogue between the engineers and the chemists (the chemical engineers are just caught in the middle) as to whether extensive properties should be given in terms of mass or moles. Engineers, ever working in terms of mass flows frequently find thinking in terms of molecule counts counter intuitive. The chemists, thoughtful of the actual physical processes at work which are inherently linked to the vibration of individual molecules, want their units in moles. In the end, there is a potato/potato (a cliché that doesn't work well in text) debate.

Being engineers ourselves, we have come down on the side of mass rather than molar units. Should you or someone you love be sympathetic to the chemists on this one, a simple division by molecular weight is all you need to bridge the gap.

1.2.2 Properties

The HOT package currently supports the following properties with the following default units:

Table 1: HOT Supported Properties

Symbol	Description	Units
h	enthalpy	J/kg
hf	enthalpy of formation	J/kg
cp	constant-pressure specific heat	J/kg/K
s	entropy	J/kg/K
MW	molecular weight	kg/kmol, g/mol,etc…

At present, the HOT thermal database package assumes ideal, but not perfect gasses. For readers without a thermodynamic textbook in easy reach, the ideal gas assumption implies that the gas constant really is constant (Pv = RT) and a perfect gas has a constant specific heat (both cv and cp).

1.2.3 Enthalpy and cp

Enthalpy is the total energy a fluid contains including both internal energy, u, and the fluid-mechanical potential energy, Pv. Thus enthalpy is defined as

(1) $$h(T,P) = u(T) + P\nu$$

When the fluid in question is an ideal gas, the specific volume, v, can be written in terms of P and T and

(2) $$P\nu = RT$$

Thus, $h(T,P) = u(T) + RT$ and enthalpy is only a function of temperature.

Given that $h = h(T)$, it can be shown that the amount of energy required to raise the gas temperature a single degree when held at constant pressure is…

(3) $$c_p = \frac{dh(T)}{dT}$$

From the fundamental theorem of calculus,

(4) $$h(T) - h(T_{ref}) = \int_{T_{ref}}^{T} c_p(T)\, dT$$

The constant-pressure specific heat can be related to the constant volume specific heat by differentiating (1) with the ideal gas substitution. In the end,

(5) $$c_p(T) = c_v(T) + R$$

Since cv can be quite easy to measure in a calorimeter, frequently that is the information tabulated, from which we use (4) to compute enthalpy. Not that Tref and h(Tref) need to be supplied. While the selection of Tref is arbitrary, h(Tref) is the enthalpy of formation and is determined through a separate experiment.

1.2.4 Entropy

Gibbs relation states that $Tds = du - Pdv$ or

(6) $$Tds = dh - vdP$$

Therefore, substituting the ideal gas law for v again...

(7) $$ds = c_p \frac{dT}{T} - R\frac{dP}{P}$$

Thus, the entropy can be computed similarly to enthalpy

(8) $$s(T,P) - s(T_{ref}, P_{ref}) = \int_{T_{ref}}^{T} c_p(T)\frac{dT}{T} - R\ln\left(\frac{P}{P_{ref}}\right)$$

It is quite common to split the entropy into two parts…

(9) $$s(T,P) = s_0(T) - R\ln\left(\frac{P}{P_{ref}}\right)$$

And …

(10) $$s_0(T) - s(T_{ref}, P_{ref}) = \int_{T_{ref}}^{T} c_p(T)\frac{dT}{T}$$

Where the later (s0) is tabulated and the pressure dependency is calculated explicitly.

The HOT system is currently designed to deal with both explicitly tabulated values and specific heat curve fits. Future versions may support integration of tabulated specific heats via a spline.

1.2.5 Mixtures

The HOT Toolbox can deal with mixtures of species. Given the above methods for computing the properties of the individual species, it remains to be seen how one computes the properties of a mixture of species. Since most of the properties supported by HOT are extensive in nature (per unit mass), finding the total mixture properties is usually as simple as a mass-weighted average.

In a mixture, the mass fraction, Yi, of specie, "i", is defined as the mass of specie "i" divided by the total mass in the mixture. The mass fraction also has relevance in a continuum, wherein the mass fraction can change in space. In this case, the mass fraction can be defined as the local mass density of specie i divided by the local density of the fluid. There are several important things to note about mass fractions.

For most extensive properties, the following is true (where "z" is a make-believe property)

$$z = \sum z_i Y_i$$

This is just the mass weighted average of the individual specie properties. It is true for the following properties currently supported in HOT:

(11) $$h = \sum h_i Y_i$$

(12) $$s = \sum s_i Y_i$$

(13) $$c_p = \sum c_{p,i} Y_i$$

(14) $$h_f = \sum h_{f,i} Y_i$$

It is NOT true, however, for molecular weight.

Molecular weight is defined as the mass per molecule of a specie. Typically MW is reported in g/mol, kg/kmol, lb/lbmol, or AMU. For a group of molecules of specie "i", the molecular weight can be defined as

(15) $$MW_i = M_i / N_i$$

Where M is the mass of the fluid and N is the number of molecules present. If the fluid is a mixture of species, then …

(16) $$MW = M / N$$

Where N = sum Ni. Substituting from (15), and with a little manipulation

(17) $$MW = 1 / \sum \left(Y_i / MW_i \right)$$

1.3 Promise of transparency

HOT is released as open-source software because we believe in the importance of shared information to research. It is our fervent hope that it be modified to meet the needs of various users. To that end, all data, all code, and all files ever having anything to do with this project are and will always be formatted either in plain text or in a format readable by free or open-source software.

2.0 Installation

2.1 Installing in Matlab

Copy the "HOT" and "utility" directories onto the desired hard drive. A typical place might be in "C:\programs\Matlab_RXX\work\", but you can place them anywhere you like. Open Matlab. Use the following commands to install the files:

Navigate to the utility directory.

```
>> cd C:\programs\Matlab_R14\work\utility
```

The install command located in the utility directory adds the current directory to the Matlab path.

```
>> install
```

With utility added to the Matlab path, the install utility is now accessible everywhere. Navigate to the HOT directory. Replace the path with your path to HOT.

```
>> cd C:\programs\Matlab_R14\work\HOT
```

Now run the install command again.

```
>> install
```

If you change your mind, you can always remove the directories using the "uinstall" command.

2.2 Installing in Octave

Copy the "HOT" and "utility" directories into the file system. A typical place might be in "/usr/share/octave/3.0.1/m/", but you can place them anywhere you like. Start Octave. Use the following copmmands to install the files:

Navigate to the utility directory. Replace the path with your path to the utility directory.

1> cd /usr/share/octave/3.0.1/m/utility

The install command located in the utility directory adds the current directory to the Octave path.

2> install

With utility added to the Octave path, the install utility is now accessible everywhere. Navigate to the HOT directory. Replace the path with your path to HOT.

3> cd /usr/share/octave/3.0.1/m/HOT

Now run the install command again.

4> install

If you change your mind, you can always remove the directories using the "uinstall" command.

3.0 Files and Directories

This is a listing of each file in the HOT and utility directories.

3.1 HOT

3.1.1 DENSITY.M

```
DENSITY.M

   rho = density(data, species, mass, T, P)
       or
   rho = density(data, state)

Computes a 2-D array of density for various mixtures
at various temperatures.
================================================================
data       -   janaaf data struct array
                    OR
               cell array containing multiple janaaf data struct arrays


species    -   cell array of species names or a single species name

mass       -   2-D array of mass fractions or absolute species masses
               each row corresponds to an element in species (t.f.
               needs the same number of rows as species has elements)
               In addition, each column in mass will correspond to a
               new set of data in the output.  In this way, one call
               to THERMAL can return multiple mixture results.
```

```
T          -   Temperature vector.  Must be a column vector of
               temperatures in K.


rho        -   a 2-D numeric array containing the computed property values.
               rho(n,m) = density at temperature (n) and mixture (m)
```

3.1.2 ENERGY.M

```
ENERGY.M

   u = energy(data, species, mass, T)
       or
   u = energy(data, state)

Computes a 2-D array of internal energy for various mixtures
at various temperatures.
================================================================
data       -   janaaf data struct array
                    OR
               cell array containing multiple janaaf data struct arrays


species    -   cell array of species names or a single species name

mass       -   2-D array of mass fractions or absolute species masses
               each row corresponds to an element in species (t.f.
               needs the same number of rows as species has elements)
               In addition, each column in mass will correspond to a
               new set of data in the output.  In this way, one call
               to THERMAL can return multiple mixture results.

T          -   Temperature vector.  T must be a vector of the same length
               as P or a scalar indicating a constant temperature.

state      -   state structure as generated by STATEGEN.

u          -   a 2-D numeric array containing the computed property values.
               u(n,m) = energy at temperature (n) and mixture (m)


See also:
   enthalpy, spheat
```

3.1.3 ENTHALPY.M

```
ENTHALPY.M

   h = enthalpy(data, species, mass, T)

Computes a 2-D array of enthalpy for various mixtures
at various temperatures.
================================================================
data       -   janaaf data struct array
                    OR
               cell array containing multiple janaaf data struct arrays
```

```
species    -   cell array of species names or a single species name

mass       -   2-D array of mass fractions or absolute species masses
               each row corresponds to an element in species (t.f.
               needs the same number of rows as species has elements)
               In addition, each column in mass will correspond to a
               new set of data in the output.  In this way, one call
               to THERMAL can return multiple mixture results.

T          -   Temperature vector.  Must be a column vector of
               temperatures in K.


h          -   a 2-D numeric array containing the computed property values.
               h(n,m) = enthalpy at temperature (n) and mixture (m)
```

3.1.4 ENTROPY.M

```
ENTROPY.M

   s = entropy(data, species, mass, T, P)
       or
   s = entropy(data, species, mass, T)
       or
   s = entropy(data, state)

Computes a 2-D array of entropy for various mixtures
at various temperatures.
=============================================================
data       -   janaaf data struct array
                    OR
               cell array containing multiple janaaf data struct arrays


species    -   cell array of species names or a single species name

mass       -   2-D array of mass fractions or absolute species masses
               each row corresponds to an element in species (t.f.
               needs the same number of rows as species has elements)
               In addition, each column in mass will correspond to a
               new set of data in the output.  In this way, one call
               to THERMAL can return multiple mixture results.

T          -   Temperature vector.  T must be a vector of the same length
               as P or a scalar indicating a constant temperature.

P          -   Pressure vector.  P must be a vector of the same length as
               T or a scalar indicating a constant pressure.  If P is
               omitted, then it will be set to 101300 Pa or standard
               pressure.

state      -   The state structure such as one generated by STATEGEN.

s          -   a 2-D numeric array containing the computed property values.
               s(n,m) = entropy at temperature (n) and mixture (m)

See also:
```

```
   enthalpy, spheat
```

3.1.5 IGCONSTANT.M

```
IGCONSTANT.M   compute a mixture's constant-pressure specific heat

   R = IGCONSTANT(data, species, mass)

Computes a 1-D array of the ideal gas constant for various mixtures
at various temperatures.  If the properties being computed do not
require temperature as an input, it may be omitted from the
function call.

================================================================
data       -   janaaf data struct array
                    OR
               cell array containing multiple janaaf data struct arrays


species    -   cell array of species names or a single species name

mass       -   2-D array of mass fractions or absolute species masses
               each row corresponds to an element in species (t.f.
               needs the same number of rows as species has elements)
               In addition, each column in mass will correspond to a
               new set of data in the output.  In this way, one call
               to THERMAL can return multiple mixture results.

R          -   a 1-D numeric array containing the computed property values.
               R(m) = ideal gas constant for mixture (m)

See also:
   enthalpy, entropy
```

3.1.6 JANCHECK.M

```
JANCHECK.M  test HOT data for compatibility with high-level functions

   flags = jancheck(data, param1, value1, ...);

data       janaf data structure array

flags      string of characters describing the data format
1) check success/failure
           'p' (p)assed. The data is properly formated.
           'f' (f)ailed. The data has at least one error.
               Run verbosely to find out what's wrong.

2) format/type
           'f' (f)it. The data specifies curve fits.
           't' (t)abular. The data is in a lookup table.
           'n' (n)one determined.

3) enthalpy computed from
           'e' (e)xplicitly. Enthalpy is given explicitly.
           'i' (i)mplicitly. Enthalpy must be computed from cp.
           'n' enthalpy is (n)ot specified.
```

```
 4) entropy computed from
            same as (3)

 5) specific heat computed from
            'e' (e)xplicitly.
            'i' (i)mplicitly. Specific heat must be computed from h.
            'n' specific heat is (n)ot specified

 JANCHECK accepts the following parameter-value pairs:
  verbose    accepts either 1 or 0 to indicate a verbose or quiet output
```

3.1.7 JANFIND.M

```
 JANFIND  find entries in a janaf library

    index = janfind(data, 'namefield', names)

 given the cell array strings in name, JANFIND
 returns indices such that
    data(index(k)).namefield = names{k}

 JANFIND also accepts a single string for the
 names variable

 JANFIND will return an error if there are
 multiple instances of one of the name strings
```

3.1.8 JANFIT.M

```
 JANFIT  evaluate curve fit data in a janaf data struct
    JANAF data lookup toolbox
    written by Chris Martin 5/2008
    modified 4/2009

    output = janfit(C, F, R, T, param1, value1, ...)

  Evaluates curve fit data in the JANAF struct, data.  JANFIT makes no
 assumptions about the fields of data.
 ========================================================
    C           Coefficient matrix.  Each row corresponds to a fit function
                Each column corresponds to a temperature range.

    F           Fit function vector.  A numeric vector with the same number
                of elements as C has rows.  NaN or inf entries in F are
                treated as natural logs.  All other entries are treated as
                exponent entries to a polynomial.  For example, if
                    F = [1 2 inf].';  C = [0.5 3 2.4].';
                then the fit function would be
                    y = 0.5*x + 3*x^2 + 2.4*log(x)

    R           Range vector.  A monotonically increasing numeric vector
                with the same number of elements as C has columns.  Each
                element of R is the greatest value of the independent
variable
                for which the fit coefficients in the corresponding column
of
                C are valid.  For example, if
                    R = [ 500 1000 2000 ];  T = [300 600 900 1200].';
```

```
                JANFIT would use the first column of C to evaluate T=300,
                the second column to evaluate T=[600 900], and the thrid
                column of C to evaluate T=1200.

 ========================================================
 optional param, value pairs:

    'deriv'     non-negative integer - indicates the order of derivative to
                compute.  For example, if C and F represent fit functions
for
                the enthalpy of a substance,
                    janfit(C,F,400,'deriv',1)
                will return the specific heat at 400 K
                If C and F represent fit functions for the specific heat of
a
                substance,
                    janfit(C,F,400,'deriv',-1)
                will return the enthalpy at 400 K.  The integration
coefficient
                is automatically taken such that all integrals are zero when
the
                independent variable is 298.15.  This reference state can be
                changed using the 'reference' parameter.
                (DEFAULT = 0)

    'reference' numeric or character - indicates the reference state for
                integrations; the value of the independent variable at which
                integrations are taken to be zero.  This typically accepts a
                numeric value, but also recognizes 'R', 'F', 'C', and 'K';
                corresponding to the standard reference temperatures in
                Rankine (527.67), Farenheit (68), Celcius (20), and Kelvin
                (298.15)
                (DEFAULT = 'K')

    'verbose'   logical - use verbose output? (prints extrapolation
warnings)
                accepts 0,1,'y','n'.  All other inputs will be evaluated to
true
                (DEFAULT = 0)

    'onfail'    what to do if field lookup fails.
                EX  'onfail', 'fatal':  returns an error and quits
                    'onfail', NaN:      returns NaN and continues running
                    'onfail', 12:       returns 12 and continues running
                (DEFAULT = 'fatal')
```

3.1.9 JANLOAD.M

```
 JANLOAD.M

    data = janload('filename', param1, value1, ...)
        or
    [data type] = janload('filename', param1, value1, ...)

    Janload uses the dread utility function to load an ascii
 file into a struct array.

 JANLOAD accepts the following optional parameter-value
```

```
 pairs:

  verbose       Enable verbose operation

  sort          Accepts the string name of any field in the
                data being loaded.  Janload automatically
                sorts the data by that field (alphabetically
                if the field contains string data).

  check         When true, janload runs jancheck on the data
                if a second variable is available in the
                output, the jancheck type flags will be
                written to them.
```

3.1.10 JANLOOKUP.M

```
 JANLOOKUP.M
    JANAF data lookup toolbox
    written by Chris Martin 5/2008

    output = janlookup(data, outfield, infield, value, param1, value1, ...)

  Interpolates table data in the JANAF data struct.
 ========================================================
 data           JANAF data struct.  Fields are flexible.

 outfield       String name of the field containing the
                output data vector (data.outfield)

 infield        String name of the field containing the
                input data vector (data.infield)

 value          numeric value of the input variable

 output         numeric interpolated value of the output field
 ========================================================
 optional param, value pairs:
    'deriv'     estimate the derivative of the interpolation spline
                to the order specified. (may not exceed the interp order)
                EX  'deriv', 0: results in direct interpolation
                    'deriv', 2: returns the second derivative
                (DEFAULT = 0)

    'interp'    Interpolation order (spline order).  Must be
                0 or odd.
                EX  'interp', 0: results in low-value lookup
                    'interp', 1: uses linear interpolation
                    'interp', 3: uses cupic interpolation
                (DEFAULT = 3)

    'verbose'   logical - use verbose output? (prints extrapolation
warnings)
                accepts 0,1,'y','n'.  All other inputs will be evaluated to
true
                (DEFAULT = 0)

    'onfail'    what to do if field lookup fails.
                EX  'onfail', 'fatal':  returns an error and quits
```

```
                    'onfail', NaN:      returns NaN and continues running
                    'onfail', 12:       returns 12 and continues running
```

3.1.11 LISTSPEC.M

```
LISTSPEC.M  list the species and data type in the library.

   listspec(data, param1, value1, ...)
       or
   [species type setnum index] = listspec(data, param1, value1, ...)

Generates a table summarizing the species present in the library.

data       data library

species    optional argument - cell array of species present
type       optional argument - cell array containing the string
           "fit" or "table", identifying the data type of each
           specie.
setnum     optional argument - a numeric array identifying which
           element of the library contains the corresponding data.
index      optional argument - a numberic array indicating the
           specie location in the data set.

LISTSPEC accepts the following parameter-value pairs:

verbose    boolean indicating verbose operation
sortby     String indicating which of the columns by which to sort.
           Must be one of "specie", "format", "set", or "none".
```

3.1.12 MWEIGHT.M

```
MWEIGHT.M

   MW = mweight(data, species, mass)
       or
   MW = mweight(data, state)

Computes a 1-D array of molecular weights for various mixtures.
================================================================
data       -   janaaf data struct array
                    OR
               cell array containing multiple janaaf data struct arrays


species    -   cell array of species names or a single species name

mass       -   2-D array of mass fractions or absolute species masses
               each row corresponds to an element in species (t.f.
               needs the same number of rows as species has elements)
               In addition, each column in mass will correspond to a
               new set of data in the output.  In this way, one call
               to THERMAL can return multiple mixture results.

state      -   the state structure as generated by STATEGEN.

MW         -   a 1-D numeric array containing the computed property values.
```

```
                MW(1,m) = molecular weight for mixture (m)
```

3.1.13 PROCESS.M

```
PROCESS.M  compute the end-state of a thermodynamic process

   state = process(data, param1, value1, ...)

Computes the final thermodynamic state for a given mixture undergoing
a process.  Constrain the state by specifying various properties at
the end state.  PROCESS will invert the relations in the data struct
to compute the corresponding temperature and pressure.

The arguments used by PROCESS are
data       The structure array generated by JANLOAD.

state      A structure generated by state.m, indicating the initial
           state.

PROCESS accepts the following parameter-value pairs:

***** Constrain the end-state *****
species    The species present in the end-state mixture.
           (required)

mass       The end-state mixture
           (required)

T          The end-state temperature

P          The end-state pressure

s          The end-state entropy

h          The end-state enthalpy

rho        The end-state density

u          The end-state internal energy
```

3.1.14 SPHEAT.M

```
SPHEAT.M   compute a mixture's constant-pressure specific heat

   cp = spheat(data, species, mass, T)
       or
   cp = spheat(data, state)

Computes a 2-D array of enthalpy for various mixtures
at various temperatures.  If the properties being computed do not
require temperature as an input, it may be omitted from the
function call.

================================================================
data       -   janaaf data struct array
                    OR
               cell array containing multiple janaaf data struct arrays
```

```
species    -   cell array of species names or a single species name

mass       -   2-D array of mass fractions or absolute species masses
               each row corresponds to an element in species (t.f.
               needs the same number of rows as species has elements)
               In addition, each column in mass will correspond to a
               new set of data in the output.  In this way, one call
               to THERMAL can return multiple mixture results.

T          -   Temperature vector.  Must be a column vector of
               temperatures in K.

state      -   the state structure as generated by STATEGEN.


cp         -   a 2-D numeric array containing the computed property values.
               cp(n,m) = specific heat at temperature (n) and mixture (m)

See also:
   enthalpy, entropy
```

3.1.15 SPRATIO.M

```
SPRATIO.M   compute a mixture's ratio of specific heats

   r = spratio(data, species, mass, T)
       or
   r = spratio(data, state)

Computes a 2-D array of the ideal gas constant for various mixtures
at various temperatures.  If the properties being computed do not
require temperature as an input, it may be omitted from the
function call.

=================================================================
data       -   janaaf data struct array
                    OR
               cell array containing multiple janaaf data struct arrays


species    -   cell array of species names or a single species name

mass       -   2-D array of mass fractions or absolute species masses
               each row corresponds to an element in species (t.f.
               needs the same number of rows as species has elements)
               In addition, each column in mass will correspond to a
               new set of data in the output.  In this way, one call
               to THERMAL can return multiple mixture results.

T          -   Temperature vector.  Must be a column vector of
               temperatures in K.

state      -   state struct as defined by STATEGEN

r          -   a 2-D numeric array containing the computed property values.
               r(m,n) = specific heat at temperature, m, for mixture, n.
```

3.1.16 STATEGEN.M

```
STATEGEN.M  build a thermodynamic state structure

   mystate = stategen(species, mass)
       or
   mystate = stategen(species, mass, T)
       or
   mystate = stategen(species, mass, T, P)
       or
   mystate = stategen(species, mass, T, P, v)
       or
   [ T, P, v, mass, species] = stategen(mystate)
       or
   stategen(mystate)

Constructs and deconstructs state structs used by PROCESS.

species    cell array of species strings used by high-level
           function like ENTHALPY and MWEIGHT.

mass       A column vector of the same length as species
           specifying the mass of each specie present.

T          The scalar temperature

P          The scalar pressure

v          mixture velocity

mystate    A state struct with fields:
           species, mass, T, P, v

The order of the outputs is purposefully changed so that the
least interesting outputs are last.  That way, one only need
collect the outputs of interest; e.g.

   [T,P] = stategen(mystate);

Also, if the temperature, pressure, velocity, or mixture
appear as vectors (or a maxtrix in the case of mixture)
then STATEGEN will check the size of all other properties
to ensure proper formatting.

In the last case, STATEGEN serves only as a means of checking
for a badly formatted state struct.  If there is a problem,
STATEGEN will exit with an error.
```

3.2 UTILITY

The utility folder contains files used by HOT and by other VACCG packages.

3.2.1 CLAMP.M

CLAMP.M forces the values of a vector between a max and minimum value.

3.2.2 COLLAPSE.M

COLLAPSE.M reduces an array of structs to a struct of arrays.

3.2.4 DREAD.M

DREAD.M parses ascii files for struct array data.

3.2.5 DWRITE.M

DWRITE.M writes data from struct arrays to ascii files.

3.2.6 INSTALL.M

INSTALL.M adds the current directory, and when configured to do so, all subdirectories to the search path.

3.2.7 ISEVEN.M

ISEVEN.M tests a number for if it is an integer divisible by 2.

3.2.8 ISODD.M

ISODD.M tests a number for if it is an odd integer.

3.2.9 MPARSEF.M

MPARSEF.M parses files for configurable data patterns.

3.2.10 MPARSES.M

MPARSES.M parses strings for configurable data patterns.

3.2.11 ORDERFIELDS.M

ORDERFIELDS.M sorts the fields of a struct in alphabetical order.

3.2.12 SHUFFLE.M

SHUFFLE.M collapses multiple vectors into a single vector.

3.2.13 UINSTALL.M

UINSTALL.M removes the current directory, and when configured to do so, all subdirectories from the search path.

3.2.14 UNIXTEXT.M

UNIXTEXT.M prunes all '\r' characters from files to ensure uniformity across platforms.

3.2.15 VARARGPARAM.M

```
VARARGPARAM.M parses parameter-value pairs and is configurable to
automatically deal with error handling.
```

4.0 Using HOT

4.1 Loading Data

Using the HOT thermal database begins with being able to access the data. For ease of access, cross-platform universality, and to encourage users to modify the code to meet their needs, all of the thermal database files are stored in ascii. They are in easy-to-read formats that make it practical to manually open the text files as one might a text book. Though editing the files is generally discouraged, the utilities that access them are programed to be as flexible as possible to allow for manual edits and comments.

To load data from a file, "filename", use the command

```
>> data = janload('filename');
```

The variable, "data", will be a 1-D struct array with fields depending on the content of the data. At this point, one can either rely on the integrity of the data or one can check that it was loaded properly.

```
>> flags = jancheck(data);
```

By default, jancheck runs verbosely; printing out a summary of the data it finds and the tests it conducts as it does them. It identifies the fields in the struct array, checks whether they are consistent with fit or tabular data, and tests each specie to see that its data is properly formatted for use with the thermal database. If it is run non-verbosely (see jancheck's help for more details) then the user has only the content of the "flags" variable to indicate the results of the check (again - see jancheck's help for details).

To do all of this in one step, run

```
>> data = janload('filename','verbose',1);
```

When janload is run verbosely, it automatically runs jancheck so the user can see a summary of the data and a reassurance of its integrity. Lastly, janload can also be configured to automatically sort the data.

```
>> data = janload('filename','sort','species');
```

This command will automatically sort the elements of data by the contents of the field, "species".

4.2 Data format

Except for users interested in understanding the nuts and bolts of the code a little better, the data format is not particularly important since users should never need to interact with it directly. If you

just want a straight-forward description for how to look up properties, it is safe to skip to subsection 4.4.

The format of the data is somewhat flexible. Fields can be added or subtracted from the data array on a whim so long as a core few remain unaltered. What those few are depends on the type of data being stored. For that reason, the functions that operate on the data can be grouped into two types; ones that look for specific fields and make assumptions about how the data is organized, and ones that need to be told explicitly how the data is organized and what to do with it. The idea is that there are certain low-level operations that are needed regardless of how the data might be re-arranged in the future, but it is hardly convenient to explicitly configure these functions every time one calls them. Therefore, there are also high-level functions that automatically configure the low-level ones based on the current configuration of the data. Usually, all of this is transparent to the user, but for the interested, the next subsection will spell out the details.

The data is a 1-D struct array. Each element of the array corresponds to a species such as H20, CH4, CO2, etc... Regardless of format (tabular or curve fit), most high-level functions look for the following fields:

Table 2: Struct Fields

field name	description
hf	enthalpy of formation
MW	molecular weight
species	string species identifier

4.2.1 Tabular Data

If the data is in a tabular format, then JANCHECK (and all the high-level functions) looks for the "T", "h", and "s" fields, containing lookup table data for the temperature, enthalpy, and entropy respectively. For each species in the array, these must be numerical vectors of the same length and T must be monotonically increasing. For a given temperature in "T", the corresponding element of “s” or “h” contains the entropy and enthalpy at that temperature respectively.

4.2.2 Curve Fit Data

If the data is in curve fit format, then JANCHECK looks for the "C", "F", and "T" fields, containing the specific heat coefficients, function specifications, and temperature range specification respectively. JANFIT uses the numerical function specifications to evaluate the fit function. An element of "F", "fk", indicates a term in the fit function, T.^fk. If "fk" is "NaN" or "inf", then the function is ln(T) instead.

Frequently, curve fit data is only valid over a limited range of temperatures and multiple fit curves can be spliced together for a more accurate fit. The "T" vector indicates the temperatures above which one fit becomes invalid and the next fit should be used. Therefore, the "T" vector must be monotonically increasing.

Given that the "F" vector length indicates the number of functions, and that the "T" vector length indicates the number of curve fits, the "C" coefficient matrix dimensions are completely constrained. Each row corresponds to an element of "F" and each column to an element of "T". For example, if a mock data element were configured by

```
>> data(1).F = [0;1;NaN];
>> data(1).T = [500 1000];
>> data(1).C = [500 550; 30 15; 0 0.1];
```

Then the specific heat at anywhere above 500K would be evaluated as …

[cp @ T>500] = 550 + 15 T + 0.1 ln(T)

And …

[cp @ T<500] = 500 + 30 T + 0.0 ln(T)

4.3 Low- And High-Level Operations

Low-level operations need lots of configuring in order to work properly, but are highly adaptable to changes in the data. On the other hand, high-level functions require very little information from the user and do most of the book keeping themselves, but they are extremely dependent on the structure and content of the data.

Most of the functions named JAN{something}.m (such as JANLOAD) are low-level functions, while most of the functions with more memorable names (such as enthalpy and entropy), are high-level functions.

JANLOAD, already mentioned in Section 4.1, is a perfect example of a low-level function. It knows that it must load a struct array, but it has absolutely no knowledge of what the fields will be, how many elements it will have, or what type of data will be in the struct. All of this, it learns from the file it is loading (implicitly through DREAD).

ENTHALPY, ENTROPY, MWEIGHT, and SPHEAT are excellent examples of high-level functions. Discussed in more detail in Section 4.4, they look for very specific data in the files they open. They do this by automatically configuring low-level functions (such as JANFIT or JANLOOKUP) to do most of the work.

JANCHECK, on the other hand, doesn't fit well into the naming convention. Rather, it bridges the gap between high- and low-level functions. JANCHECK is responsible for testing data that is usable by the low-level functions to see if it is compatible with the high-level functions. If there are problems, it will make recommendations on how to fix them.

4.4 Calculations

4.4.1 Basic

The high-level functions are designed to be as convenient for common calculations as possible by taking full use of MATLAB's vector-friendliness. Assuming that the "data" variable created in Section

4.1 is still in memory, very little else needs to be done to compute properties. For example, the commands

```
>> enthalpy(data, 'H2O', 1, 450)
>> entropy(data, 'H20', 1, 450)
>> mweight(data, 'H2O', 1)
```

will print the enthalpy, entropy, and molecular weight of water at 450K respectively.

In each of these statements, there is quite a bit going on that is not obvious. Each function checks the struct array to see if it contains fit or lookup data. It then calls JANFIND to locate 'H2O' the element in the data struct whose "species" field contains 'H2O'. Then it uses the appropriate low-level function (JANFIT or JANLOOKUP) to compute/interpolate the requested property.

4.4.2 Mixtures

That doesn't explain why the third argument in the above function calls was "1". All of the high-level functions can deal with mixtures of multiple species. For example,

```
>> enthalpy(data, {'N2' 'O2'},[52.64; 16], 450)
```

… computes the enthalpy of a nitrogen and oxygen mixture (52.64 kg N2 and 16 kg O2). This is the enthalpy of air at 450 K. When these sorts of mixtures are dealt with, the function automatically normalizes the masses to be mass fractions.

4.4.3 Libraries

It is quite possible that one might want to consider mixtures of species that were loaded from different files. If those data sets have different fields, it would be impossible to collapse them into a single struct array. For that reason, all of the high-level functions can deal with a library of struct arrays contained in a cell array. For example:

```
>> A = janload('nasa.fit');
>> B = janload('stanjan.tab');
>> enthalpy({A B}, {'H2O' 'As'}, [0.99; 0.01], 450);
```

ENTHALPY is smart enough to look around in the library formed by the cell array, "{A B}", to find 'H2O' and 'As', even though they were loaded from different files. It also deals with the fact that the properties must be computed differently from each of them.

4.4.4 Vectors

The whole advantage to MATLAB is that one can deal with gobs and gobs of data in relatively complicated structures quite easily. The high-level functions take full advantage of that fact. Consider the session:

```
>> data = janload('nasa.fit');
>> T = [300:100:2000];
>> mix = [1 0.5 0;0 0.5 1];
```

```
>> spec = {'CH4' 'O2};
>> h = enthalpy(data, spec, mix, T);
```

The variable, "h", will be an 18x3 matrix. Each row in "h" corresponds to one of the temperatures given. Each column corresponds to one of the three mixtures specified.

This is also useful if one simply wants the properties of a list of species.

```
>> spec = {'CH4', 'O2', 'N2', 'CO2', 'H2O'};
>> h = enthalpy(data, spec, eye(length(spec)), 400);
```

This session segment will be a row vector containing the enthalpy of the respective species at 400 K.

4.5 Data Files and Units

4.5.1 Files

In the present release, there are two files that contain thermodynamic data; "nasa.fit" and "stanjan.tab". The file extensions denote the format of the data, but apart from good practices, there is no reason to preserve the extension convention. JANLOAD relies on the names and format of the fields to determine the data type and not the file extension. They are named for their origins. "nasa.fit" is a derivative from the data used by GRI-MECH. Similarly, "stanjan.tab" is a derivative of one of the data files used by the old combustion favorite, STANJAN.

The original NASA data is available from the GRI-MECH homepage, here:

http://www.me.berkeley.edu/gri-mech/version30/files30/thermo30.dat

http://www.me.berkeley.edu/gri-mech/data/thermo_table.html

The University of Wisconsin has a site devoted to the STANJAN files here.

http://ecow.engr.wisc.edu/cgi-bin/get/me/774/foster/stanjan/

Colorado State University also hosts a web interface for Stanjan that's quite useful.

http://navier.engr.colostate.edu/tools/equil.html

4.5.2 Units

ALL DATA USED BY HOT IS ASSUMED TO BE IN SI UNITS. That implies the following:

Table 3: System of Units

Units	
Energy	J
Mass	kg
Time	s
Force	N
Length	m
Press.	Pa

The high-level functions have a limited capacity for converting to other unit systems, but there is very little error checking and the interface is still a little difficult to use properly. **IT IS STRONGLY RECOMMENDED THAT YOU HANDLE UNITS CONVERSION IN YOUR OWN CODE.** Future releases of HOT will address this.

5.0 Support

5.1 Reporting Bugs

If you find a bug, please contact me:

Chris Martin

chmarti1@vt.edu

To be sure that I can deal with the problem in as timely a manner as possible, please include the following:

1) place "hot-tdb" somewhere in the subject line,
2) a copy of your commands or the m-file you were running at the time to help me reconstruct the problem,
3) a brief description of the bug you encountered in case it doesn't do the same for me.

I'll get back to you as soon as I can.

5.2 Reverse compatibility

Since some of us are sluggish about updating systems and others are yet more sluggish when told the must pay to do so, a number of folks are still using old versions of Octave and Matlab. On this matter, I must be very clear. I have only tested this code on Matlab 7.1 (R14) and Octave 3.0.1, but I have designed the code with as much attention to out-dated versions as possible. In fact, the sluggishness in loading the text files is entirely do to the fact that I wrote my own parsers to avoid compatibility problems. That's something you can expect to see improve in later versions.

6.0 Contributing to HOT

We welcome suggestions. We are eager to find ways in which the project can be adapted to meet people's needs.

We also welcome the opportunity to work with international contributors to put together translated version. We are currently looking for French and German translations for the help headers and especially this README.

6.1 Contact

Via mail: Attn: NAMES LISTED BELOW 100 Randolph Hall Virginia Active Combustion Control Group Department of Mechanical Engineering Virginia Tech Blacksburg, VA 24061	**Fax:** Attn: NAMES LISTED BELOW (540) 231-9100

Table 4: Contacts

Position	Name	Email
Author, PhD Candidate	Chris Martin	chmarti1@vt.edu
Lab Manager	Steve Lepera	leperas@vt.edu
Lead Professor	Uri Vandsburger, Ph.D.	uri@vt.edu

Thanks also to:

Nikita Sharakov

Robert Wiedman