NIST RefProp Add-in
Reference Fluid Thermodynamic and Transport Properties

RefProp Mathcad Add-in: Version 2.1.0

Mathcad wrapper functions for
NIST Reference Fluid Thermodynamic and Transport Properties Database

NIST RefProp DLL Currently Loaded

NIST RefProp Library: Version 10.0.0.2

Updated: 01/28/2023

Contents

About the NIST RefProp Mathcad Add-in

Introduction

This User's Guide accompanies the **NIST RefProp Mathcad Add-in.** The add-in contains over 50 functions that return thermodynamic and transport properties computed from the **NIST Standard Reference Database**. To make the NIST RefProp DLL available to this add-in, NIST RefProp 9.1.1 or later must be installed on the machine. This User's Guide contains live math documents and text, explaining how all of these functions work, and using them to calculate real-world examples. It also has an index of all functions contained in the Add-in.

Once the Add-in is installed, these functions can be used in any Mathcad document, either though the Insert Function dialog or using the provided reference worksheet with units. They produce output that can be used in the same way as output from the operators and functions that are part of Mathcad.

Organization

This Guide is organized into seven chapters and five appendices that fully explain the add-in interface to the **RefProp** Functions and give examples of their usage. A Reference Section and Function Index are also included.

Units

All of the **NIST REFPROP** native DLL functions adhere to the SI system of units, excepting pressures in [kPa] and molar based energy units of [J/mole]. The **RefProp Mathcad Add-in** functions are ***mass based only*** and follow the same convention, except that pressure values are passed and retuned in units of [MPa] and energy units of [kJ/kg]; mostly to keep numerical input/output values manageable and of an order of magnitude that minimizes roundoff error when working with these numbers in Mathcad. Conversion between pressure and energy units is handled by the wrapper code. This set of units is consistent with unit standards of the IAPWS and ASME functions and databases. Although the parameters passed to the direct functions should be scaled to the proper units, they should not have actual Mathcad units attached.

Throughout this Guide, examples are given for stripping and re-applying units for each of the **RefProp Add-in** functions.

Double Precision

The **NIST RefProp** functions are implemented in double precision Fortran and the **RefProp Mathcad Add-in** wrapper functions are implemented in double precision C++.

Chapter 1 Introduction

About RefProp

REFPROP is an acronym for **REF**erence fluid **PROP**erties. This program, developed by the National Institute of Standards and Technology (NIST) provides formulations of the thermodynamic and transport properties of industrially important fluids and their mixtures with an emphasis on refrigerants and hydrocarbons. These properties can be displayed in tables and plots through the graphical user interface; they are also accessible through spreadsheets or user-written applications accessing the REFPROP DLL or the FORTAN property subroutines.

About the Mathcad Add-In

The REFPROP formulations are coded in standard Fortran-77 and compiled into a Dynamically Linked Library (DLL). This library of routines is loaded by a Mathcad Custom Function library, created in Microsoft Visual C++, and compiled into it's own DLL. Each custom function calls its corresponding NIST RefProp function in REFPROP.DLL (or 64-bit REFPRP64.DLL) and passes parameters and results to and from the Mathcad interface.

The Mathcad functions have been "verified" to give the same answers as can be obtained by other implemented wrappers of the REFPROP library to a precision 6 significant figures or better. "Validation" of the REFPROP library for accuracy is performed by NIST, but should be obtained independently by users of this code for the specific materials of interest.

Version 1.0 of this Mathcad Add-In provided functionality for ***pure fluids only***. Support for ***mixtures*** of pure components was not supported due to inaccuracies observed in the mixture models. Version 1.0 was based on the library files from, version 7.1 of the NIST REFPROP library, which were linked into the Mathcad wrapper DLL directly as a compiled LIB file.

Version 1.1 added some additional reverse function calls, and incorporated the library files from version 8.0 of the NIST REFPROP database.

Version 2.0 is a rewrite of the wrapper function stubs to load the NIST REFPROP DLL from the user's machine. The NIST RefProp software must be installed (purchased from NIST), in addition to the RefProp Mathcad Add-in DLL. New features in version 2.0 include

- Loading of predefined mixture files in addition to pure and pseudo-pure fluids
- New fluids and mixtures available in RefProp 9.1 and later,
- More flexible and automatic location of the fluid files on the user's machine,
- Specification of custom mixtures in the fluid string,
- Dynamic use of the currently installed RefProp version DLL,
- Static linking of the Intel Fortran runtime libraries with RefProp 9.1.1, removing the Intel Composer (Fortran) Run-Time Library as an additional installation requirement.

Version 2.1 includes Custom Functions for the High-Level API that is included in REFPROP 10. REFPROP 10 or higher must be installed on the machine to use these functions. Access to the legacy API functions available in earlier versions of REFPROP is still maintained.

Chapter 2 RefProp Add-in Installation

Prerequisites

The latest NIST RefProp program ***must*** be installed on the machine or the Mathcad Add-in will not run (user will be notified if it is not and the Add-in will not load). This program is available for purchase from NIST. It is recommended that NIST RefProp 10.0.0 or later is installed as this version greatly improves installation and integration issues with other codes and provides a superior API.

NOTE: Earlier versions of NIST RefProp may work, however, they will most likely require the installation of the Intel Composer (Fortran) Redistributable Runtime Components as previous REFPROP.DLL and REFPROP64.DLL libraries were dynamically linked with these libraries. Version 9.1.1 and later DLLs link to these Fortran components statically, so they are included within the DLL and not required on the machine.

Installing the NIST RefProp Add-in

Compiling and Installing on Mathcad Prime

The procedure for compiling and installing the NIST RefProp add-in for Mathcad Prime is identical to the instructions below for Legacy Mathcad, with a few minor differences:

1. The Visual Studio 2015 solution file is located in the **\buildPrime** directory of the code repository. This solution should load in any newer version of Visual Studio and will be automatically converted to the new version when loaded.

2. Ensure that the project configuration is set to Prime8 | x64, as this is where file path configuration is stored and builds a 64-bit DLL for Mathcad Prime. A Prime7 configuration is also available if compiling for Mathcad Prime 7.0.0.0.

3. The compiled **PrimeREFPROPwrapper.DLL** is copied into the Mathcad Prime **\Custom Functions** directory. This is typically located here:

   ```
   C:\Program Files\PTC\Mathcad Prime 8.0.0.0\Custom Functions
   ```

   When compiling the wrapper DLL using the provided Visual Studio solution, this copy procedure is performed automatically.

The above path should be modified in the project properties if using newer than Mathcad Prime version 8.0.0.0 to reflect the correct location of the user's Mathcad installation in the following locations:

- C/C++ | General > Additional Include Directories
- Linker | Input > Additional Dependencies
- Build Events | Post-Build Event > Command Line & Description

NOTE: Mathcad Prime must be shut down when compiling the RefProp add-in or the files will not be copied to the Mathcad Prime installation directory.

Installation on Legacy Mathcad 15

The NIST RefProp add-in for legacy Mathcad is very simply installed by copying the compiled MathcadREFPROPwrapper.DLL into the Mathcad 15 **userefi** directory. This is typically located here:

```
C:\Program Files (x86)\Mathcad\Mathcad 15\userefi
```

When compiling the wrapper DLL using the provided Visual Studio 2015 solution, the resulting target DLL is copied automatically as a Post-Build Event, provided that the DLL compiled correctly and the users has write access to the **userefi** directory.

Compiling the Legacy Mathcad 15 DLL

Go to the /build15 directory of the wrapper repository and open the Visual Studio 2015 solution in Visual studio. Make sure that the project configuration is set to "**Release**" (not Debug) for "**x86**". This compiles the 32-bit version of the DLL add-in for Mathcad 15. Then select **Build | Build MathcadREFPROPwrapper** from the main menu. The DLL and support files should be copied to the Mathcad 15 installation directory automatically (user must have write access to Mathcad's **userefi** folder).

NOTE: Mathcad 15 must be shut down when compiling the RefProp add-in or the files will not be copied to the Mathcad 15 installation directory.

NOTE: Legacy Mathcad 15 is deprecated, no longer distributed by PTC, and not tested with the RefProp Add-in beyond version 2.0. However, users with permanent licenses should still be able to compile and use a Legacy Mathcad DLL.

Help Files

The ***NIST RefProp*** Handbook, context sensitive help, and Insert Function assistant will all be copied to the appropriate Mathcad 15 installation directories by the Pre-Build Event in the Visual Studio solution. These files can can also be copied to the appropriate directories manually if not building the DLL with Visual Studio.

Alternate NIST RefProp Location (Optional)

The RefProp DLLs as well as the fluids and mixtures files from NIST are stored and accessed from the RefProp installation directory on the user's hard drive. If NIST RefProp has been installed in a non-default location, the add-in does a pretty good job of finding it. However, you can manually point the Mathcad DLL to that location with a user environment variable, **NIST_PATH**.

The location provided should have two subdirectories underneath it,

{*NIST_PATH*}
\fluids
\mixtures

Fluid and mixture files should be loaded in these subdirectories appropriately and have a corresponding $HMX.BNC$ file in the \fluids directory.

Uninstalling the NIST RefProp Add-in

The NIST RefProp Add-in can be uninstalled very easily by removing the add-in DLL from the Mathcad Prime\Custom Functions directory (or the Legacy Mathcad 15 \userefi directory). The DLL file can be moved temporarily to a \save subdirectory to keep it from loading when Mathcad Prime (or Legacy Mathcad) is launched.

Chapter 3 Legacy API General Usage

The RefProp Legacy Add-in Functions

Once ***RefProp*** is installed, the ***RefProp*** functions will be registered by Mathcad Prime. Unfortunately, Mathcad Prime (as of version 8.0) does not yet have a facility to integrate these functions into the user interface. However, these functions follow a specific naming convention. This is *exactly* the naming convention used in other Mathcad add-ins such as the legacy **ASME Steam Tables add-in** from Adept Science (*no longer distributed or supported*) and the **IF97 add-in** from **CoolProp.org**.

- Each function starts with the prefix "***rp_***". This avoids conflicts with any other add-ins that may be loaded.

- The next characters in the function identify the physical property to be calculated. The available properties are:

  Thermodynamic properties
  rho (density),
  h (enthalpy),
  u (internal energy),
  s (entropy),
  cp (isobaric specific heat),
  cv (isochoric specific heat),
  w (speed of sound)

  Transport properties
  mu (viscosity)
  k (thermal conductivity)

  State Point
  t (temperature)
  p (pressure)
  tsat (saturation temperature)
  psat (saturation pressure)

- The property may be followed by either f (liquid) or g (gas) to get the ***saturation*** state, typically as a function of either the saturation temperature or saturation pressure (e.g. ***hf*** = saturated liquid/fluid enthalpy, ***sg*** = saturated vapor/gas entropy).

Fluid Material Selection

Fluid properties for each material in the ***RefProp*** library are stored in individual files in the `fluids\` and `mixtures\` sub-folders of the ***RefProp*** installation directory.

Each of the property functions requires a fluid string as its first parameter. This parameter communicates to the ***RefProp*** internal routines which fluid material file(s) should be used to calculate the material properties.

> *NOTE: Each fluid property function makes a call internally to the RefProp* ***SETUP*** *routine, which loads the fluid file from the database location. However, if the fluid requested is already loaded, no action is taken; eliminating unnecessary file I/O.*

Details:

- The "**fluid string**" for a **Pure Fluid** file to be loaded from ***RefProp*** can (optionally) end with an extension of "**.fld**". If the suffix is omitted, RefProp will assume an extension of "**.fld**" and look for the fluid in the "`fluids\`" sub-folder.

  e.g. to load the properties for HELIUM, define a variable such as:

  fluid := "HELIUM.fld" *or* *fluid* := "HELIUM"

- **Pseudo-Pure fluids** are mixtures that have been re-fit to load like a pure fluid. The "**fluid string**" for these files ***must*** end with the extension "**.ppf**".

  e.g. to load the pseudo-pure properties for AIR, define a variable such as:

  fluid := "AIR.ppf"

- Predefined Mixtures are located in the "`mixtures\`" sub-folder. The "**fluid string**" ***must*** use the suffix "**.mix**".

  e.g. to load the mixture properties for AIR, define a variable such as:

  fluid := "AIR.mix"

- If the suffix is omitted, RefProp will look preferentially in the "`fluids\`" folder for a **.fld** file, then a **.ppf** file, and then in the "`mixtures\`" folder for a **.mix** file.

- An **Ad-hoc Mixture** can also be specified in the "**fluid string**" as outlined in **Chapter 8 - Mixture Properties**.

- Available pure fluid files as of ***RefProp 9.1*** are shown below in **Table 1**, pseudo-pure fluids in **Table 2**, and mixture files are listed in **Table 3**. However, users should consult their latest version documentation.

Table 1 : RefProp Pure Fluid Files

Pure Fluids use extension ".fld" (optional)			
acetone	hexane	oxylene	r1233zd
ammonia	hydrogen	octane	r1234yf
argon	hcl	orthohyd	r1234ze
benzene	h2s	oxygen	r124
butane	isobutan	pxylene	r125
1butene	ibutene	parahyd	r13
co2	ihexane	pentane	r134a
co	ioctane	c4f10	r14
cos	ipentane	c5f12	r141b
c2butene	krypton	propane	r142b
cyclohex	mxylene	c3cc6	r143a
cyclopen	md2m	propylen	r152a
cyclopro	md3m	propyne	r161
d4	md4m	so2	r21
d5	mdm	sf6	r218
d6	methane	toluene	r22
decane	methanol	t2butene	r227ea
d2	mlinolea	cf3i	r23
dee	mlinolen	c11	r236ea
dmc	moleate	water	r236fa
dme	mpalmita	xenon	r245ca
c12	mstearat	novec649	r245fa
ethane	c1cc6	r11	r32
ethanol	mm	r113	r365mfc
ebenzene	neon	r114	r40
ethylene	neopentn	r115	r41
fluorine	nitrogen	r116	rc318
d2o	nf3	r12	re143a
helium	n2o	r1216	re245cb2
heptane	nonane	r123	re245fa2
			re347mcc

Table 2 : RefProp Pseudo-Pure Fluid Files

Pseudo-Pure Fluids use extension ".ppf"				
Air	R404A	R407A	R410A	R507A

Table 3 : RefProp Predefined Mixture Files

Pre-defined Mixtures use extension ".mix"				
R401A	R408A	R419A	R431A	R503
R401B	R409A	R420A	R432A	R504
R401C	R409B	R421A	R433A	R507A
R402A	R410A	R421B	R434A	R508A
R402B	R410B	R422A	R435A	R508B
R403A	R411A	R422B	R436A	R509A
R403B	R411B	R422C	R436B	R510A
R404A	R412A	R422D	R437A	R512A
R405A	R413A	R423A	R438A	AIR
R406A	R414A	R424A	R441A	AMARILLO
R407A	R414B	R425A	R442A	EKOFISK
R407B	R415A	R426A	R443A	GLFCOAST
R407C	R415B	R427A	R444A	HIGHCO2
R407D	R416A	R428A	R500	HIGHN2
R407E	R417A	R429A	R501	
R407F	R418A	R430A	R502	

Examples:

Here are some examples of making pure fluid property calls at room temperature conditions ($T_{rm} := 295$ and $P_{rm} := 0.101325$). Values passed must be scaled to modified SI (using [MPa] and [kJ]), but cannot have Mathcad units attached, unless they are converted in the call.

$fluid := \text{“Water”}$

$\text{rp\_getname}(fluid, 1) = \text{“Water”}$ $\quad$ $\text{rp\_getcasn}(fluid, 1) = \text{“7732-18-5”}$

$T_{crit} := \text{rp\_tcrit}(fluid, 0) = 647.096$ $\quad$ $P_{crit} := \text{rp\_pcrit}(fluid, 0) = 22$

$T_2 := 0.5 \cdot T_{crit} = 323.548$

Density @ T_2 and P_{crit} $\quad$ $\text{rp\_rhotp}(fluid, T_2, P_{crit}) = 997.211$

Enthalpy @ T_2 and P_{crit} $\quad$ $\text{rp\_htp}(fluid, T_2, P_{crit}) = 229.923$

Saturated Liquid Enthalpy @ T_2 $\quad$ $\text{rp\_hft}(fluid, T_2) = 211.006$

Saturated Vapor Enthalpy @ T_2 $\quad$ $\text{rp\_hgt}(fluid, T_2) = 2592$

Chapter 4 High-Level API General Usage

The RefProp High-Level Add-in Functions

If ***RefProp 10*** is installed, the ***RefProp High-Level API*** functions will be registered by Mathcad Prime. These functions follow the same naming convention as the Legacy API functions described above. However, there are far fewer of them and the new High-Level API provides a much simplified interface to REFPROP.

The REFPROP Function

The public function **REFPROPdll** and its companion functions, **REFPROP1dll** and **REFPROP2dll**, are provided as one-stop functions that can be used to access all of the features of REFPROP through one single function. This high-level API call may be the only function that some users require to obtain the physical properties needed. It is versatile enough that it can even provide fluid information and can be used to set and retrieve REFPROP program behavior.

This versatility comes at a performance cost, due to the string manipulation required in the underlying **REFPROPdll** Fortran code. REFPROP function calls can be 2 to 5 times slower than direct calls to the Legacy API functions. Most users, however, will not notice this performance hit unless making making tens of thousands of calls in succession.

Unfortunately, because of the rigidity in Mathcad's Custom Function parameter, calls to the **REFPROPdll** function have to be broken up into three separate Custom Functions.

1. rp_REFPROP - returns *arrays* of results, takes a fluid string.
2. rp_REFPROP1 - returns a *single scalar* result, takes no fluid string.
3. rp_REFPROPc - returns a *character string*, takes a fluid string.

These functions have been simplified to take just five input parameters, as demonstrated by the rp_REFPROP parameter list,

$$\text{rp\_REFPROP}(\text{“hFld”}, \text{“hIn”}, \text{“hOut”}, a, b)$$

with the exception of rp_REFPROP1, which does not require the first input parameter string, *hFld*.

Where,

hFld = The fluid/mixture string to be loaded. May be a blank or empty string if the previously loaded fluid/mixture is to be used.

hIn = The input state point specification. Valid codes are T, P, D, E, H, S, and Q (temperature, pressure, density, energy, enthalpy, entropy, and quality). Two of these should be sent together to identify the contents of the a and b variables. The numeric inputs should be in the order specified in hIn. Other special codes are also available[1].

hOut = The output code string for requested values. Multiple outputs can be requested (except for $rp\_REFPROP1$ and $rp\_REFPROPc$ calls) and must be separated by spaces, commas, semicolons, or bars (pipe character). Mixed delimiters should not be used. See ALLPROPS documentation for valid output codes and REFPROP documentation for other special output codes[1].

a = First numeric input, as specified in hIn. May be zero or arbitrary when retrieving non-state-dependent values. No Mathcad units applied, but scaled to the modified SI units.

b = Second numeric input, as specified in hIn. May be zero or arbitrary when retrieving non-state-dependent values. No Mathcad units applied, but scaled to the modified SI units.

In order to streamline the input parameter list, some simplifying assumptions have been made, as follows, for the many remaining **REFPROPdll** function parameters.

- $iUnits$ - Since Mathcad can handle units, and to facilitate implementation of unit wrapper functions, this parameter is maintained internally and set to a Modified MASS SI units system (with Thermal Conductivity in [W/m-K] and Surface Tension in [N/m]). This value may not be changed in rp_REFPROPx calls.

- $iMass$ - Set internally to 0, such that all composition values are set and returned on a molar basis. Mole fraction arrays can easily be converted to Mass fraction arrays in Mathcad.

- $iFlag$ - Set internally to 1. SATSPLN is always called for new mixtures. This mimics the behavior of the REFPROP GUI program.

[1] Current REFPROP DLL documentation should be consulted for the complete list

- *z* - To facilitate calling pure fluids and mixtures with the same function, the composition parameter is maintained internally and reset whenever a new fluid or mixture is loaded. Manual deviation from initial mixture compositions can be achieved using rp_setx and passing a new composition array.

- *Output* - This is a 200 element array containing the requested output values. A trimmed array containing only the number of values requested is returned from the function. This array is expanded if multi-component values are requested.

- *c* - Output value when calling rp_REFPROP1. Returned as a scalar value, instead of the *Output* array from rp_REFPROP calls.

- *hUnits* - This output parameter is ignored.

- *iUCode* - This output parameter is ignored except in special cases where it contains the return value, instead of the *Output* array.

- *x* and *y* - These output parameters, containing the liquid and vapor compositions in two-phase states, are ignored. They can be returned in the *Output* array by requesting compositions through the *hOut* codes XMOLE, YMOLE, XMASS, and YMASS.

- *x3* - Reserved output array for returning the composition of a second liquid phase for LLE or VLLE calculations. Ignored.

- *q* - This output parameter contains the vapor quality in two-phase states and is ignored. Vapor quality can be returned in the *Output* array by using the *hOut* codes QMOLE and/or QMASS.

- *ierr* and *herr* - Output parameters containing the return error code and corresponding error string. These are trapped and handled internally.

- **_length* - Input parameter specifying maximum lengths of the string parameters. Handled internally by the add-in DLL.

rp_REFPROP(*hFld*, *hIn*, *hOut*, *a*, *b*) Examples

Return Water density [kg/m³], molecular weight [gm/mol], specific enthalpy [kJ/kg], and Entropy [kJ/kg-K] at $T := 72\ °F$ and $P := 1\ atm$.

$$RP10 := \text{rp\_REFPROP}\left(\text{“Water”}, \text{“TP”}, \text{“D M H S”}, \frac{T}{K}, \frac{P}{MPa}\right) = \begin{bmatrix} 997.723 \\ 18.015 \\ 93.304 \\ 0.328 \end{bmatrix} \quad \begin{bmatrix} \text{“D”} \\ \text{“M”} \\ \text{“H”} \\ \text{“S”} \end{bmatrix}$$

To extract return values to a vector of individual variables:

$$\begin{bmatrix} \rho_l \\ h_l \\ tsat \end{bmatrix} := \text{rp\_REFPROP}\left(\text{“”}, \text{“PQ”}, \text{“D H T”}, \frac{P}{MPa}, 0\right)$$

$\rho_l = 958.367$ $h_l = 419.058$ $tsat = 373.124$

rp_REFPROP1(*hIn*, *hOut*, *a*, *b*) Examples

Returns single scalar values as specified in *hOut*, mimicking the exact behavior of the MS Excel add-in. Does not take a fluid string! Fluid must already be loaded through a call to **rp_REFPROP** or **rp_SETFLUIDS** (or rp_setup from the Legacy API).

$T = 72\ °F$ $P = 1\ atm$

$$\text{rp\_REFPROP1}\left(\text{“TP”}, \text{“D”}, \frac{T}{K}, \frac{P}{MPa}\right)\frac{kg}{m^3} = 62.286\ \frac{lb}{ft^3}$$

Density ***scalar*** returned; not a vector.

$$\text{rp\_REFPROP1}\left(\text{“TP”}, \text{“D M H S”}, \frac{T}{K}, \frac{P}{MPa}\right) = ?$$

Unknown error: Only_single_outputs_from_rp%_REFPROP1._Call_rp%_REFPROP.

Can't return multiple values from **rp_REFPROP1**

rp_REFPROPc(*hFld*, *hIn*, *hOut*, *a*, *b*) Examples

Same **REFPROPdll** call, but returns a **string** as specified by special codes in *hOut*. No values from the **Output** array parameter can be returned as scalar or array values from this function.

DLL# (as a string): $\text{rp\_REFPROPc}(\text{“”}, \text{“”}, \text{“DLL\#”}, 0, 0) = \text{“10.0.0.02”}$

Pure Fluid Name: $\text{rp\_REFPROPc}(\text{“H2O”}, \text{“”}, \text{“NAME”}, 0, 0) = \text{“Water”}$

Chemical Abstract Number: $\text{rp\_REFPROPc}(\text{“1BUTENE”}, \text{“”}, \text{“CAS\#”}, 0, 0) = \text{“106-98-9”}$

Chemical Formula: $\text{rp\_REFPROPc}(\text{“”}, \text{“”}, \text{“LONGNAME”}, 0, 0) = \text{“1-Butene”}$

Chemical Formula: $\text{rp\_REFPROPc}(\text{“”}, \text{“”}, \text{“CHEMFORM”}, 0, 0) = \text{“C4H8”}$

*See **Chapter 5 - Applying Units to RefProp Functions** for examples of using the unit wrapper function* REFPROP() *that makes the decision between these three functions automatically and handles inputs and outputs with Mathcad units applied.*

The SETFLUIDS Function

Fluid files (or a set of fluid files) can be loaded very simply with a call to rp_SETFLUIDS(*hFld*), which has only one parameter (the fluid string). For a single fluid, any other REFPROP calls can be made without providing the fluid string as shown here:

rp_SETFLUIDS("H2O") = 0 $\qquad$ *idum* := rp_SETFLUIDS("H2O")

The function will load the fluid and return the error code, which is always zero (otherwise an error is thrown). To hide the result, just assign it to a dummy variable.

Now, fluid info and properties can be retrieved without specifying a fluid.

rp_REFPROPc("","","NAME",0,0) = "Water" $\qquad$ $P = 1$ ***atm***

$Tsat := \text{rp\_REFPROP1}\left("PQ","T",\frac{P}{MPa},0\right)$ $K = 211.954$ ***°F***

If loading multiple fluids, either the fluids must be accessed individually by setting a single component flag, or rp_setx(*x*) must be called immediately following this call to set the mixture mole fraction.

ierr := rp_SETFLUIDS("Oxygen|Nitrogen|Argon|CO2") = 0

$$x_{Air4} := \begin{bmatrix} 0.20948 \\ 0.78084 \\ 0.00934 \\ 0.00034 \end{bmatrix} \qquad \text{rp\_setx}\left(x_{Air4}\right) = \text{"Composition set"}$$

ncomp := rp_REFPROP1("","NCOMP",0,0) = 4

$$\begin{bmatrix} T_{crit} \\ T_{maxP} \\ T_{maxT} \\ P_{maxP} \\ P_{maxT} \end{bmatrix} := \text{rp\_REFPROP}("","","TC TMAXP TMAXT PMAXP PMAXT",0,0)$$

$T_{crit} = 132.973$ $\qquad$ $T_{maxP} = 133.015$ $\qquad$ $T_{maxT} = 123.18$

$P_{maxP} = 3.871$ $\qquad$ $P_{maxT} = 1.387$

The SETMIXTURE Function

Similar to loading pure fluids with $\text{rp\_SETFLUIDS}(hFld)$, the function $\text{rp\_SETMIXTURE}(hMixNme)$ will load a mixture file. The mixture file name, $hMixNme$, must be a non-empty string containing a mixture file name from the REFPROP\MIXTURES\ subdirectory, but does not have to include the "*.mix*" extension in the file name.

This function makes a direct call to **SETMIXTUREdll** and the molar composition array, z, is returned to the user; however, it is also stored and managed internally by the Mathcad add-in DLL. The composition can always be retrieved later through rp_REFPROP or with the legacy API call rp_getx.

$\text{rp\_FLAGS}(\text{“No Warnings”}, 1) = 1$ *Hides warnings when retrieving composition.*

$$z := \text{rp\_SETMIXTURE}(\text{“R440A”}) = \begin{bmatrix} 0.00901 \\ 0.01039 \\ 0.98060 \end{bmatrix}$$

$$x_{mole} := \text{rp\_REFPROP}(\text{“”}, \text{“”}, \text{“XMOLE”}, 0, 0) = \begin{bmatrix} 0.00901 \\ 0.01039 \\ 0.98060 \end{bmatrix} \qquad \text{rp\_getx}(\text{“”}) = \begin{bmatrix} 0.00901 \\ 0.01039 \\ 0.98060 \end{bmatrix}$$

$$x_{mass} := \text{rp\_REFPROP}(\text{“”}, \text{“”}, \text{“XMASS”}, 0, 0) = \begin{bmatrix} 0.00600 \\ 0.01600 \\ 0.97800 \end{bmatrix}$$

$$ncomp := \text{length}(z) = 3 \qquad icomp := 1 \,..\, ncomp$$

$$Fluids_{icomp-1} := \text{rp\_REFPROPc}(\text{“”}, \text{“”}, \text{format}(\text{“Name}(\{0\})\text{”}, icomp), 0, 0)$$

$$Fluids = \begin{bmatrix} \text{“Propane”} \\ \text{“R134a”} \\ \text{“R152a”} \end{bmatrix}$$

The ALLPROPS0 Function (beta)

The ALLPRPOPS0dll function was provided with the power user in mind. REFPROP calls require text parsing of the $hFld$, hIn, and $hOut$ strings, which can be extremely slow in the underlying Fortran code. ALLPROPS0dll makes single-phase calls with Temperature and Density inputs for a pre-loaded fluid/mixture, eliminating the need to pass the $hFld$ and hIn strings. Output parameters are specified as a numeric array populated with the enumerated codes for each property requested, making use of GETENUMdll to retrieve these codes.

The wrapper function $\text{rp\_ALLPROPS0}(iOut, T, D, z, Output)$ is provided as a ***beta feature*** for fast single-phase function that returns the enumerated properties provided in the array as a function of T, D, and z. Its goal is to reach the speeds of the low level functions, but allows many properties to be returned at once without any string manipulation. To use the function, the enumerated values of the requested output property strings must be captured and provided in the array.

An OUTPUT array must be provided, either initialized to the length of the desired number of outputs, or containing the values from a previous call to ALLPROPS so that the blanks can be filled in (as specified in the $iOut$ array).

Temperature and density must be provided in the *default, molar* REFPROP units of [K] and [mol/L], respectively. Composition, z, must be provided in mole fractions.

Example:

Set the $iOut$ array with the desired enumerated property codes and initialize the $Output$ array accordingly.

$$iOut := \begin{bmatrix} \text{rp\_GETENUM}(0, \text{“P”}) \\ \text{rp\_GETENUM}(0, \text{“H”}) \\ \text{rp\_GETENUM}(0, \text{“E”}) \\ \text{rp\_GETENUM}(0, \text{“S”}) \\ \text{rp\_GETENUM}(0, \text{“Cp”}) \\ \text{rp\_GETENUM}(0, \text{“Cv”}) \\ \text{rp\_GETENUM}(0, \text{“W”}) \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \\ 5 \\ 7 \\ 12 \\ 11 \\ 14 \end{bmatrix} \qquad OUTPUT_{\text{last}(iOut)} := 0 \qquad OUTPUT = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Load a fluid (either with $rp\_SETFLUIDS$ or through another property call and set values for single-phase Temperature, Density, and composition array.

$$T := 72 \ \textbf{°F} \qquad P := 1 \ \textbf{atm} \qquad z := [1]$$

$$D := \text{rp\_rhotp}\left(\text{“H2O”}, \frac{T}{\textbf{K}}, \frac{P}{\textbf{MPa}}\right) \frac{\textbf{kg}}{\textbf{m}^3}$$

$$mw := \text{rp\_wmol}(\text{“H2O”}, 0) \ \frac{\textbf{gm}}{\textbf{mol}} \qquad D_{molar} := D \div mw = 55.382 \ \frac{\textbf{mol}}{\textbf{L}}$$

Call ALLPROPS0 with these values/arrays.

$$\begin{bmatrix} P_{ap} \\ H_{ap} \\ E_{ap} \\ S_{ap} \\ Cp_{ap} \\ Cv_{ap} \\ W_{ap} \end{bmatrix} := \text{rp\_ALLPROPS0}\left(iOut, \frac{T}{\boldsymbol{K}}, \frac{D_{molar}}{\boldsymbol{mol \cdot L^{-1}}}, z, OUTPUT\right) = \begin{bmatrix} 101.325 \\ 1.681 \cdot 10^3 \\ 1.679 \cdot 10^3 \\ 5.91 \\ 75.352 \\ 74.734 \\ 1.489 \cdot 10^3 \end{bmatrix}$$

Make the same call using *rp_REFPROP* and set the corresponding units.

$$\begin{bmatrix} P_{rp} \\ H_{rp} \\ E_{rp} \\ S_{rp} \\ Cp_{rp} \\ Cv_{rp} \\ W_{rp} \end{bmatrix} := \text{rp\_REFPROP}(\text{“”}, \text{“TD”}, \text{“P H E S Cp Cv W”}, T, D) \overrightarrow{\begin{bmatrix} \boldsymbol{MPa} \\ \boldsymbol{kJ\ kg^{-1}} \\ \boldsymbol{kJ\ kg^{-1}} \\ \boldsymbol{kJ\ kg^{-1}\ K^{-1}} \\ \boldsymbol{kJ\ kg^{-1}\ K^{-1}} \\ \boldsymbol{kJ\ kg^{-1}\ K^{-1}} \\ \boldsymbol{m\ s^{-1}} \end{bmatrix}}$$

Compare Results

$P_{ap} = 101.325$ $\quad P_{rp} = 101.325\ \boldsymbol{kPa}$

$H_{ap} = 1680.889$ $\quad H_{rp} \cdot mw = 1680.889\ \boldsymbol{J\ mol^{-1}}$

$E_{ap} = 1679.06$ $\quad E_{rp} \cdot mw = 1679.06\ \boldsymbol{J\ mol^{-1}}$

$S_{ap} = 5.91$ $\quad S_{rp} \cdot mw = 5.91\ \boldsymbol{J\ mol^{-1}\ K^{-1}}$

$Cp_{ap} = 75.352$ $\quad Cp_{rp} \cdot mw = 75.352\ \boldsymbol{J\ mol^{-1}\ K^{-1}}$

$Cv_{ap} = 74.734$ $\quad Cv_{rp} \cdot mw = 74.734\ \boldsymbol{J\ mol^{-1}\ K^{-1}}$

$W_{ap} = 1488.971$ $\quad W_{rp} = 1488.971\ \boldsymbol{m\ s^{-1}}$

Now, check the timing of each method over 1,000 calls.

$t_0 := \text{time}(0)\ s$

$$\left\| \begin{array}{l} \text{for } i \in \text{ORIGIN} .. \text{ORIGIN} + 1000 \\ \left\| M \leftarrow \text{rp\_ALLPROPS0}\left(iOut, \frac{T}{K}, \frac{D_{molar}}{mol \cdot L^{-1}}, z, OUTPUT\right) \right\| \\ M \end{array} \right\|$$

$\Delta t_{ap} := \text{time}(0)\ s - t_0 = 9.000\ ms$

$t_1 := \text{time}(0)\ s$

$$\left\| \begin{array}{l} \text{for } i \in ORIGIN .. ORIGIN + 1000 \\ \left\| M \leftarrow rp\_REFPROP(\text{“”}, \text{“TD”}, \text{“P H E S Cp Cv W”}, T, D) \right\| \\ M \end{array} \right\|$$

$\Delta t_{rp} := \text{time}(0)\ s - t_0 = 94.000\ ms$

$SpeedUp_{ap} := \frac{\Delta t_{rp}}{\Delta t_{ap}} = 10.4$

The rp_ALLPROPS0 calls are on the order of ***10X faster*** than calling rp_REFPROP . However, based on the algorithm in which this function is being used, performance improvement may be less than the raw function calls shown above. Keep in mind,

- *rp_ALLPROPS0* can only be called for single phase solutions.
- *rp_ALLPROPS0* can only be called for temperature-density state points. Performance will degrade if density has to also be determined first for every call of the function
- Changing the composition, z, between calls to *rp_ALLPROPS0* does not create new saturation curves with calls to SATSPLN.
- Any scaling or unit application/removal before and after the call, will be performed in Mathcad equations and will slow down performance of the overall algorithm.
- The *iOut* array has to be built in Mathcad, instead of parsing out the *hOut* string in the faster, low-level DLL code.

Chapter 5 Applying Units to RefProp Functions

Creating User Function Wrappers

Parameters to the ***RefProp*** functions must be scaled to the required units (as specified in the help text for each function under Insert Function). However, these parameters cannot have actual Mathcad Units attached to them (i.e. they should be unit-less numbers); otherwise incorrect values may be returned as a function result. The results, too, will be returned in the specified units.

To enable usage of Mathcad units and handle unit conversions automatically, create a Mathcad user function that:

1. Divides all of the parameters by the units required by the ***RefProp*** function.
2. Multiplies the result by the units of the ***RefProp*** function return value.

Example:

The RefProp function $\mathbf{rp\_hfp}(fluidstr, p)$ returns the saturated liquid enthalpy in [***kJ/kg***]. The parameter, p, is the input saturation pressure [***MPa***].

Create a user function to handle the units according to the instructions above:

$$h_{lsat}(fluidstr, p) := rp\_hfp\left(fluidstr, \frac{p}{MPa}\right) \cdot \frac{kJ}{kg}$$

Note: In Mathcad, kJ is not a pre-defined unit, so it is defined in the units reference worksheet (see below).

We can now use the wrapper function, instead of calling the ***RefProp*** function directly.

$fluid := \text{“WATER”}$ *Specify Water as the fluid*

$P := 1200 \cdot psi$ *Pressure can then be supplied in any pressure units.*

$h_{lsat}(fluid, P) = 571.938 \ \frac{BTU}{lb}$ *The result can be displayed in any specific energy units.*

But there is an easier way...

Using the Supplied Reference Sheet

Creating function wrappers for all of the functions required can be tedious and provide for inconsistencies between worksheets. A complete set of function wrappers has been supplied in the reference worksheet, **Refprop_Units.mcdx**, in the ***Units*** directory (see Appendix C).

The **Refprop_Units.mcdx** worksheet can be ***referenced*** near the top of any worksheet in which you want to use the wrapper functions.

To reference the units worksheet:

1. Choose **Input/Output** from the main menu.
2. Select the Include Worksheet button on the ribbon.

The reference will show up in the worksheet at the cursor location as shown:

Include << .\RefProp_units.mcdx

Click on the [Include <<] button and browse to the local or shared location of the **Refprop_Units.mcdx** file. The reference inserted can be edited to make it a relative reference to the local worksheet, or left as an absolute reference to a shared location for permanent/shared access.

All of the pre-defined, Legacy API wrapper functions in the reference worksheet can now be used:

$\text{Name}(fluid, 1) = \text{“Water”}$

$\text{CASN}(fluid, 1) = \text{“7732-18-5”}$

$T_c(fluid, 0) = 705.103\ °F$

$P_c(fluid, 0) = 3200\ psi$

$T_2 := 0.5 \cdot T_c(fluid, 0)$

$\rho_{tp}(fluid, T_2, P_c(fluid, 0)) = 62.254\ \frac{lb}{ft^3}$

$h_{tp}(fluid, T_2, P_c(fluid, 0)) = 98.849\ \frac{BTU}{lb}$

$h_{ft}(fluid, T_2) = 90.716\ \frac{BTU}{lb}$

$h_{gt}(fluid, T_2) = 1114.4\ \frac{BTU}{lb}$

The available functions in the reference worksheet are listed below. Keep in mind that each function requires, as its first parameter, a "**fluid string**".

RefProp Property Notation

- Fluid Properties
  - ρ Density
  - h Enthalpy
  - u Internal Energy
  - s Entropy
  - C_p Isobaric Specific Heat
  - C_v Isochoric Specific Heat
  - μ Kinematic Viscosity
  - v Dynamic Viscosity (μ/ρ)
  - k Thermal Conductivity
  - Pr Prandtl Number.
- Subscript notation for thermodynamic state:
  - tp Sub-cooled liquid or Super-heated vapor as a function of temperature (t) and pressure (p)
  - ft Saturated Liquid as a function of temperature (t)
  - fp Saturated Liquid as a function of pressure (p)
  - gt Saturated Vapor as a function of temperature (t)
  - gp Saturated Liquid as a function of pressure (p)

Add-in Property Functions

Property	Sub-cooled liquid/ super-heated vapor	Saturated Liquid	Saturated Vapor
Density	$\rho_{tp}(fl,T,P)$*	$\rho_{fp}(fl,P),\rho_{ft}(fl,T)$	$\rho_{gp}(fl,P),\rho_{gt}(fl,T)$
Enthalpy	$h_{tp}(fl,T,P)$	$h_{fp}(fl,P),h_{ft}(fl,T)$	$h_{gp}(fl,P),h_{gt}(fl,T)$
Internal Energy	$u_{tp}(fl,T,P)$	$u_{fp}(fl,P),\rho_{ft}(fl,T)$	$u_{gp}(fl,P),\rho_{gt}(fl,T)$
Entropy	$s_{tp}(fl,T,P)$	$s_{fp}(fl,P),s_{ft}(fl,T)$	$s_{gp}(fl,P),s_{gt}(fl,T)$
Isobaric Specific Heat	$C_{ptp}(fl,T,P)$	$C_{pfp}(fl,P),C_{pft}(fl,T)$	$C_{pgp}(fl,P),C_{pgt}(fl,T)$
Isochoric Specific Heat	$C_{vtp}(fl,T,P)$	$C_{vfp}(fl,P),C_{vft}(fl,T)$	$C_{vgp}(fl,P),C_{vgt}(fl,T)$
Kinematic Viscosity	$\mu_{tp}(fl,T,P)$	$\mu_{fp}(fl,P),\mu_{ft}(fl,T)$	$\mu_{gp}(fl,P),\mu_{gt}(fl,T)$
Dynamic Viscosity	$v_{tp}(fl,T,P)$	$v_{fp}(fl,P),v_{ft}(fl,T)$	$v_{gp}(fl,P),v_{gt}(fl,T)$
Thermal Conductivity	$k_{tp}(fl,T,P)$	$k_{fp}(fl,P),k_{ft}(fl,T)$	$k_{gp}(fl,P),k_{gt}(fl,T)$

** The parameter, fl, is the fluid string required by the REFPROP functions.*

Two-Phase Functions

Property	Function	Property	Function
Saturation Temperature	$T_{sat}(fl,P)$	Vapor Quality	$X_{ph}(fl,P,H)$
Saturation Pressure	$P_{sat}(fl,P)$		$X_{ps}(fl,P,S)$
			$X_{pp}(fl,P,\rho)$
Surface Tension	$\sigma_t(fl,T)$		$X_{pu}(fl,P,U)$
			$X_{th}(fl,T,H)$
Density Difference $[\rho_f - \rho_g]$	$\Delta\rho_t(fl,T)$, or $\Delta\rho_p(fl,P)$		$X_{ts}(fl,T,S)$
			$X_{tp}(fl,T,\rho)$
			$X_{tu}(fl,T,U)$
Heat of Vaporization $[h_g - h_f]$	$h_{fg}(fl,T)$		

** The parameter, fl, is the fluid string required by the REFPROP functions.*

Additional Properties

Isentropic Exponent, γ	$\gamma(fl,T,P)$ *
Thermal Expansion Coef., β	$\beta(fl,T,P)$
Compressibility, Z	$Z_{tp}(fl,T,P)$, $Z_c(fl,i)$ ** $Z_{ft}(fl,T)$, $Z_{fp}(fl,P)$ $Z_{gt}(fl,T)$, $Z_{gp}(fl,P)$
Morton Number, Mo	$Mo(fl,P)$

* *The parameter, fl, is the fluid string required by the REFPROP functions.*

** *The parameter, i, is 0 for a mixture or pure fluid, or an integer component number to retrieve the individual, pure-fluid, component values in a mixture.*

Reverse Functions

$T_{ph}(fl, P, H)$ *	$T_{hs}(fl, H, S)$
$T_{ps}(fl, P, S)$	$P_{hs}(fl, H, S)$
$P_{th}(fl, T, H, r)$ **	$P_{ts}(fl, T, S)$
$\rho_{th}(fl, T, H, r)$ **	$\rho_{ts}(fl, T, S)$
$\rho_{ph}(fl, P, H)$	$\rho_{ps}(fl, P, S)$
$h_{ph}(fl, P, H)$	$s_{ps}(fl, P, S)$
$h_{ts}(fl, T, S)$	$s_{th}(fl, T, H)$ **

* *The parameter, fl, is the fluid string required by the REFPROP functions.*
** *Functions of Temperature (T) and enthalpy (H) can be multi-valued. The last parameter, r, should be unity(1) for the lower root, or two (2) for the upper root.*

Add-in Property Constants

Description	Function
Critical Point	$T_c(fl, i), P_c(fl, i), \rho_c(fl, i)$ *
Triple Point	$T_t(fl, i), P_t(fl, i), \rho_t(fl, i)$ **
Molecular Weight	$MW(fl, i)$ *
Ideal Gas Constant	$R_g(fl, i)$ ***

* *The parameter, fl, is the fluid string required by the REFPROP functions.*
The parameter, i , is 0 for a mixture or pure fluid, or an integer component number to retrieve the individual, pure-fluid, component values in a mixture.
** *The triple point values for Pressure (P) and Density (ρ) are the liquid phase values.*
*** *The ideal gas constant for a particular fluid EOS may vary slightly from Mathcad's built-in value.*

High-Level REFPROP with Units

As stated in Chapter 4, the direct Custom Function calls to the **REFPROPdll** function have to be broken up into three separate functions.

1. **rp_REFPROP** - returns arrays of results, takes a fluid string.
2. **rp_REFPROP1** - returns a single scalar result, takes no fluid string.
3. **rp_REFPROPc** - returns a character string, takes a fluid string.

None of these functions, being DLL calls, takes or returns values with Mathcad units applied. Unlike Custom Functions provided in the DLL, Mathcad User function can handle unit inputs and outputs and can return any data type, depending on the specified input parameters.

The **Refprop_Units.mcdx** include file contains a single universal **REFPROP** function that will call the necessary Custom Function, depending on the input parameters supplied. The function $\mathbf{REFPROP}(hFld, hIn, hOut, a, b)$

- Handles unit inputs and application of units to the output values.
- Returns single values a scalar.
- Multiple outputs are returned in an array; expanded to include multi-component values for mixtures and then compressed into component sub-arrays when required.
- Character based information is returned as a string.
- Requires a fluid string ($hFld$) as the first parameter, but may be passed an empty string ("") to use the pre-loaded fluid.

Here are examples of the three call methods for the universal REFPROP function.

1. Multiple numeric outputs:

$$\begin{bmatrix} P \\ D \\ H \\ mw \end{bmatrix} := \text{REFPROP}(\text{"Water"}, \text{"TQ"}, \text{"P D H M"}, 200\ °F, 0)$$

$$P = 0.08\ MPa \qquad D = 60.12\ \frac{lb}{ft^3} \qquad H = 168.13\ \frac{BTU}{lb} \qquad mw = 0.04\ \frac{lb}{mol}$$

2. Multiple numeric outputs (mixture):

$$\begin{bmatrix} P \\ D \\ H \\ X_{mol} \end{bmatrix} := \text{REFPROP}(\text{"R401A"}, \text{"TQ"}, \text{"P D H XMOLE"}, 72\ °F, 0)$$

The extracted results are:

$P = 0.718\ \textbf{\textit{MPa}}$ $D = 74.865\ \frac{\textbf{\textit{lb}}}{\textbf{\textit{ft}}^3}$ $H = 97.544\ \frac{\textbf{\textit{BTU}}}{\textbf{\textit{lb}}}$ $X_{mol} = \begin{bmatrix} 0.579 \\ 0.186 \\ 0.235 \end{bmatrix}$

3. Single numeric outputs (using fluid from previous call):

$\text{REFPROP}(\text{“”}, \text{“TQ”}, \text{“P”}, 150\ \textbf{\textit{°F}}, 0) = 2.079\ \textbf{\textit{MPa}}$

4. String outputs:

$\text{REFPROP}(\text{“R401A”}, \text{“”}, \text{“NAME(0)”}, 0, 0) = \text{“R401A [R-22/152a/124 (53/13/34)]”}$

$\text{REFPROP}(\text{“Water”}, \text{“”}, \text{“CAS\#”}, 0, 0) = \text{“7732-18-5”}$

The input parameters, hIn and $hOut$, can be any valid strings as specified by the NIST REFPROP manual for the **REFPROPdll** function. Only single string outputs can be requested and multiple output types cannot be mixed (i.e. numeric and string).

Unit Wrapper vs. non-Wrapper Calls

Speed may be an issue over calling the REFPROP wrapper functions directly, or even the Legacy API wrapper functions. Timings vary depending on CPU and other process running on the machine.

If making single, or a dozen, property calls, wrapper function performance will not be noticeably different from calling the Custom Functions directly (milliseconds slower). However, if many thousands of calls are being made, say for generating plots or solution and optimization of thermodynamic cycles:

1. Make calls directly to the **rp_REFPROP** function, without units applied will be almost 2X faster than using the universal **REFPROP** function with units.

2. If requesting single property requests, calling **rp_REFPROP1** directly will be up to 5X faster than using the universal **REFPROP** function with units.

3. For ~5X to 10X performance increase over using the universal **REFPROP** unit wrapper function, call the Legacy API functions with or without the unit wrapper functions.

Chapter 6 Reverse Functions (Legacy API)

Since all of the ***RefProp*** functions rely on the Helmholtz free energy surface (f), they are all constructed as direct functions of Temperature and Density (just like f). While every function of Temperature and Pressure starts with a reverse function to calculate density (so that the direct functions may be used), there are some additional functions that are of use in standard thermodynamic energy cycle calculations. These functions require more complex iteration and convergence behavior and so may require more computational time to evaluate. This should be kept in mind when using these functions, especially if called repeatedly within a loop.

Additionally, some of the functions below are multi-valued; specifically the Pressure, Density, and Entropy functions of Temperature and Enthalpy. This occurs because the curves may fold back on themselves (*Figure 1*); creating an upper and lower density intersection point for a given input (H) parameter (see Figure 1 below). In these situations, there is an extra parameter, r, in the parameter list that indicates whether the lower density root ($r = 1$), or the upper root ($r = 2$) is requested.

The reverse functions are categorized below by return value. The first parameter in all of these functions is a "**fluid string**" variable that identifies the fluid material to be used for the property calculations.

Following the standard notation, the reverse function names all begin with ***rp_*** followed by the returned property, which is followed by the two input parameters in the order expected in the parameter list.

A list of the various reverse functions is shown below.

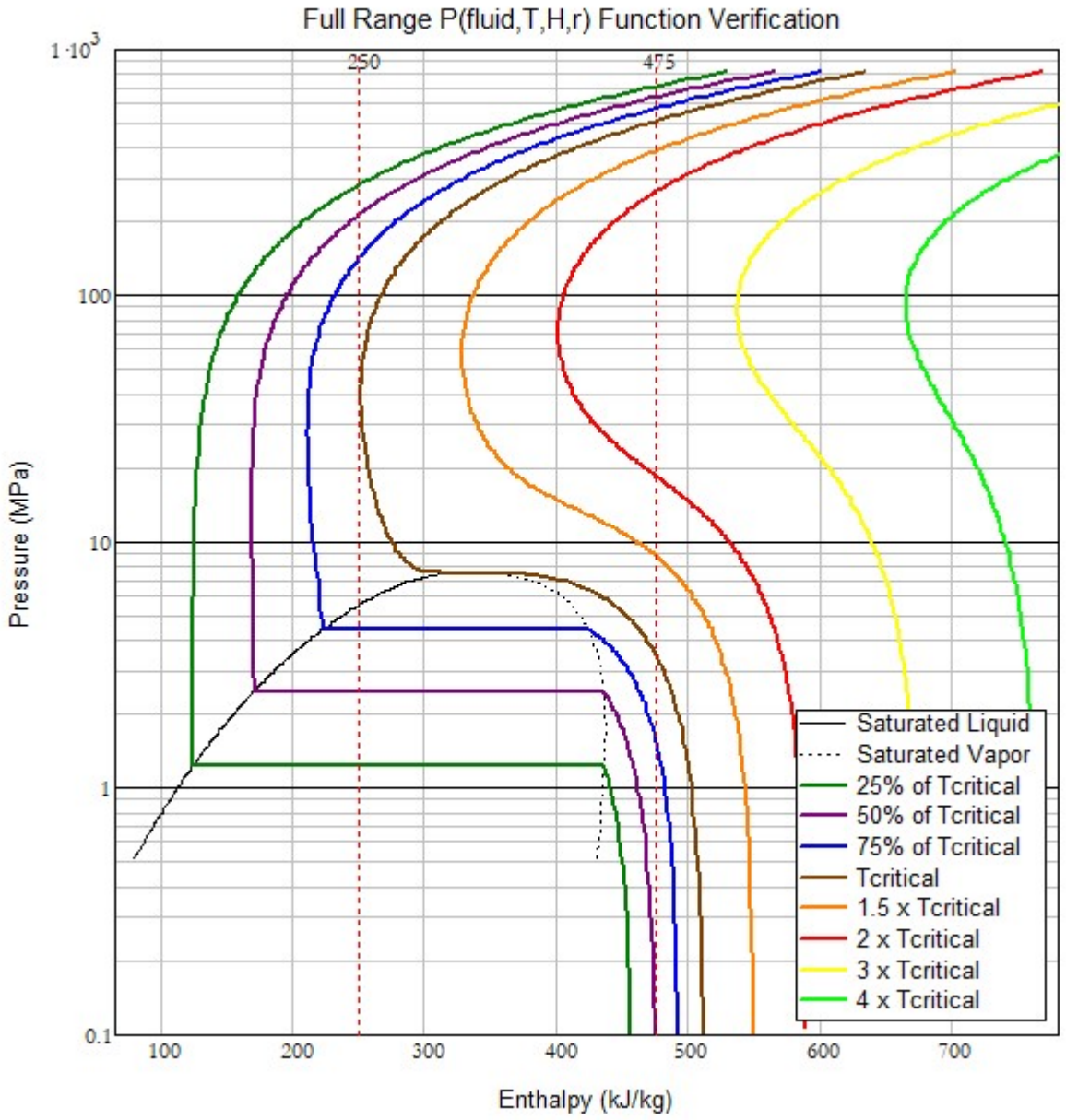


Figure 1 : *Pressure-Enthalpy plot showing double valued isotherms for given values of Enthalpy, H. The higher density roots usually occur at very high pressures, way above the critical point. However, at low enthalpy and temperature values, dual roots can occur in the sub-cooled liquid region.*

Temperature Functions, t

Raw Function	Application of Units
$\text{rp\_tph}(fl,t,h)$	$T_{ph}(fl,p,h) := \text{rp\_tph}\left(fl, \frac{p}{MPa}, h \cdot \frac{kg}{kJ}\right) \cdot K$
$\text{rp\_tps}(fl,p,s)$	$T_{ps}(fl,p,s) := \text{rp\_tps}\left(fl, \frac{p}{MPa}, s \cdot \frac{kg \cdot K}{kJ}\right) \cdot K$
$\text{rp\_ths}(fl,h,s)$	$T_{hs}(fl,h,s) := \text{rp\_ths}\left(fl, h \cdot \frac{kg}{kJ}, s \cdot \frac{kg \cdot K}{kJ}\right) \cdot K$

Pressure Functions, p

Raw Function	Application of Units
$\text{rp\_pth}(fl,t,h,r)$	$P_{th}(fl,t,h,r) := \text{rp\_pth}\left(fl, \frac{t}{K}, h \cdot \frac{kg}{kJ}, r\right) \cdot MPa$
$\text{rp\_pts}(fl,t,s)$	$P_{ts}(fl,t,s) := \text{rp\_pts}\left(fl, \frac{t}{K}, s \cdot \frac{kg \cdot K}{kJ}\right) \cdot MPa$
$\text{rp\_phs}(fl,h \cdot s)$	$P_{hs}(fl,h,s) := \text{rp\_phs}\left(fl, h \cdot \frac{kg}{kJ}, s \cdot \frac{kg \cdot K}{kJ}\right) \cdot MPa$

Density Functions, rho

Raw Function	Application of Units
$\text{rp\_rhoth}(fl,t,h \cdot r)$	$\rho_{th}(fl,t,h,r) := \text{rp\_rhoth}\left(fl, \frac{t}{K}, h \cdot \frac{kg}{kJ}, r\right) \cdot \frac{kg}{m^3}$
$\text{rp\_rhoph}(fl,p,h)$	$\rho_{ph}(fl,p,h) := \text{rp\_rhoph}\left(fl, \frac{p}{MPa}, h \cdot \frac{kg}{kJ}\right) \cdot \frac{kg}{m^3}$
$\text{rp\_rhots}(fl,t,s)$	$\rho_{ts}(fl,t,s) := \text{rp\_rhots}(fl,t,s) \cdot \frac{kg}{m^3}$
$\text{rp\_rhops}(fl,p,s)$	$\rho_{ps}(fl,p,s) := \text{rp\_rhops}\left(fl, \frac{p}{MPa}, s \cdot \frac{kg \cdot K}{kJ}\right) \cdot \frac{kg}{m^3}$

Enthalpy Functions, h

Raw Function	**Application of Units**
$\text{rp\_hps}(fl, p, s)$	$h_{ps}(fl, p, s) := \text{rp\_hps}\left(fl, \frac{p}{MPa}, s \cdot \frac{kg \cdot K}{kJ}\right) \cdot \frac{kJ}{kg}$
$\text{rp\_hts}(fl, t, h)$	$h_{ts}(fl, t, h) := \text{rp\_hts}\left(fl, \frac{t}{K}, h \cdot \frac{kg \cdot K}{kJ}\right) \cdot \frac{kJ}{kg}$

Entropy Functions, s

Raw Function	**Application of Units**
$\text{rp\_sph}(fl, p, h)$	$s_{ph}(fl, p, h) := \text{rp\_sph}\left(fl, \frac{p}{MPa}, h \cdot \frac{kg}{kJ}\right) \cdot \frac{kJ}{kg \cdot K}$
$\text{rp\_sth}(fl, t, h, r)$	$s_{th}(fl, t, h, r) := \text{rp\_sth}\left(fl, \frac{t}{K}, h \cdot \frac{kg}{kJ}, r\right) \cdot \frac{kJ}{kg \cdot K}$

Chapter 7 Utility Functions

Various utility functions are also provided through the Mathcad ***RefProp*** interface. These routines are provided for verification of the loaded material files, retrieval of material parameters, and retrieval of the version of the NIST ***RefProp*** library that has been loaded. Alternative High-Level call through the REFPROP10 interface is provided in each case if it exists.

Mathcad RefProp Interface Add-in Version

This function returns a string containing the version number of the Mathcad ***RefProp*** interface DLL. This routine is provided for version control, and to ensure that specific calculations are prepared and viewed with the same version of the Add-In DLL.

Function Definition

$\text{rp\_getvers}(dummy)$ *NOTE: The dummy parameter can by any integer, i.e. 0.*

Example

$\text{rp\_getvers}(0) = \text{“RefProp Mathcad Add-in: Version 2.1.0”}$

NIST RefProp Library Version

This function returns a string containing the version number of the ***NIST RefProp*** library. This string is extracted from the database file location auto-selected by the Mathcad ***RefProp*** Add-in or the environment variable, ***NIST_PATH***.

Function Definition

$\text{rp\_getNIST}(dummy)$ *NOTE: The dummy parameter can by any integer, i.e. 0.*

$hOut = \text{“DLL\#”}$ *High-Level call through* rp_REFPROPx.

Example

$\text{rp\_getNIST}(0) = \text{“NIST RefProp Library: Version 10.0.0.2”}$

$\text{rp\_REFPROPc}(\text{“”}, \text{“”}, \text{“DLL\#”}, 0, 0) = \text{“10.0.0.02”}$

Path to Loaded REFPROP.DLL and /FLUIDS, /MIXTURES Directories

For debugging purposes, it can be helpful to know where Mathcad found the REFPROP dynamic linked library (DLL). This function shows the full path to the DLL that was loaded when Mathcad initialized the RefProp Add-in functions.

$\text{rp\_getpath}(0) = \text{“C:\Program Files (x86)\REFPROP”}$

Note that if Mathcad cannot find the REFPROP (DLL), a the user is warned with a pop-up message and the RefProp Add-in functions will not be loaded and registered into the Mathcad interface.

An alternate location for the /FLUIDS and /MIXTURES directories can be set with the $\text{rp\_setpath}(\text{“path-location”})$ function. The string for the path location can be set by browsing for an arbitrary fluid or mixture file using Mathcad's Data Filename browser. The parent directory will be the actual directory set.

Ideal Gas Constant, $\mathbf{R_{gas}}$

For some fluid materials, the value of the ideal gas constant may vary slightly. This utility function will report the Ideal Gas Constant, R_{gas}, in units of ***J/mol-K*** as listed for a specific "**fluid string**".

Function Definition

$\text{rp\_rgas}(fluid, comp)$

Application of Units

$R_{gas}(fluid, comp) := \text{rp\_rgas}(fluid, comp) \cdot \frac{J}{mol \cdot K}$

NOTE: The additional parameter, ***comp****, is a component number for a mixture. Since mixtures are not yet implemented, this number should be 1 or 0.*

$hOut = \text{“R”}$

High-Level call through rp_REFPROPx.

Example

$R_{gas}(\text{“WATER.fld”}, 0) = 8.3144 \ \frac{J}{mol \cdot K}$ Molar basis.

$\text{REFPROP}(\text{“”}, \text{“”}, \text{“R”}, 0, 0) = 461.518 \ \frac{J}{kg \ K}$ Mass basis.

Molecular Weight,

This function extracts the molecular weight of the material specified by the "**fluid string**", in units of ***gm/mol***.

Function Definition

$\text{rp\_wmol}(fl, comp)$

Application of Units

$\text{WM}(fl, comp) := \text{rp\_wmol}(fl, comp) \cdot \frac{gm}{mol}$

NOTE: The additional parameter, ***comp****, is a component number for a mixture. Since mixtures are not yet implemented, this number should be 1 or 0.*

$hOut = \text{“M”}$

High-Level call through rp_REFPROPx.

Example

$\text{WM}(\text{“WATER.FLD”}, 0) = 18.0153 \ \frac{gm}{mol}$

$\text{WM}(\text{“CO2.fld”}, 0) = 44.0098 \ \frac{gm}{mol}$

$\text{REFPROP}(\text{“WATER”}, \text{“”}, \text{“M”}, 0, 0) = 18.015 \ \frac{gm}{mol}$

Material Full Name

This function returns a string containing the full name of the material specified by the "**fluid string**". This can provide useful verification that the correct fluid string is being used.

Function Definition

$\text{rp\_getname}(fl, comp)$

NOTE: The additional parameter, ***comp****, is a component number for a mixture. Since mixtures are not yet implemented, this number should be 1 or 0.*

$hOut = \text{“FULLNAME”}$

High-Level call through rp_REFPROPx.

Example

$\text{rp\_getname}(\text{“D2.fld”}, 1) = \text{“Deuterium”}$

$\text{rp\_getname}(\text{“R12.fld”}, 1) = \text{“Dichlorodifluoromethane”}$

$\text{REFPROP}(\text{“D2”}, \text{“”}, \text{“LONGNAME”}, 0, 0) = \text{“Deuterium”}$

Material Chemical Abstracts Serial Number, CASN

This function returns a string containing the Chemical Abstracts Serial Number, ***CASN***, of the material specified by the "**fluid string**". This can provide useful verification that the correct fluid string is being used.

Function Definition

$\text{rp\_getcasn}(fl, comp)$

NOTE: The additional parameter, ***comp****, is a component number for a mixture. Since mixtures are not yet implemented, this number should be 1 or 0.*

$hOut = \text{“CAS\#”}$

High-Level call through rp_REFPROPx.

Example

$\text{rp\_getcasn}(\text{“D2.fld”}, 1) = \text{“7782-39-0”}$

$\text{rp\_getcasn}(\text{“R12.fld”}, 1) = \text{“75-71-8”}$

$\text{REFPROP}(\text{“D2”}, \text{“”}, \text{“CAS\#”}, 0, 0) = \text{“7782-39-0”}$

Extrapolation Flag

The ***NIST RefProp*** database allows all functions to be extrapolated out to 150% of the maximum temperature and 200% of the maximum pressure values specified in the material files. This behavior has been found to be fairly accurate, since the curves are well behaved, if not linear, beyond the experimental temperature/ pressure limits in the super-critical region.

The rp_extrap function sets a flag that allows extrapolation out to *1.5 x* T_{max} and *2.0 x* P_{max} , and returns a verification string as to the state of the flag. Once set, this flag is used for all calculations below and to the right of the call, or until another call is made. ***Extrapolation is OFF by default***.

Function Definition

$\text{rp\_extrap}(flag)$

flag: 0 = no extrapolation past T_{max} *(default)*
1 = extrapolate to $1.5 \cdot T_{max}$.
extrapolate to $2.0 \cdot P_{max}$

Example

$\text{rp\_extrap}(0) = \text{“Extrapolation Disabled”}$

$\text{rp\_extrap}(1) = \text{“Extrapolation Enabled”}$

Error Message (High-Level API) - rp_ERRMSG(*ierr*)

While the Mathcad wrapper tries to intercept and handle most errors, the error message table is static and cannot be modified to reflect the actual errors coming from REFPROP. It is inefficient and cumbersome (aka "slow") to try and capture all of the error possibly coming from REFPROP. Therefore, the function rp_ERRMSG(*ierr*) will return REFPROP's error string for a given error code (*ierr*).

However, Mathcad users will rarely know the REFPROP error code returned, since functions that fail don't return anything but the message from the add-in's static error message table. Passing $ierr = 0$ to the rp_ERRMSG function lets the user retrieve the error message returned from the ***last*** REPROP function call (i.e. the failed function evaluated just prior to this call). Even if an error is trapped by the Mathcad REFPROP add-in, the actual REFPROP error message can provide more useful details on the error.

The full error message is returned as a Mathcad string. The function returns "No Error" if the last internal error code generated was $ierr = 0$.

Only available if REFPROP 10 or greater is in use. The REFPROP unit wrapper function returns this error by default if a non-zero error occurs.

Examples:

Bad *hFlag* string:

rp_FLAGS("BadFlag", 1) = ?

Unknown error: Flag_string_not_valid.

rp_ERRMSG(0) = "[FLAGS error 601] Invalid input: BADFLAG"

Bad *hIn* string to REFPROP:

rp_REFPROP(" ", "T", "D", 0, 0) = ?

Unknown error: Invalid_hIn,_state-point_string.

rp_ERRMSG(0) = "[GETINPUTS error 601] Invalid input: T"

FLAGS (High-Level API) - rp_FLAGS(*hFlag*, *iFlag*)

The internal behavior of REFPROP can be modified by setting a number of FLAGS through the public function **FLAGSdll**. The the add-in function rp_FLAGS(*hFlag*, *iFlag*) exactly mirrors the behavior of **FLAGSdll**, except that some flags are not allowed to be modified in the Mathcad wrapper as it would cause unexpected behavior within the Mathcad front end. In these cases, the user is warned through a Message Box and the default value is returned.

Please see the NIST REFPROP DLL documentation for a complete list of valid *hFlag* strings.

Examples:

rp_FLAGS("Calorie", −999) = 0	Call with −999 to retrieve current value set for flag.
rp_FLAGS("Calorie", 1) = 1	Provide ("*hFlag*" , "*jFlag*") to set the FLAG to a specific value. The set value is returned.
rp_FLAGS("Calorie", −999) = 1	Double check set value
rp_FLAGS("Calorie", 0) = 0	Set FLAG value back to default.
rp_FLAGS("Calorie", 2) = 0	Invalid FLAG values (*jFlag*) ; action ignored.

One "extra" flag is provided that is not native to REFPROP's **FLAGSdll** public function. Some functions will create "pop-up" Warnings messages to warn the user that the function is returning unexpected results or possibly being used in an unexpected way. While helpful, these Warning pop-ups can be extremely difficult and annoying to clear if the function is called within a Mathcad loop, say for 1,000 iterations; generating 1,000 individual pop-ups. The "No Warnings" Flag will, when set to 1, suppress these warning pop-up messages.

rp_FLAGS("No Warnings", 1) = 1	Set "No Warnings" to 1 to hide warning message pop-ups.

Flags should generally be set at the top of a Mathcad worksheet for consistency throughout and will impact behavior of other open worksheets, since they are native to the Mathcad program (and its loaded Custom Functions) and not the individual worksheet.

Enumeration Value (High-Level API) - rp_GETENUM(*iFlag*, *hEnum*)

The function rp_GETENUM(*iFlag*, *hEnum*) returns the REFPROP enumerated value that matches the string sent from *hEnum*. It is used mainly for retrieving numerical, enumerated values sent to REFPROP and ALLPROPS, instead of strings. This function follows the NIST REFPROP DLL documentation for **GETENUMdll** exactly, but only the first two parameters are passed and the third, *iEnum*, is the value returned by the function to Mathcad. If the error code, *ierr*, is non-zero, it will flag an error and the actual error message in *herr* can be retrieved using rp_ERRMSG(0).

These enumerated values are typically only needed for calls to ALLPROPS where speed is of primary importance by avoiding string manipulation within the function.

The first parameter *iFlag* must be either:

0 Check all strings possible.
1 Check strings for property ***units*** only (e.g. SI, English, etc.).
2 Check property strings and those in #3 only.
3 Check property strings only that are not functions of *T* and *D*
(For example, the critical point, acentric factor, limits of the EOS, etc.).

Examples:

Unit strings

rp_GETENUM(0, "SI") = 2 *Look in all possible string ranges*

rp_GETENUM(1, "SI") = 2 *Look in unit string range only*

rp_GETENUM(2, "SI") = ? *Look in property string range only (iFlag = 2) - results in error*

rp_GETENUM(1, "MOLAR BASE SI") = 20

rp_GETENUM(1, "MASS BASE SI") = 21

rp_GETENUM(1, "USER2") = 12 *This is the iUnit value that the Mathcad wrapper uses by default for modified mass based SI.*

Property Strings

rp_GETENUM(0, "D") = 3 rp_GETENUM(2, "D") = 3

rp_GETENUM(0, "XMASS") = 456 rp_GETENUM(3, "XMASS") = 456

rp_GETENUM(0, "BETA") = 42 rp_GETENUM(2, "BETA") = 42

Set Fluids Path (High-Level API)

This routine is identical to the lower-case rp_setpath(*hPth*) function. It sets the path where the fluid files are located; does not change the path for the loaded REFPROP DLL.

The path (*hPth*) does not need to contain the ending "\" and it can point directly to the location where the REFPROP DLL is stored if a fluids subdirectory (with the corresponding fluid files) is located there.

Using Mathcad's Data File browser, the path to an actual fluids file (*.fld*) can be selected and the ending "\FLUIDS\name.fld" will be stripped off of the path string so that the fluids path is set to the parent directory.

Function Definition

rp_SETPATH(*hPth*)	*Location of the fluid files.*

CAUTION: This function should only be called at the beginning of a worksheet. Swapping fluids directories in the middle of a calculation can cause unexpected results.

CAUTION: A valid binary mixtures file (HMX.BNC) corresponding to the currently loaded REFPROP version and fluid files must exist in the \FLUIDS directory or unexpected and inaccurate values may result.

Chapter 8 Mixture Properties

In addition to selecting pure fluids, RefProp has very sophisticated algorithms for calculating properties of mixtures. There are several ways to specify a mixture as the requested fluid in RefProp:

- **Pseudo-Pure Fluids**
- **Predefined Mixture Files**
- **Custom Mixture Files**
- **Ad-Hoc Fluid Mixture Strings** (*provided by the Mathcad add-in*)

These methods will be discussed in detail below.

Pseudo-Pure Fluids

There are a few mixtures for which the mixture properties have been curve fit as though the mixture was a pure fluid. These are pseudo-pure fluids

Details:

- The "**fluid string**" can be set to one of the pseudo-pure fluids files to be loaded from the "fluids\" directory. These files MUST end with the extension "**.ppf**".

  e.g. to load the properties for AIR, define a variable such as:

  $fluid := \text{“AIR.ppf”}$

- A list of the pseudo-pure fluid files available as of REFPROP 9.1 are are shown in the table below[2].

Pseudo-Pure Fluids use extension ".ppf"				
Air	R404A	R407A	R410A	R507A

[2] Current REFPROP version documentation should be consulted for the complete list

Predefined Mixture Files

Mixture files are small files that identify the component pure fluids that make up the mixture and the single-phase mole fractions of each component. The file also contains a number of parameters to provide the mixture molecular mass and the limiting temperatures and pressures for the overall mixture.

For example, this is the mixture file for Air (as opposed to the .ppf file described above). This text file, **AIR.mix**, is in the [REFPROP]\Mixtures directory.

```
Air (dry)
 28.958600656 132.791424977327 3844.70114021364 11.9051814519786
 3
NITROGEN.FLD
ARGON.FLD
OXYGEN.FLD
 .7812
 .0092
 .2096
0
```

The table below contains a list of all of the predefined mixture files available as of RefProp 9.1, stored in the mixtures folder of the REFPROP installation directory. The vast majority of these mixture files are refrigerant combinations in addition to AIR and a handful of regional natural gas mixtures[3].

Pre-defined Mixtures use extension ".mix"				
R401A	R408A	R419A	R431A	R503
R401B	R409A	R420A	R432A	R504
R401C	R409B	R421A	R433A	R507A
R402A	R410A	R421B	R434A	R508A
R402B	R410B	R422A	R435A	R508B
R403A	R411A	R422B	R436A	R509A
R403B	R411B	R422C	R436B	R510A
R404A	R412A	R422D	R437A	R512A
R405A	R413A	R423A	R438A	AIR
R406A	R414A	R424A	R441A	AMARILLO
R407A	R414B	R425A	R442A	EKOFISK
R407B	R415A	R426A	R443A	GLFCOAST
R407C	R415B	R427A	R444A	HIGHCO2
R407D	R416A	R428A	R500	HIGHN2
R407E	R417A	R429A	R501	
R407F	R418A	R430A	R502	

[3] Current REFPROP version documentation should be consulted for the complete list

Details:

- The "**fluid string**" can be set to one of the pre-defined mixture files to be loaded from the "mixtures\" directory. These files MUST end with the extension "**.mix**".

e.g. to load the properties for AIR, define a variable such as:

fluid := "AIR.mix"

Custom Mixture Files

Since mixture files are just text files, new mixtures of pure fluids can be created. REFPROP will apply its mixture correlations to provide fairly accurate mixture properties.

The format of a mixture file is as follows,

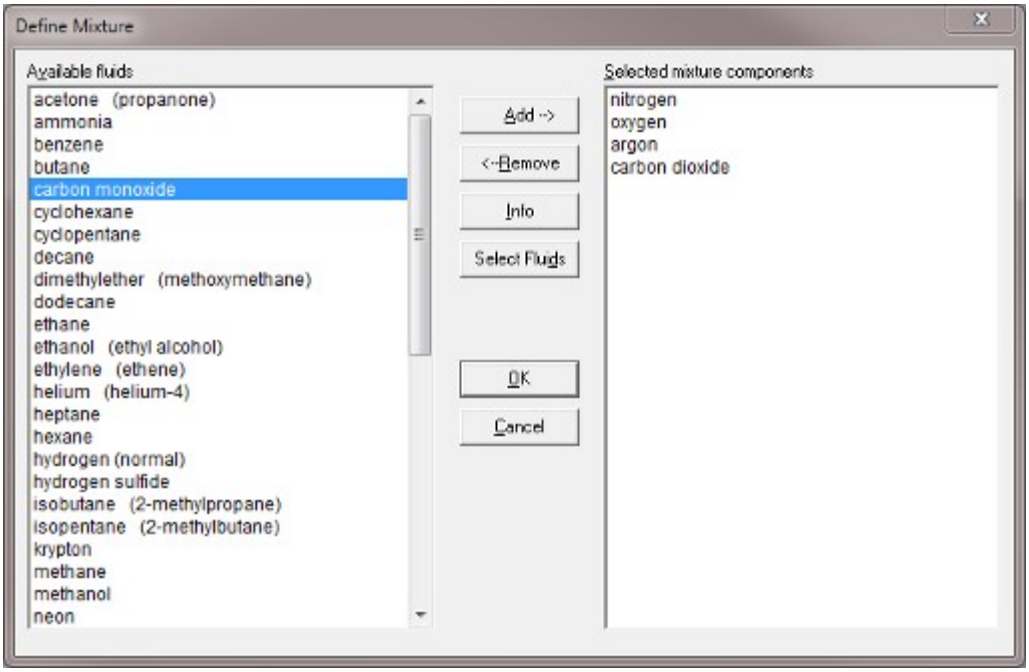
```
Mixture Name
 [molar mass] [Tcrit(K)] [Pcrit(kPa)] [Dcrit(mol/m³)]
 n                 // = number of components
fluid1.fld         // component 1
fluid2.fld         // component 2
 |
fluidn.fld         // component n
 mf1               // mole fraction 1
 mf2               // mole fraction 2
 |
 mfn               // mole fraction n
0                  // end of file delimiter
```

These files can be constructed for new mixtures so that they can be reused over and over again, without having to specify the mixture parameters repeatedly. Luckily, the Graphical User Interface (GUI) for RefProp provides a handy facility for creating and saving new mixture files[4].

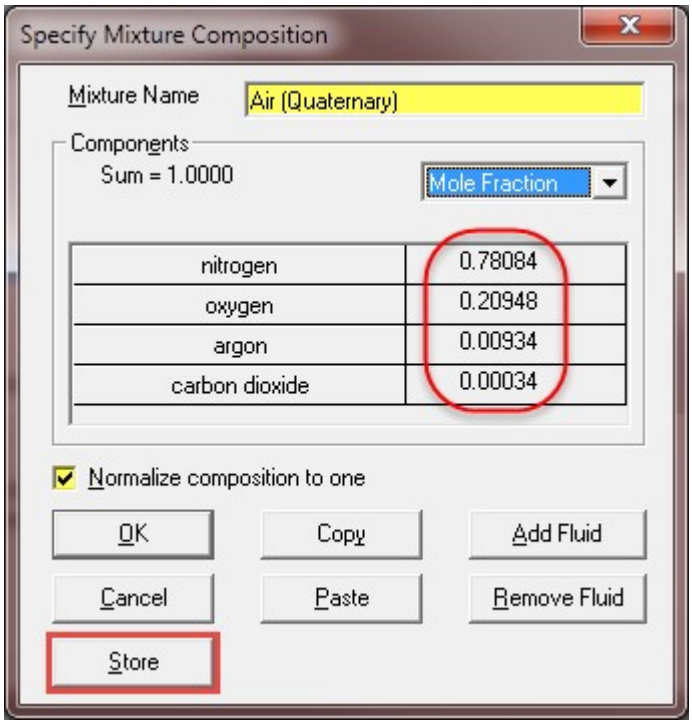
As an example, let's create an alternative mixture for air that has slightly different mole fractions and an additional carbon dioxide component than the predefined AIR.mix file.

[4] The [REFPROP]\mixtures folder should not be read-only and the user must have write access permission for this method to work.

Open the REFPROP GUI and select New Mixture from the Substance menu. In the Define Mixture panel, select the four components for air, including carbon dioxide, to form a quaternary mixture.



Select **OK** and then fill in the ***mole fractions*** for each component per the on the right. Make sure **Mole Fraction** is selected from the drop-down and now Mass Fraction. (Although mass fractions can be entered, they will be converted to mole fractions when saving the file.) All component mole fractions should sum to 1.0; however, you can select the checkbox to normalize the mole fractions to unity (1.0) and REFPROP will do just that. Change the name in the top box to a suitable name for the mixture.



Select Store to save the mixture file in the mixtures directory, giving it an appropriate file name (e.g. AIR4.MIX). This mixture file will now be available for use in REFPROP.

NOTE: On systems where users cannot write to the C: drive for security reasons, while the REFPROP (GUI) program can store the new fluid and load it for use, the REFPROP add-in for Mathcad Prime (as well as any other wrapper add-ins, like Excel) will not be able to find the new mixture file in the "REFPROP\MIXTURES\" directory.

$$\text{rp\_getname}(\text{“AIR.mix”}, 0) = \text{“Air (dry)”} \quad \Leftarrow \textit{This is REFPROP's built–in Mixture}$$

The REFPROP package, instead writes this file to to the user's “VirtualStore” directory for the application. This s a Windows security feature. However, the REFPROP add-in looks for the file in the user's “VirtualStore” if it cannot find it in the the default MIXTURES directory and sets the full path to:

"*C:\Users\username\AppData\Local\VirtualStore\Program Files (x86)\REFPROP\MIXTURES*"

The file can then be found just be entering the custom mixture file name, as seen here.

$$\text{rp\_getname}(\text{“AIR4.mix”}, 0) = \text{“Air (Quaternary - Dry)”} \quad \Leftarrow \textit{Custom mixture in VirtualStore}$$

Note also that the "*VirtualStore*" directory is in the user's ***Local*** profile on the C: drive. This means that any custom .mix files will not be available on other network machines, or to other users on the same machine, and will have to be recreated if the user's profile is erased (i.e. new machine, Windows update, etc.).

Let's use the original and new mixture files for air to calculate room temperature density and compare it to the pseudo-pure fluid value.

$$T := 22\ °C \qquad P := 1 \cdot atm$$

$$\rho_{orig} := \text{rp\_rhotp}\left(\text{“AIR.MIX”}, \frac{T}{K}, \frac{P}{MPa}\right) \cdot \frac{kg}{m^3} = 1.196079\ \frac{kg}{m^3}$$

$$\rho_{AIR4.MIX} := \text{rp\_rhotp}\left(\text{“AIR4.mix”}, \frac{T}{K}, \frac{P}{MPa}\right) \cdot \frac{kg}{m^3} = 1.196354\ \frac{kg}{m^3}$$

$$\rho_{ppf} := \text{rp\_rhotp}\left(\text{“AIR.ppf”}, \frac{T}{K}, \frac{P}{MPa}\right) \cdot \frac{kg}{m^3} = 1.196397\ \frac{kg}{m^3}$$

$$\frac{\rho_{orig} - \rho_{ppf}}{\rho_{ppf}} = -0.0265\% \qquad \frac{\rho_{AIR4.MIX} - \rho_{ppf}}{\rho_{ppf}} = -0.0036\%$$

It appears that the new quaternary mixture including carbon dioxide is closer to the pseudo-pure fluid density than the ternary mixture. Both mixture densities, however, are fairly close to the density calculated from the pseudo-pure fluid.

Ad-Hoc Mixture Strings (Legacy API)

REFPROP provides facility for embedding multiple mixture components right in the fluid string passed to the interface functions. The Mathcad Add-In Legacy API functions, takes this a step further and allow not only the component names, but also the *mole fractions* of each component to be embedded in the fluid string. The format for mixture fluids is as follows. Within the fluid string,

1. Separate each component with an "**&**" character,
2. Follow each component with its mole fraction, surrounded by "**[**" and "**]**",
3. The number of components is limited to ***20***, although the more components makes the flash calculations harder and they may fail in the REFPROP solver,
4. Zero mole fraction components will be parsed out,
5. Total of mole fractions must sum exactly to one (1.0).

For example, a binary representation for Air could be input with the following fluid string.

$$Air_{binary} := \text{“Oxygen[0.21]\&Nitrogen[0.79]”}$$

$$\rho_{air2} := \text{rp\_rhotp}\left(Air_{binary}, \frac{T}{K}, \frac{P}{MPa}\right) \cdot \frac{kg}{m^3} = 1.191606 \ \frac{kg}{m^3}$$

$$\rho_{ppf} = 1.196397 \ \frac{kg}{m^3} \qquad \frac{\rho_{ppf} - \rho_{air2}}{\rho_{ppf}} = 0.4\%$$

This binary representation for air carries a bit more error than the more detailed mixtures shown previously. The mixture string, however, can hold up to 20 components, allowing the mixture representation to be as accurate as needed, including many of the trace components of air. Failure to adhere to the mixture string format will result in a Mathcad error on the function call.

To facilitate this process, a utility function, $\text{MixString}(M)$ is provided in the ***Refprop_units*** reference worksheet that simplifies assembling these mixture strings and making sure that the mole fractions all sum to unity.

To use this utility user function, first include the reference worksheet:

Include << .\RefProp_units.mcdx

Set up the representation for the mixture in a matrix, where the first column is the pure component file string and the second column is the corresponding mole fraction of each component. Check that the second column sums to 1.0.

$$M2 := \begin{bmatrix} \text{“Oxygen”} & 0.21 \\ \text{“Nitrogen”} & 0.79 \end{bmatrix} \qquad \sum M2^{\langle 1 \rangle} = 1.000000000000000$$

$$M3 := \begin{bmatrix} \text{“Oxygen”} & 0.2096 \\ \text{“Nitrogen”} & 0.7812 \\ \text{“Argon”} & 0.0092 \end{bmatrix} \qquad \sum M3^{\langle 1 \rangle} = 1.000000000000000$$

$$M4 := \begin{bmatrix} \text{“Oxygen”} & 0.20948 \\ \text{“Nitrogen”} & 0.78084 \\ \text{“Argon”} & 0.00934 \\ \text{“CO2”} & 0.00034 \end{bmatrix} \qquad \sum M4^{\langle 1 \rangle} = 1.000000000000000$$

Now the mixture strings can be assembled using the MixString utility function.

$Air2 := \text{MixString}(M2) = \text{“Oxygen[0.21]\&Nitrogen[0.79]”}$

$Air3 := \text{MixString}(M3) = \text{“Oxygen[0.2096]\&Nitrogen[0.7812]\&Argon[0.0092]”}$

$Air4 := \text{MixString}(M4) = \text{“Oxygen[0.20948]\&Nitrogen[0.78084]\&Argon[0.00934]\&CO2[0.00034]”}$

The property functions, $\rho_{tp}(\text{“fluid”}, T, P)$, from the unit reference worksheet can then be called using this utility function directly.

$$\rho_{air2} := \rho_{tp}(\text{MixString}(M2), T, P) = 1.191606 \ \frac{kg}{m^3} \qquad \frac{\rho_{ppf} - \rho_{air2}}{\rho_{ppf}} = 0.4\%$$

$$\rho_{air3} := \rho_{tp}(\text{MixString}(M3), T, P) = 1.196079 \ \frac{kg}{m^3} \qquad \frac{\rho_{ppf} - \rho_{air3}}{\rho_{ppf}} = 0.027\%$$

$$\rho_{air4} := \rho_{tp}(\text{MixString}(M4), T, P) = 1.196354 \ \frac{kg}{m^3} \qquad \frac{\rho_{ppf} - \rho_{air4}}{\rho_{ppf}} = 0.004\%$$

The tertiary ad-hoc mixture gives exactly the same result as the predefined mixture file ($AIR.MIX$), since the ad-hoc string contains exactly the same components and mole fractions. The quaternary mixture results in a more accurate density calculation, compared to the pseudo-pure fluid ($AIR.PPF$) density, and exactly the same result, $\rho_{AIR4.MIX} = 1.196354 \ kg \cdot m^{-3}$, as was calculated using the custom air mixture file ($AIR4.MIX$) above.

Ad-Hoc Mixture Strings (High-Level API)

The REFPROP 10 High-Level API provides its own facility for specifying mixture compositions right in the fluid string passed to the function calls and it varies slightly from the Legacy API formatting shown above. The format for High-Level API mixture fluids is as follows.

Mixture Component Lists can be passed in a single string separated by semicolons or asterisks. This method does not set the ***molar*** composition, which must be subsequently performed with the custom function **rp_setx**.

$hfld_1 := \text{“METHANE;ETHANE;PROPANE;BUTANE;ISOBUTANE”}$

$hfld_2 := \text{“METHANE*ETHANE*PROPANE*BUTANE*ISOBUTANE”}$

$hfld_3 := \text{“Nitrogen; Oxygen; Argon”}$

Mixture Compositions can be passed in a single string, separating the components and their interleaved ***molar*** compositions with semicolons or asterisks. The following are valid formats for mixture composition strings.

$hfld_4 := \text{“R134a;0.3; R1234yf;0.3; R1234ze(Z);0.4”}$

$hfld_5 := \text{“CO2;0.2 * isobutane;0.3 * propadiene;0.5”}$

Alternatively, the compositions can be passed after the component list, separating the components from the ***molar*** compositions with a pipe "|" or a tilde "~" character. The following are valid formats for mixture composition strings using this method.

$hfld_6 := \text{“Nitrogen;Oxygen;Argon|0.4;0.3;0.3”}$

$hfld_7 := \text{“Nitrogen; Oxygen; Argon ~ 0.4; 0.3; 0.3”}$

Note that the spaces separating components and compositions in the string are ignored. A formatting function, $\text{MixString10}(M)$, provided in the ***Refprop_units*** reference worksheet that simplifies assembling these mixture strings using the High-Level formatting and making sure that the mole fractions all sum to unity. To demonstrate, the previous composition matrices are used.

$Air2 := \text{MixString10}(M2) = \text{“Oxygen;0.21;Nitrogen;0.79”}$

$Air3 := \text{MixString10}(M3) = \text{“Oxygen;0.2096;Nitrogen;0.7812;Argon;0.0092”}$

$Air4 := \text{MixString10}(M4) = \text{“Oxygen;0.20948;Nitrogen;0.78084;Argon;0.00934;CO2;0.00034”}$

Any mixture strings assembled in this way can be passed as the first parameter to the rp_REFPROP custom functions.

Mixture Limitations

When using REFPROP's mixing rules, REFPROP must contain binary interaction parameters for each combinatorial pair in the mixture. If these binary interaction parameters do not exist, a pop-up warning will result that parameters for a similar mixture are being used.

Additionally, the more components included in a mixture, the harder the property equations will be to converge. Simple binary, tertiary, and quaternary mixtures behave fairly well. Complex mixtures with many components (maximum of 20) can become much harder to converge, especially with component that have widely varying states at the same state point (i.e. air and water at atmospheric conditions) or are immiscible.

Mixture Behavior

For pure fluids, the dew point and bubble point occur at the same temperature and pressure, since the state point temperature and pressure remain constant through the two-phase region as the fluid vaporizes and condenses. This means that on a Pressure-Temperature (P-T) plot, there is one single saturation curve; that is, the saturated liquid curve overlaps the saturated vapor curve exactly. Here is the saturation curve for pure water.

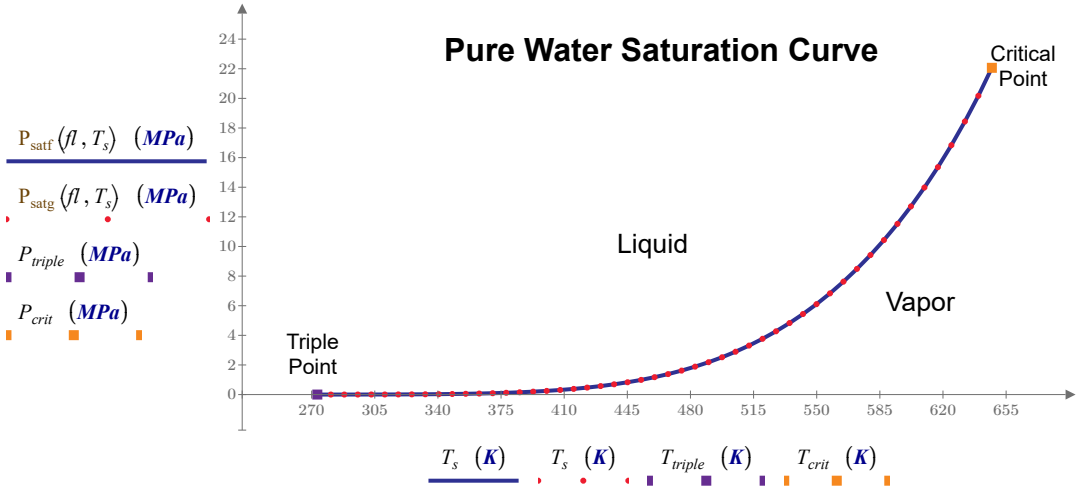
$fl := \text{"Water"}$

$T_{triple} := \mathrm{T_t}(fl, 0)$ ⇐ *This is T_{min} on the saturated liquid curve*

$P_{triple} := \mathrm{P_{satf}}(fl, T_{triple})$ ⇐ *This is P_{min} on the saturated liquid curve*

$T_{crit} := \mathrm{T_c}(fl, 0)$ $\quad P_{crit} := \mathrm{P_c}(fl, 0)$

$N := 50$ $\quad \Delta T := (T_{crit} - T_{triple}) \div N$ $\quad T_s := T_{triple}, T_{triple} + \Delta T .. T_{crit}$



For multi-component mixtures, however, this is not the case and the saturation temperature/pressure depends on the molar composition, which is not the same in the vapor and liquid equilibrium states. This can be demonstrated most dramatically for the Amarillo Gas pre-defined mixture.

$fl := \text{“Amarillo.mix”}$

$T_{triple} := T_t(fl, 0)$ ⇐ *This is T_{min} on the saturated liquid curve*

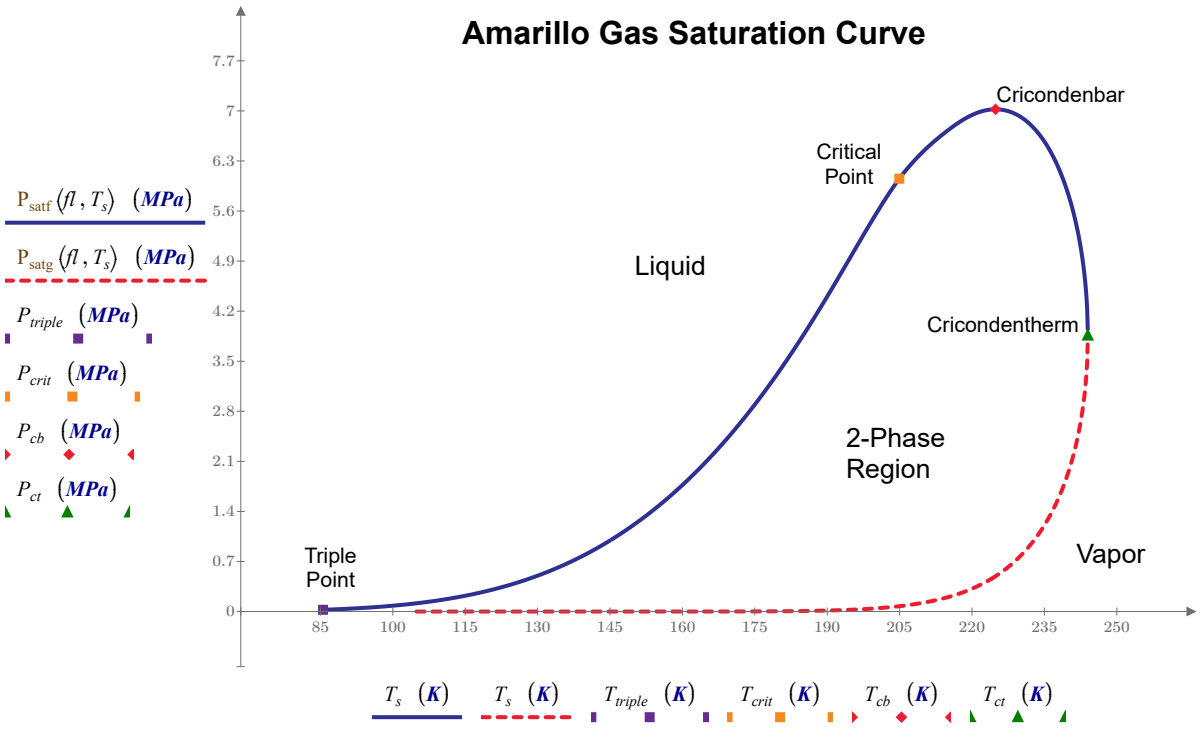
$P_{triple} := P_{satf}(fl, T_{triple})$ ⇐ *This is P_{min} on the saturated liquid curve*

$T_{crit} := T_c(fl, 0)$ $P_{crit} := P_c(fl, 0)$ ⇐ *Critical point on the saturated liquid curve*

$T_{ct} := T_{maxT}(fl)$ $P_{ct} := P_{maxT}(fl)$ ⇐ *Point of maximum temperature on the saturation curve (**Cricondentherm**)*

$T_{cb} := T_{maxP}(fl)$ $P_{cb} := P_{maxP}(fl)$ ⇐ *Point of maximum pressure on the saturation curve (**Cricondenbar**)*

$N := 1000$ $\Delta T := (T_{crit} - T_{triple}) \div N$ $T_s := T_{triple}, T_{triple} + \Delta T .. T_{ct}$



Of note is the fact that the saturated liquid curve (bubble point curve) and the saturated vapor curve (dew point curve) do not overlap. The compositions along these curves are also different, which is what allows distillation processes to separate out components from mixtures by separating the liquid and vapor phases mechanically, and then condensing the vapor phase containing higher concentrations of the more volatile components.

Secondly, the critical point is usually not at the maximum pressure on saturation curves (the ***Cricondenbar***) or the maximum temperature on the saturation curves (the ***Cricondentherm***). Not that the critical point can occur to the left (lower temperature) of the Cricondenbar, as it does for Amarillo Gas in the plot above, or in between the Cricondenbar and Cricondentherm temperatures, depending on the mixture components and composition.

In addition to the functions $rp\_psatt(fl,t)$ and $rp\_tsatp(fl,p)$, for pure fluids, two additional saturation functions are provided for mixtures (one of which was used to generate the plots above).

Raw Functions	Unit Wrapper Functions	Description
$\mathrm{rp\_tsatpf}(fl,P)$	$T_{satf}(fl,p)$	*Saturated liquid temperature at p*
$\mathrm{rp\_tsatpg}(fl,P)$	$T_{satg}(fl,p)$	*Saturated vapor temperature at p*
$\mathrm{rp\_psattf}(fl,T)$	$P_{satf}(fl,t)$	*Saturated liquid pressure at t*
$\mathrm{rp\_psattg}(fl,T)$	$P_{satg}(fl,t)$	*Saturated vapor pressure at t*

These same values can be retrieved using the high-level **REFPROP** functions with the following input parameters.

hIn String	*hOut* String	Description
"PQ" ($Q=0$)	"T" or "TLIQ"	*Saturated liquid temperature at p*
"PQ" ($Q=1$)	"T" or "TVAP"	*Saturated vapor temperature at p*
"TQ" ($Q=0$)	"P" or "PLIQ"	*Saturated liquid pressure at t*
"TQ" ($Q=1$)	"P" or "PVAP"	*Saturated vapor pressure at t*

The Cricondentherm and Cricondenbar values can also be retrieved.

Raw Functions	Unit Wrapper Functions	Description
$rp\_maxX(fl,\text{“T”})_0$	$T_{maxT}(fl)$	*Cricondentherm Temperature* [*K*]
$rp\_maxX(fl,\text{“T”})_1$	$P_{maxT}(fl)$	*Cricondentherm Pressure* [*MPa*]
$rp\_maxX(fl,\text{“P”})_0$	$T_{maxP}(fl)$	*Cricondenbar Temperature* [*K*]
$rp\_maxX(fl,\text{“P”})_1$	$P_{maxP}(fl)$	*Cricondenbar Pressure* [*MPa*]

Of course these values can also be retrieved using the high-level REFPROP functions using *hOut* values of "TMAXT", "PMAXT", "TMAXP", and "PMAXP", which are independent of the *hIn* and a,b values passed.

Appendix A - Legacy API Fluid Property Functions

REFPROP is based on the most accurate pure fluid and mixture models currently available. It implements three models for the thermodynamic properties of pure fluids: equations of state explicit in Helmholtz energy, the modified Benedict-Webb-Rubin equation of state, and an extended corresponding states (ECS) model.

Mixture calculations employ a model that applies mixing rules to the Helmholtz energy of the mixture components; it uses a departure function to account for the departure from ideal mixing.

Thermodynamic Properties

The underlying NIST routines are provided to calculate thermodynamic and transport properties at a given (T,ρ,x) state. The functions below, are provided as functions of Temperature (T) and Pressure (P). This requires an iterative calculation of Density (ρ) followed by a direct calculation of the requested property in terms of Temperature and Density.

Transport Properties

Viscosity and thermal conductivity are modeled with either fluid-specific correlations, an ECS (extended corresponding states) method, or in some cases the friction theory method. Surface tension is calculated from curve fit correlations specific to each fluid material.

See also

Some of the examples in this appendix require the units reference from Appendix C.

Include << .\RefProp_units.mcdx

A.1 Density, ρ

The fluid density describes the mass of a unit volume of fluid at a given state and is expressed in units of $kg \cdot m^{-3}$. Since the Helmholtz equation is a function of temperature and density, calculating density in terms of temperature and pressure requires an iterative solution.

solved equation: $$p = \rho^2 \cdot \left(\frac{\mathrm{d}}{\mathrm{d}\rho} f \right)_T$$

Specific volume can be evaluated as the inverse of the density.

This iterative density calculation provides the basis for all other thermodynamic property calculations, which can be evaluated directly in terms of Temperature and Density.

Functions that return Density (rho)

rp_rhotp(*fluid*, *t*, *p*)	Obtains density of ***superheated vapor*** or ***compressed liquid*** in kg/m³ as a function of temperature (t) in K and Pressure (p) in MPa.
rp_rhoft(*fluid*, *t*)	Obtains density of ***saturated liquid*** in kg/m³ as a function of temperature (t) in K.
rp_rhofp(*fluid*, *p*)	Obtains density of ***saturated liquid*** in kg/m³ as a function of pressure (p) in MPa.
rp_rhogt(*fluid*, *t*)	Obtains density of ***saturated vapor*** in kg/m³ as a function of temperature (t) in K.
rp_rhogp(*fluid*, *p*)	Obtains density of ***saturated vapor*** in m³/kg as a function of pressure (p) in MPa.

NOTE : The first parameter in all of these functions is a "**fluid string**" variable that identifies the fluid material to be used for the property calculations.

Application of Units

Mathcad user functions are shown here to calculate specific volume ($1/\rho$) with the application of units. These functions are defined in the include file, *Refprop_units.mcdx*.

$$v_{tp}(fluid,t,p) := \left(\text{rp\_rhotp}\left(fluid,\frac{t}{K},\frac{p}{MPa}\right)\cdot\frac{kg}{m^3}\right)^{-1}$$

$$v_{ft}(fluid,t) := \left(\text{rp\_rhoft}\left(fluid,\frac{t}{K}\right)\cdot\frac{kg}{m^3}\right)^{-1} \qquad v_{gt}(fluid,t) := \left(\text{rp\_rhogt}\left(fluid,\frac{t}{K}\right)\cdot\frac{kg}{m^3}\right)^{-1}$$

$$v_{fp}(fluid,p) := \left(\text{rp\_rhofp}\left(fluid,\frac{p}{MPa}\right)\cdot\frac{kg}{m^3}\right)^{-1} \qquad v_{gp}(fluid,p) := \left(\text{rp\_rhogp}\left(fluid,\frac{p}{MPa}\right)\cdot\frac{kg}{m^3}\right)^{-1}$$

Example

$fl := \text{"WATER.fld"}$

$T_c := \text{rp\_tcrit}(fl,0)\ K = 705.103\ °F$

$T_t := \text{rp\_ttrip}(fl,0)\ K = 32.018\ °F$

Temperatures

$n := 200$

$\Delta T := (T_c - T_t)\cdot\frac{1}{n}$

$t1 := T_t, T_t + \Delta T .. 2\cdot T_c$

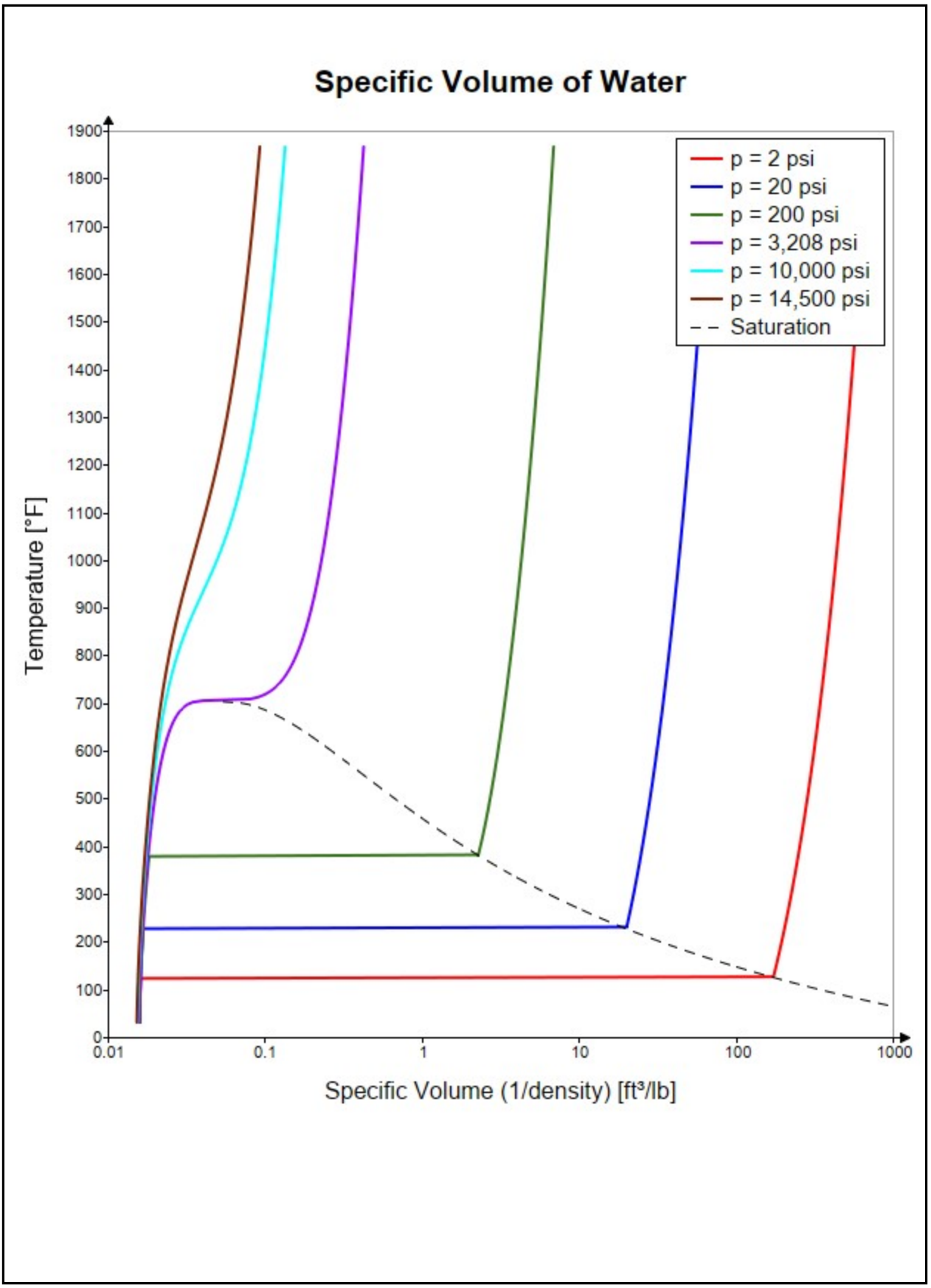
Pressures

$p1 := 2\cdot psi$ $\quad p4 := 3208\cdot psi$

$p2 := 20\cdot psi$ $\quad p5 := 10000\cdot psi$

$p3 := 200\cdot psi$ $\quad p6 := 14500\cdot psi$

The plot of specific volume vs. temperature at the specified pressures on the following page provides a continuity check for these functions.


Specific Volume of Water
Temperature [°F]
Specific Volume (1/density) [ft³/lb]
p = 2 psi
p = 20 psi
p = 200 psi
p = 3,208 psi
p = 10,000 psi
p = 14,500 psi
Saturation
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
1700
1800
1900
0.01
0.1
1
10
100
1000

A.2 Specific Internal Energy, u

Internal energy represents the total energy stored in a fluid. It is the sum of the molecular kinetic energy and the molecular kinetic energy within the fluid and has the units of kJ/kg. The internal energy of a substance does not include any energy that it may posses as a result of its position or movement as a whole. Rather it refers to the energy of the molecules making up the substance.

Internal energy is calculated from the Helmholtz free energy surface, $f(\rho, T)$, as follows..

Internal Energy: $$u = f - T \cdot \left(\frac{\mathrm{d}}{\mathrm{d}T} f \right)_{\rho}$$

The total internal energy of a fluid can not be measured. However, changes in internal energy calculated from a reference state can be evaluated. For water, this reference state is saturated liquid at 0.01°C (273.16 K). At the reference state, internal energy, u, is set to zero (as demonstrated in the example below).

Evaluate at reference state: $T_{ref} := 273.16 \cdot K$ $fluid := \text{“WATER.fld”}$

$$\text{rp\_uft}\left(fluid, \frac{T_{ref}}{K}\right) \cdot \frac{kJ}{kg} = 0.00000000 \ \frac{kJ}{kg}$$

Functions that return Internal Energy

$rp\_utp(fluid, t, p)$	Obtains internal energy of ***superheated vapor*** or ***compressed liquid*** in kJ/kg as a function of temperature (t) in K and Pressure (p) in MPa.
$rp\_uft(fluid, t)$	Obtains internal energy of ***saturated liquid*** in kJ/kg as a function of temperature (t) in K.
$rp\_ufp(fluid, p)$	Obtains internal energy of ***saturated liquid*** in kJ/kg as a function of pressure (p) in MPa.

$rp\_ugt(fluid, t)$	Obtains internal energy of ***saturated vapor*** in kJ/kg as a function of temperature (t) in K.
$rp\_ufp(fluid, p)$	Obtains internal energy of ***saturated liquid*** in kJ/kg as a function of pressure (p) in MPa.

NOTE : The first parameter in all of these functions is a "**fluid string**" variable that identifies the fluid material to be used for the property calculations.

Application of Units

Mathcad user functions are shown here to calculate internal energy (u) with the application of units. These functions are defined in the include file, *Refprop_units.mcdx*.

$$u_{tp}(fluid, t, p) := \text{rp\_utp}\left(fluid, \frac{t}{K}, \frac{p}{MPa}\right) \cdot \frac{kJ}{kg}$$

$$u_{ft}(fluid, t) := \text{rp\_uft}\left(fluid, \frac{t}{K}\right) \cdot \frac{kJ}{kg} \qquad u_{gt}(fluid, t) := \text{rp\_ugt}\left(fluid, \frac{t}{K}\right) \cdot \frac{kJ}{kg}$$

$$u_{fp}(fluid, p) := \text{rp\_ufp}\left(fluid, \frac{p}{MPa}\right) \cdot \frac{kJ}{kg} \qquad u_{gp}(fluid, p) := \text{rp\_ugp}\left(fluid, \frac{p}{MPa}\right) \cdot \frac{kJ}{kg}$$

Example

$fl := \text{“WATER.fld”}$

Temperatures

$n := 200$

$\Delta T := (T_c - T_t) \cdot \frac{1}{n}$

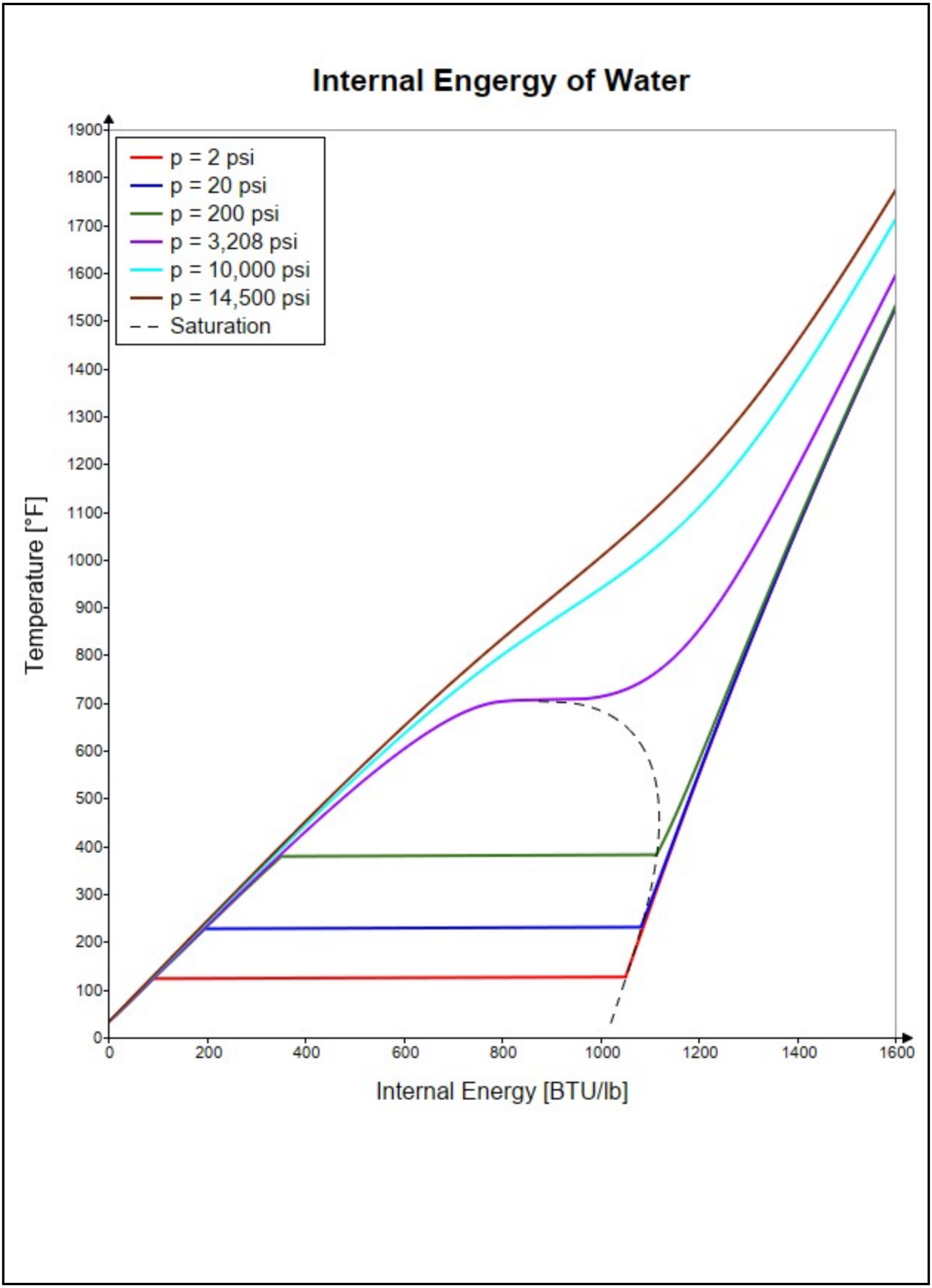
$t1 := T_t, T_t + \Delta T .. 2 \cdot T_c$

Pressures

$p1 := 2 \cdot psi$ $\quad p4 := 3208 \cdot psi$

$p2 := 20 \cdot psi$ $\quad p5 := 10000 \cdot psi$

$p3 := 200 \cdot psi$ $\quad p6 := 14500 \cdot psi$


Internal Engergy of Water
p = 2 psi
p = 20 psi
p = 200 psi
p = 3,208 psi
p = 10,000 psi
p = 14,500 psi
Saturation
Temperature [°F]
Internal Energy [BTU/lb]
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
1700
1800
1900
0
200
400
600
800
1000
1200
1400
1600

A.3 Specific Enthalpy, h

Enthalpy is defined (out of convenience) as the sum of the internal energy, u and the pressure-volume product, $P \cdot v$ and has the units of kJ/kg:

$$h = u + P \cdot v$$

Internally, enthalpy is calculated from the Helmholtz free energy surface, $f(\rho, T)$, in the following manner:

specific enthalpy: $$h = f - T \cdot \left(\frac{\mathrm{d}}{\mathrm{d}T} f \right)_{\rho} + \rho \cdot \left(\frac{\mathrm{d}}{\mathrm{d}\rho} f \right)_{T}$$

Keep in mind that enthalpy is a property, and its use in calculating internal energy at the same state is not dependent on any processes that may be occurring. Values of enthalpy are relative to an arbitrarily chosen reference state. For water, this reference state is saturated liquid at 0.01°C (273.16 K). At the reference state, internal energy, u, is set to zero, leaving $h = P \cdot v$
(as demonstrated in the example below).

$$fluid := \text{“WATER.fld”}$$

Evaluate at reference state: $$T_{ref} := 273.16 \cdot \boldsymbol{K}$$

$$P_{sat}(fluid, T) := \text{rp\_psatt}\left(fluid, T \cdot \boldsymbol{K}^{-1}\right) \cdot \boldsymbol{MPa}$$

$$h_f(fluid, t) := \text{rp\_hft}\left(fluid, \frac{t}{\boldsymbol{K}}\right) \cdot \frac{kJ}{\boldsymbol{kg}} \qquad v_f(fluid, t) := \left(\text{rp\_rhoft}\left(fluid, \frac{t}{\boldsymbol{K}}\right) \cdot \frac{\boldsymbol{kg}}{\boldsymbol{m}^3}\right)^{-1}$$

$$h_f(fluid, T_{ref}) - P_{sat}(fluid, T_{ref}) \cdot v_f(fluid, T_{ref}) = 0.00000000 \ \frac{kJ}{kg}$$

Functions that return Enthalpy

$rp\_htp(fluid, t, p)$	Obtains enthalpy of ***superheated vapor*** or ***compressed liquid*** in kJ/kg as a function of temperature (t) in K and Pressure (p) in MPa.
$rp\_hft(fluid, t)$	Obtains enthalpy of ***saturated liquid*** in kJ/kg as a function of temperature (t) in K.
$rp\_hfp(fluid, p)$	Obtains enthalpy of ***saturated liquid*** in kJ/kg as a function of pressure (p) in MPa.
$rp\_hgt(fluid, t)$	Obtains enthalpy of ***saturated vapor*** in kJ/kg as a function of temperature (t) in K.
$rp\_hgp(fluid, p)$	Obtains enthalpy of ***saturated vapor*** in kJ/kg as a function of pressure (p) in MPa.

NOTE : The first parameter in all of these functions is a **"fluid string"** variable that identifies the fluid material to be used for the property calculations.

Application of Units

Mathcad user functions are shown here to calculate enthalpy (h) with the application of units. These functions are defined in the include file, *Refprop_units.mcdx*.

$$h_{tp}(fluid, t, p) := \text{rp\_htp}\left(fluid, \frac{t}{K}, \frac{p}{MPa}\right) \cdot \frac{kJ}{kg}$$

$$h_{ft}(fluid, t) := \text{rp\_hft}\left(fluid, \frac{t}{K}\right) \cdot \frac{kJ}{kg} \qquad h_{gt}(fluid, t) := \text{rp\_hgt}\left(fluid, \frac{t}{K}\right) \cdot \frac{kJ}{kg}$$

$$h_{fp}(fluid, p) := \text{rp\_hfp}\left(fluid, \frac{p}{MPa}\right) \cdot \frac{kJ}{kg} \qquad h_{gp}(fluid, p) := \text{rp\_hgp}\left(fluid, \frac{p}{MPa}\right) \cdot \frac{kJ}{kg}$$

Example

$fluid$ = "WATER.fld"

Temperatures

$n := 200$

$\Delta T := (T_c - T_t) \cdot \frac{1}{n}$

$t1 := T_t, T_t + \Delta T .. 2 \cdot T_c$

Pressures

$p1 := 2 \cdot psi$

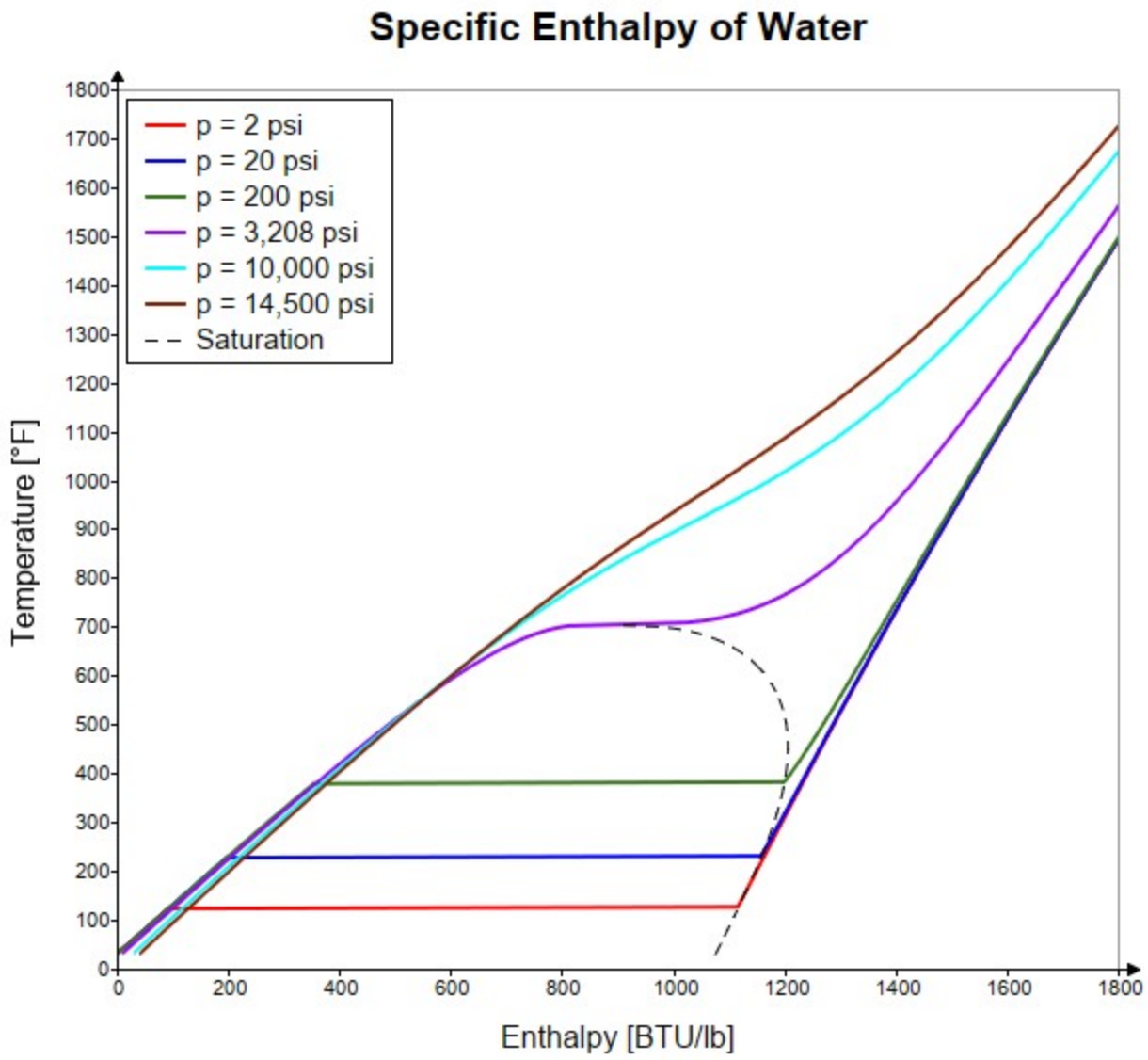
$p2 := 20 \cdot psi$

$p3 := 200 \cdot psi$

$p4 := 3200 \cdot psi$

$p5 := 10000 \cdot psi$

$p6 := 14500 \cdot psi$



A.4 Specific Entropy, *s*

For a reversible cyclical process where temperature varies during heat absorption and rejection,

$$\int \frac{1}{T} \mathrm{d}Q = 0 \qquad \text{is true.}$$

Since this is true, it follows that the above integral is path independent. This integral has been defined as the entropy change, s, and has units of kJ/kg-K. The entropy of water is only dependent on its state. Like internal energy, entropy, ***s***, is defined as zero for saturated liquid at 0.01 °C (273.16 K).

The specific entropy of a fluid is determined from the Helmholtz free energy surface, $f(\rho, T)$, by the following equation.

Specific Entropy:
$$s = -\left(\frac{\mathrm{d}}{\mathrm{d}T} f\right)_{\rho}$$

Functions that return Specific Entropy

$rp\_stp(fluid, t, p)$	Obtains entropy of ***superheated vapor*** or ***compressed liquid*** in kJ/kg-K as a function of temperature (t) in K and Pressure (p) in MPa. ***fluid*** is the fluid string for the requested material.
$rp\_sft(fluid, t)$	Obtains entropy of ***saturated liquid*** in kJ/kg-K as a function of temperature (t) in K.
$rp\_sfp(fluid, p)$	Obtains entropy of ***saturated liquid*** in kJ/kg-K as a function of pressure (p) in MPa.
$rp\_sgt(fluid, t)$	Obtains entropy of ***saturated vapor*** in kJ/kg-K as a function of temperature (t) in K.
$rp\_sgt(fluid, p)$	Obtains entropy of ***saturated vapor*** in kJ/kg-K as a function of pressure (p) in MPa.

NOTE : The first parameter in all of these functions is a **"fluid string"** variable that identifies the fluid material to be used for the property calculations.

Application of Units

Mathcad user functions are shown here to calculate entropy (s) with the application of units. These functions are defined in the include file, *Refprop_units.mcdx* .

$$s_{tp}(fluid,t,p) := \text{rp\_stp}\left(fluid,\frac{t}{K},\frac{p}{MPa}\right)\cdot\frac{kJ}{kg\cdot K} \qquad kJ \equiv 1000\cdot J$$

$$s_{fp}(fluid,p) := \text{rp\_sfp}\left(fluid,\frac{p}{MPa}\right)\cdot\frac{kJ}{kg\cdot K} \qquad s_{ft}(fluid,t) := \text{rp\_sfp}\left(fluid,\frac{t}{K}\right)\cdot\frac{kJ}{kg\cdot K}$$

$$s_{gp}(fluid,p) := \text{rp\_sgp}\left(fluid,\frac{p}{MPa}\right)\cdot\frac{kJ}{kg\cdot K} \qquad s_{gt}(fluid,t) := \text{rp\_sgt}\left(fluid,\frac{t}{K}\right)\cdot\frac{kJ}{kg\cdot K}$$

Example

$fluid :=$ "WATER.fld"

Temperatures

$n := 200$

$\Delta T := (T_c - T_t)\cdot\frac{1}{n}$

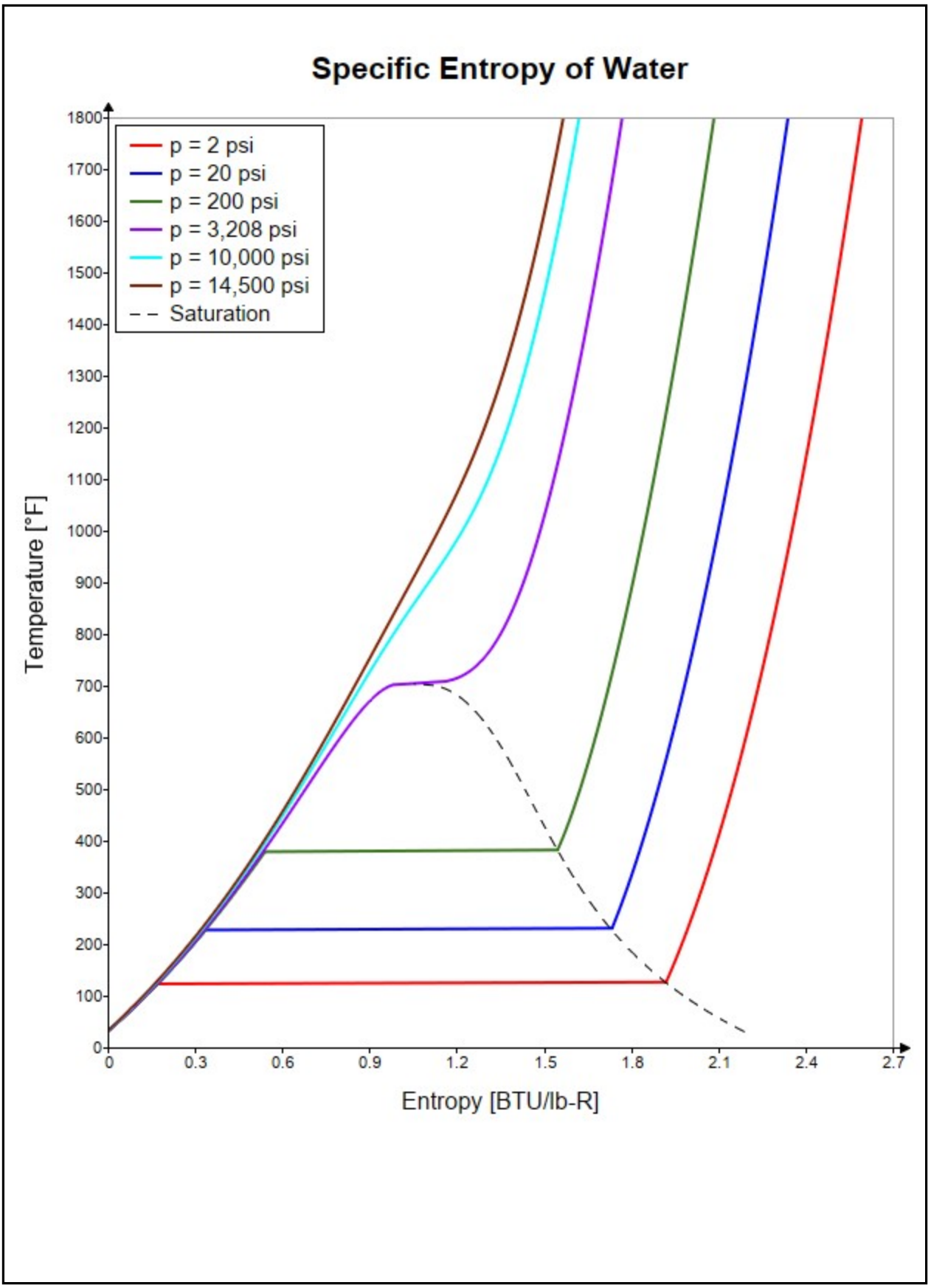
$t1 := T_t, T_t + \Delta T .. 2\cdot T_c$

Pressures

$p1 := 2\cdot psi$ $\quad p4 := 3200\cdot psi$

$p2 := 20\cdot psi$ $\quad p5 := 10000\cdot psi$

$p3 := 200\cdot psi$ $\quad p6 := 14500\cdot psi$


Specific Entropy of Water
p = 2 psi
p = 20 psi
p = 200 psi
p = 3,208 psi
p = 10,000 psi
p = 14,500 psi
Saturation
Temperature [°F]
Entropy [BTU/lb-R]
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
1700
1800
0
0.3
0.6
0.9
1.2
1.5
1.8
2.1
2.4
2.7

A.5 Isobaric Specific Heat, Cp

Isobaric Specific Heat is defined as the rate of change of enthalpy of a unit mass of material with respect to temperature at a given constant pressure (isobaric):

$$c_p = \left(\frac{\mathrm{d}}{\mathrm{d}T}h\right)_p = -T \cdot \left(\frac{\mathrm{d}^2}{\mathrm{d}T^2}f\right)_{V,\, dA/dV_T}$$

(in terms of the Helmholtz free energy, f)

If heat, Q, is slowly added to a material, allowing the material to expand reversibly at constant pressure, the heat added, Q, is related to the change in material temperature by the relation:

$$Q = \left(\int Cp\, \mathrm{d}T\right)_p = \Delta H$$

Functions that return Isobaric Specific Heat

$rp\_cptp(fluid, t, p)$	Obtains Specific isobaric Heat Capacity (C_p) of ***superheated vapor*** or ***compressed liquid*** in kJ/kg-K as a function of temperature (t) in K and Pressure (p) in MPa.
$rp\_cpft(fluid, t)$	Obtains Specific isobaric Heat Capacity (C_p) of ***saturated liquid*** in kJ/kg-K as a function of temperature (t) in K.
$rp\_cpfp(fluid, p)$	Obtains Specific isobaric Heat Capacity (C_p) of ***saturated liquid*** in kJ/kg-K as a function of Pressure (p) in MPa.
$rp\_cpgt(fluid, t)$	Obtains Specific isobaric Heat Capacity (C_p) of ***saturated vapor*** in kJ/kg-K as a function of temperature (t) in K.
$rp\_cpgp(fluid, p)$	Obtains Specific isobaric Heat Capacity (C_p) of ***saturated vapor*** in kJ/kg-K as a function of Pressure (p) in MPa.

NOTE : The first parameter in all of these functions is a **"fluid string"** variable that identifies the fluid material to be used for the property calculations.

Application of Units

Mathcad user functions are shown here to calculate specific heat (Cp) with the application of units. These functions are defined in the include file, *Refprop_units.mcdx*.

$$C_{ptp}(fluid,t,p) := \text{rp\_cptp}\left(fluid,\frac{t}{K},\frac{p}{MPa}\right)\cdot\frac{kJ}{kg\cdot K}$$

$$C_{pft}(fluid,t) := \text{rp\_cpft}\left(fluid,\frac{t}{K}\right)\cdot\frac{kJ}{kg\cdot K} \qquad C_{pgt}(fluid,t) := \text{rp\_cpgt}\left(fluid,\frac{t}{K}\right)\cdot\frac{kJ}{kg\cdot K}$$

$$C_{pfp}(fluid,p) := \text{rp\_cpfp}\left(fluid,\frac{p}{MPa}\right)\cdot\frac{kJ}{kg\cdot K} \qquad C_{pgp}(fluid,p) := \text{rp\_cpgp}\left(fluid,\frac{p}{MPa}\right)\cdot\frac{kJ}{kg\cdot K}$$

Example

$fluid := \text{“WATER.fld”}$

Temperatures

$n := 200$

$\Delta T := (T_c - T_t)\cdot\frac{1}{n}$

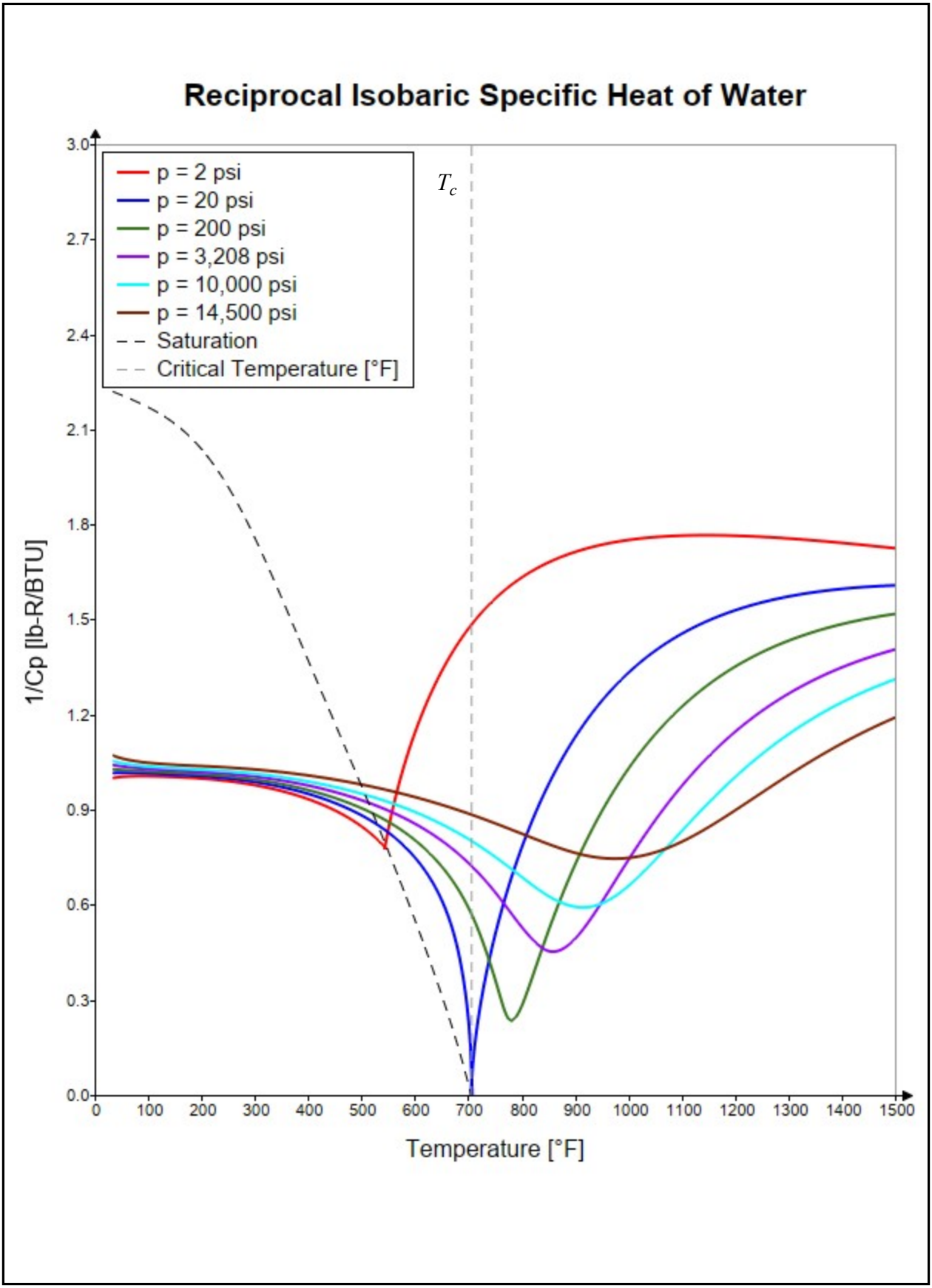
$t1 := T_t, T_t + \Delta T .. 2\cdot T_c$

Pressures

$p1 := 1000\cdot psi$ $\quad p4 := 7500\cdot psi$

$p2 := 3200\cdot psi$ $\quad p5 := 10000\cdot psi$

$p3 := 5000\cdot psi$ $\quad p6 := 14500\cdot psi$


Reciprocal Isobaric Specific Heat of Water
p = 2 psi
p = 20 psi
p = 200 psi
p = 3,208 psi
p = 10,000 psi
p = 14,500 psi
Saturation
Critical Temperature [°F]
T_c
1/Cp [lb-R/BTU]
Temperature [°F]
3.0
2.7
2.4
2.1
1.8
1.5
1.2
0.9
0.6
0.3
0.0
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500

A.6 Isochoric (Isometric) Specific Heat, Cv

Isochoric Specific Heat is defined as the rate of change of enthalpy of a unit mass of material with respect to temperature at a given constant volume (isochoric):

$$c_v = \left(\frac{\mathrm{d}}{\mathrm{d}T} u\right)_V$$

The expression above is evaluated using the internal energy, u. A similar expression can be derived in terms of the Helmholtz free energy.

If heat, Q, is slowly added to a material while keeping the volume constant, the heat added, Q, is related to the change in material temperature by the relation:

$$Q = \left(\int Cv\,\mathrm{d}T\right)_v = \Delta U$$

Functions that return Isobaric Specific Heat

$rp\_cvtp(fluid, t, p)$	Obtains Specific isochoric Heat Capacity (C_v) of ***superheated vapor*** or ***compressed liquid*** in kJ/kg-K as a function of temperature (t) in K and Pressure (p) in MPa.
$rp\_cvfp(fluid, p)$	Obtains Specific isochoric Heat Capacity (C_v) of ***saturated liquid*** in kJ/kg-K as a function of Pressure (p) in MPa.
$rp\_cvfp(fluid, p)$	Obtains Specific isochoric Heat Capacity (C_v) of ***saturated liquid*** in kJ/kg-K as a function of Pressure (p) in MPa.
$rp\_cvgp(fluid, p)$	Obtains Specific isochoric Heat Capacity (C_v) of ***saturated vapor*** in kJ/kg-K as a function of Pressure (p) in MPa.
$rp\_cvgp(fluid, p)$	Obtains Specific isochoric Heat Capacity (C_v) of ***saturated vapor*** in kJ/kg-K as a function of Pressure (p) in MPa.

NOTE : The first parameter in all of these functions is a **"fluid string"** variable that identifies the fluid material to be used for the property calculations.

Application of Units

Mathcad user functions are shown here to calculate isochoric specific heat (Cv) with the application of units. These functions are defined in the include file, *Refprop_units.mcdx*.

$$C_{vtp}(fluid,t,p) := \text{rp\_cvtp}\left(fluid,\frac{t}{K},\frac{p}{MPa}\right)\cdot\frac{kJ}{kg\cdot K}$$

$$C_{vft}(fluid,t) := \text{rp\_cvft}\left(fluid,\frac{t}{K}\right)\cdot\frac{kJ}{kg\cdot K} \qquad C_{vgt}(fluid,t) := \text{rp\_cvgt}\left(fluid,\frac{t}{K}\right)\cdot\frac{kJ}{kg\cdot K}$$

$$C_{vfp}(fluid,p) := \text{rp\_cvfp}\left(fluid,\frac{p}{MPa}\right)\cdot\frac{kJ}{kg\cdot K} \qquad C_{vgp}(fluid,p) := \text{rp\_cvgp}\left(fluid,\frac{p}{MPa}\right)\cdot\frac{kJ}{kg\cdot K}$$

Example

$fluid := \text{“fluids/WATER.fld”}$

Temperatures

$n := 200$

$\Delta T := (T_c - T_t)\cdot\frac{1}{n}$

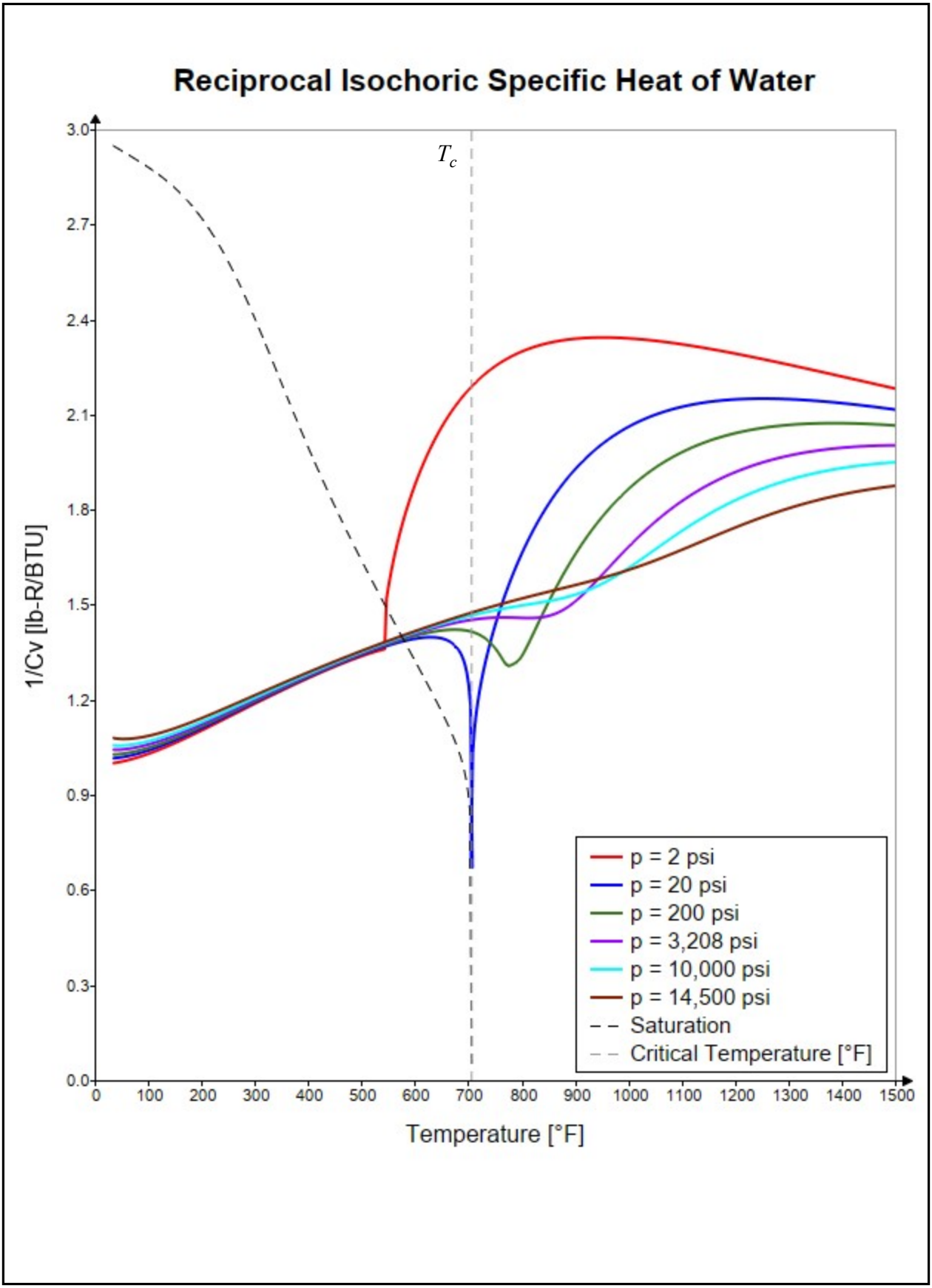
$t1 := T_t, T_t + \Delta T .. 2\cdot T_c$

Pressures

$p1 := 1000\cdot psi$ $\quad p4 := 7500\cdot psi$

$p2 := 3200\cdot psi$ $\quad p5 := 10000\cdot psi$

$p3 := 5000\cdot psi$ $\quad p6 := 14500\cdot psi$


Reciprocal Isochoric Specific Heat of Water
T_c
1/Cv [lb-R/BTU]
Temperature [°F]
3.0
2.7
2.4
2.1
1.8
1.5
1.2
0.9
0.6
0.3
0.0
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
p = 2 psi
p = 20 psi
p = 200 psi
p = 3,208 psi
p = 10,000 psi
p = 14,500 psi
Saturation
Critical Temperature [°F]

4.7 Speed of Sound, w

The speed of sound in a fluid is related to the specific volume and the fluid compressibility by the following thermodynamic relationship:

$$w^2 = -v^2 \cdot \left(\frac{\mathrm{d}}{\mathrm{d}v}P\right)_S = \left(\frac{\mathrm{d}}{\mathrm{d}\rho}P\right)_S$$

RefProp evaluates this differential using the Helmholtz free energy expression.

Functions that return Speed of Sound

$rp\_wtp(fluid, t, p)$	Obtains speed of sound through superheated vapor or compressed liquid in m/s as a function of temperature (t) in K and Pressure (p) in MPa.
$rp\_wft(fluid, t)$	Obtains speed of sound through ***saturated liquid*** in m/s as a function of Temperature (t) in K.
$rp\_wfp(fluid, p)$	Obtains speed of sound through ***saturated liquid*** in m/s as a function of Pressure (p) in MPa.
$rp\_wgt(fluid, t)$	Obtains speed of sound through ***saturated vapor*** in m/s as a function of Temperature (t) in K.
$rp\_wgp(fluid, p)$	Obtains speed of sound through ***saturated vapor*** in m/s as a function of Pressure (p) in MPa.

NOTE : The first parameter in all of these functions is a **"fluid string"** variable that identifies the fluid material to be used for the property calculations.

Application of Units

Mathcad user functions are shown here to calculate speed of sound (w) with the application of units. These functions are defined in the include file, *Refprop_units.mcdx*.

$$w_{tp}(fluid, t, p) := \text{rp\_wtp}\left(fluid, \frac{t}{K}, \frac{p}{MPa}\right) \cdot \frac{m}{s}$$

$$w_{ft}(fluid, t) := \text{rp\_wft}\left(fluid, \frac{t}{K}\right) \cdot \frac{m}{s} \qquad w_{gt}(fluid, t) := \text{rp\_wgt}\left(fluid, \frac{t}{K}\right) \cdot \frac{m}{s}$$

$$w_{fp}(fluid, p) := \text{rp\_wfp}\left(fluid, \frac{p}{MPa}\right) \cdot \frac{m}{s} \qquad w_{gp}(fluid, p) := \text{rp\_wgp}\left(fluid, \frac{p}{MPa}\right) \cdot \frac{m}{s}$$

Example

$fluid := \text{“WATER.fld”}$

Temperatures (°F)

$n := 200$

$\Delta T := (T_c - T_t) \cdot \frac{1}{n}$

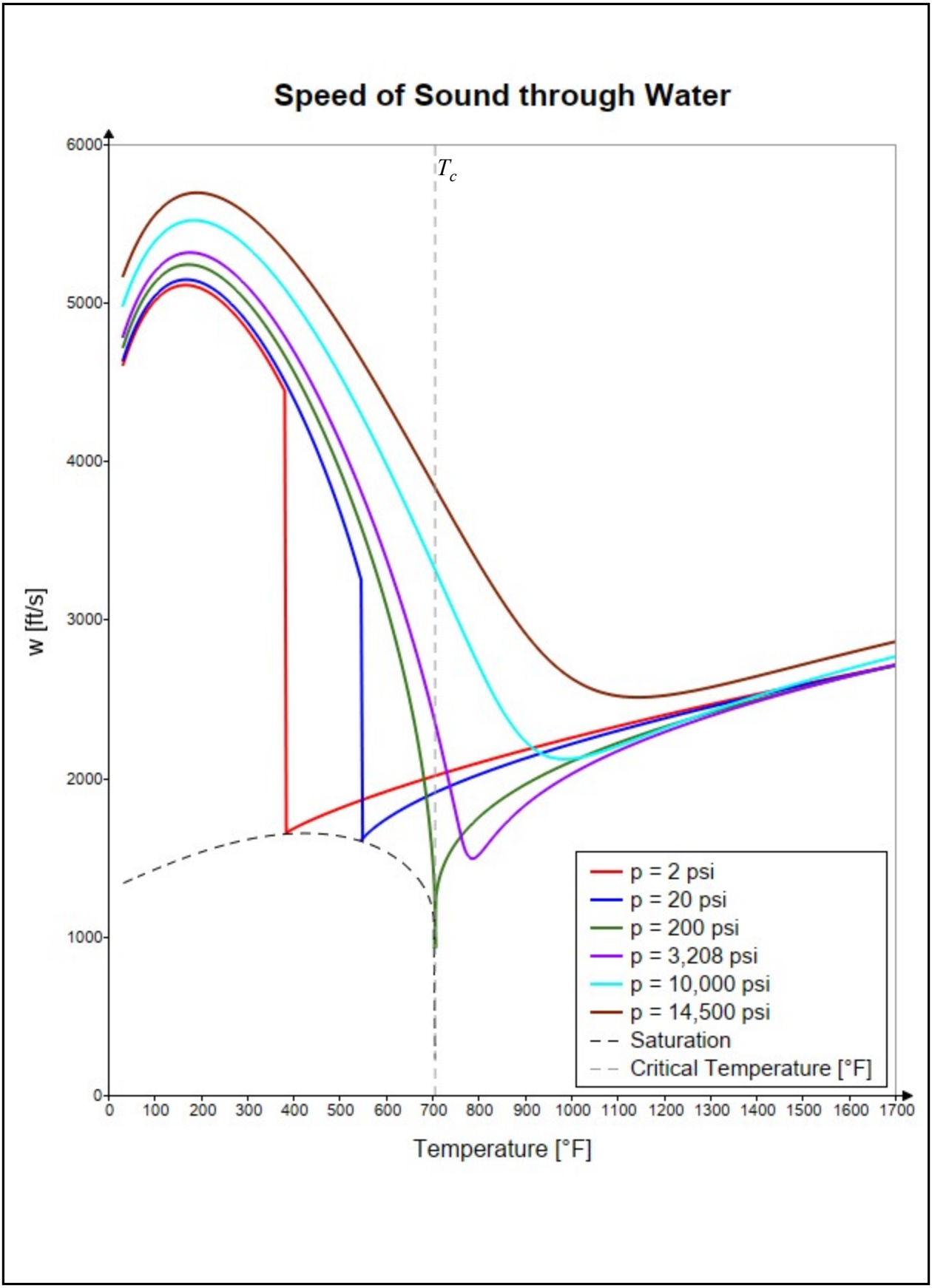
$T2 := T_t, T_t + \Delta T .. 2 \cdot T_c$

Pressures (psi)

$p1 := 200 \cdot psi$ $\quad p4 := 5000 \cdot psi$

$p2 := 1000 \cdot psi$ $\quad p5 := 10000 \cdot psi$

$p3 := 3200 \cdot psi$ $\quad p6 := 14500 \cdot psi$


Speed of Sound through Water
T_c
w [ft/s]
6000
5000
4000
3000
2000
1000
0
0 100 200 300 400 500 600 700 800 900 1000 1100 1200 1300 1400 1500 1600 1700
Temperature [°F]
p = 2 psi
p = 20 psi
p = 200 psi
p = 3,208 psi
p = 10,000 psi
p = 14,500 psi
Saturation
Critical Temperature [°F]

4.8 Saturation Curve

The ***Saturation Curve*** is defined as the Temperature / Pressure boundary curve that separates the liquid phase from the vapor phase. This curve starts at the ***Triple Point*** (where the solid, liquid, and vapor regions meet) to the ***Critical Point*** (where the fluid becomes super-critical; neither liquid or vapor).

The saturation curve and the temperature / pressure points at either end can all be retrieved from the RefProp functions for any fluid.

Functions that return Pressure

$rp\_ttrip(fluid, 0)$	Obtains the ***triple point*** temperature in K. Requires a dummy parameter (0).
$rp\_ptrip(fluid, 0)$	Obtains the ***triple point*** pressure in MPa. Requires a dummy parameter (0).
$rp\_tcrit(fluid, 0)$	Obtains the ***critical point*** temperature in K. Requires a dummy parameter (0).
$rp\_pcrit(fluid, 0)$	Obtains the ***critical point*** pressure in MPa. Requires a dummy parameter (0).
$rp\_rhocrit(fluid, 0)$	Obtains the ***critical point*** density in kg/m³. Requires a dummy parameter (0).
$rp\_tsatp(fluid, p)$	Obtains the ***saturation temperature*** in K from the given saturation pressure (p) in MPa. Valid from P_t to P_c.
$rp\_tsatpf(fluid, p)$	Obtains the ***liquid mixture saturation temperature*** in K from the given saturation pressure (p) in MPa. Valid from P_t to P_c.
$rp\_tsatpg(fluid, p)$	Obtains the ***vapor mixture saturation temperature*** in K from the given saturation pressure (p) in MPa. Valid from P_t to P_c.
$rp\_psatt(fluid, t)$	Obtains the ***saturation pressure*** in MPa from the given saturation temperature (t) in K. Valid from T_t to T_c.
$rp\_psattf(fluid, t)$	Obtains the ***liquid mixture saturation pressure*** in MPa from the given saturation temperature (t) in K. Valid from T_t to T_c.
$rp\_psattg(fluid, t)$	Obtains the ***vapor mixture saturation pressure*** in MPa from the given saturation temperature (t) in K. Valid from T_t to T_c.

NOTE : The first parameter in all of these functions is a **"fluid string"** variable that identifies the fluid material to be used for the property calculations.

Application of Units

Mathcad user functions are shown here to calculate saturation points with the application of units. These functions are defined in the include file, *Refprop_units.mcdx*.

$P_t(fluid) := \text{rp\_ptrip}(fluid, 0) \cdot MPa$ $\quad P_{sat}(fluid, t) := \text{rp\_psatt}\left(fluid, \frac{t}{K}\right) \cdot MPa$

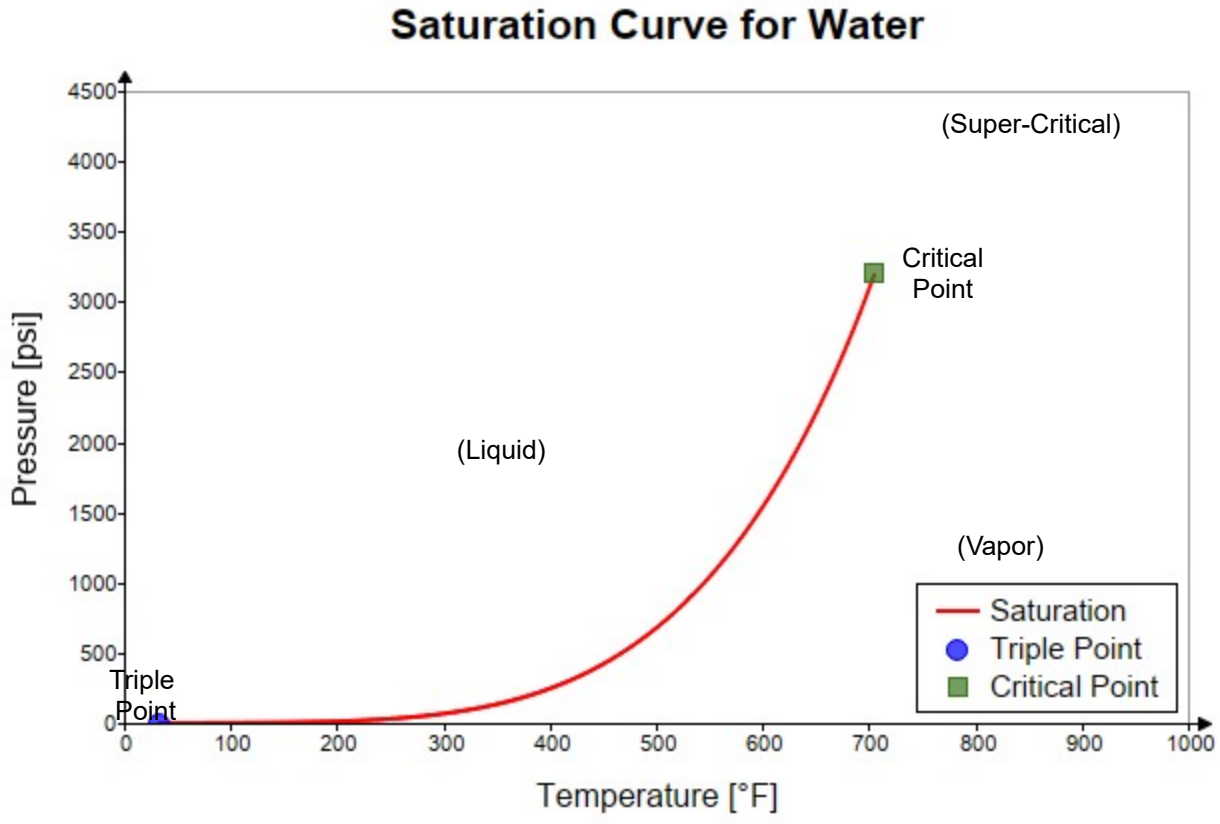
$T_t(fluid) := \text{rp\_ttrip}(fluid, 0) \cdot K$ $\quad T_{sat}(fluid, p) := \text{rp\_tsatp}\left(fluid, \frac{p}{MPa}\right) \cdot K$

$T_c(fluid) := \text{rp\_tcrit}(fluid, 0) \cdot K$ $\quad P_c(fluid) := \text{rp\_pcrit}(fluid, 0) \cdot MPa$

Example

$fl := \text{“WATER.fld”}$ $\quad n := 1000$

Temperatures $\quad \Delta T := \left(T_c(fl) - T_t(fl)\right) \cdot \frac{1}{n}$ $\quad TI := T_t(fl), T_t(fl) + \Delta T .. T_c(fl)$



4.9 Surface Tension, σ

Surface Tension is a property that describes the work required to create a unit surface area of liquid water. Its units are N-m/m² or N/m and it is a function of temperature alone (decreases with increasing temperature). Surface Tension is important in the formation of bubbles and in the study of atomization. As this is a property of the vapor / liquid interface, it is only defined along the saturation curve and has no meaning in the single phase regions.

Functions that return Surface Tension

$rp\_surften(fluid, t)$ Obtains surface tension of the fluid in kg/s² as a function of temperature (t) in K.

NOTE : The first parameter in all of these functions is a **"fluid string"** variable that identifies the fluid material to be used for the property calculations.

Example

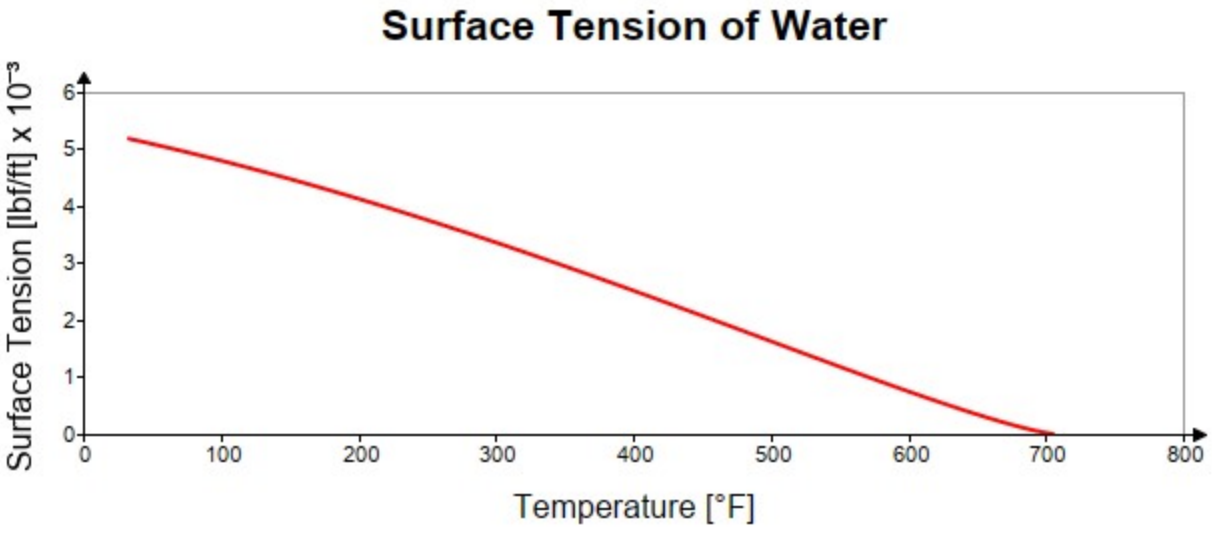
$fluid := \text{“WATER.fld”}$ $n := 100$

Temperatures (°F)

$\Delta T_s := (T_c(fluid) - T_t(fluid)) \div n$

$T_s := T_t(fluid), T_t(fluid) + \Delta T_s .. T_c(fluid)$

$\sigma(T) := \text{rp\_surften}\left(fluid, \frac{T}{K}\right) \cdot \frac{N}{m}$



4.10 Thermal Conductivity, k

Thermal Conductivity arises from the law of conduction in the study of heat transfer. It is often referred to as a constant of proportionality. It provides important information on the heat conduction rate given a temperature gradient and surface area perpendicular to the heat flow. Thermal conductivity is also used in many empirically derived relationships which are important in the study of heat transfer.

Functions that return Thermal Conductivity

Mathcad user functions are shown here to calculate thermal conductivity (k) with the application of units. These functions are defined in the include file, *Refprop_units.mcdx*.

$rp\_ktp(fluid, t, p)$	Obtains thermal conductivity (k) of superheated vapor or compressed liquid in kW/m-K as a function of temperature (t) in K and Pressure (p) in MPa.
$rp\_kft(fluid, t)$	Obtains thermal conductivity (k) of saturated liquid in kW/m-K as a function of Temperature (t) in K.
$rp\_kfp(fluid, p)$	Obtains thermal conductivity (k) of saturated liquid in kW/m-K as a function of Pressure (p) in MPa.
$rp\_kgt(fluid, t)$	Obtains thermal conductivity (k) of saturated vapor in kW/m-K as a function of Temperature (t) in K.
$rp\_kgp(fluid, p)$	Obtains thermal conductivity (k) of saturated vapor in kW/m-K as a function of Pressure (p) in MPa.

NOTE : The first parameter in all of these functions is a **"fluid string"** variable that identifies the fluid material to be used for the property calculations.

Application of Units

$$k_{tp}(fluid, T, P) := \text{rp\_ktp}\left(fluid, \frac{T}{K}, \frac{P}{MPa}\right) \cdot \frac{W}{m \cdot K}$$

$$k_{ft}(fluid, T) := \text{rp\_kft}\left(fluid, \frac{T}{K}\right) \cdot \frac{W}{m \cdot K} \qquad k_{gt}(fluid, T) := \text{rp\_kgt}\left(fluid, \frac{T}{K}\right) \cdot \frac{W}{m \cdot K}$$

$$k_{fp}(fluid, P) := \text{rp\_kfp}\left(fluid, \frac{P}{MPa}\right) \cdot \frac{W}{m \cdot K} \qquad k_{gp}(fluid, P) := \text{rp\_kgp}\left(fluid, \frac{P}{MPa}\right) \cdot \frac{W}{m \cdot K}$$

Example $fluid := \text{"WATER.fld"}$

Pressures (psi)

$p1 := 200 \cdot psi$ $p4 := 5000 \cdot psi$

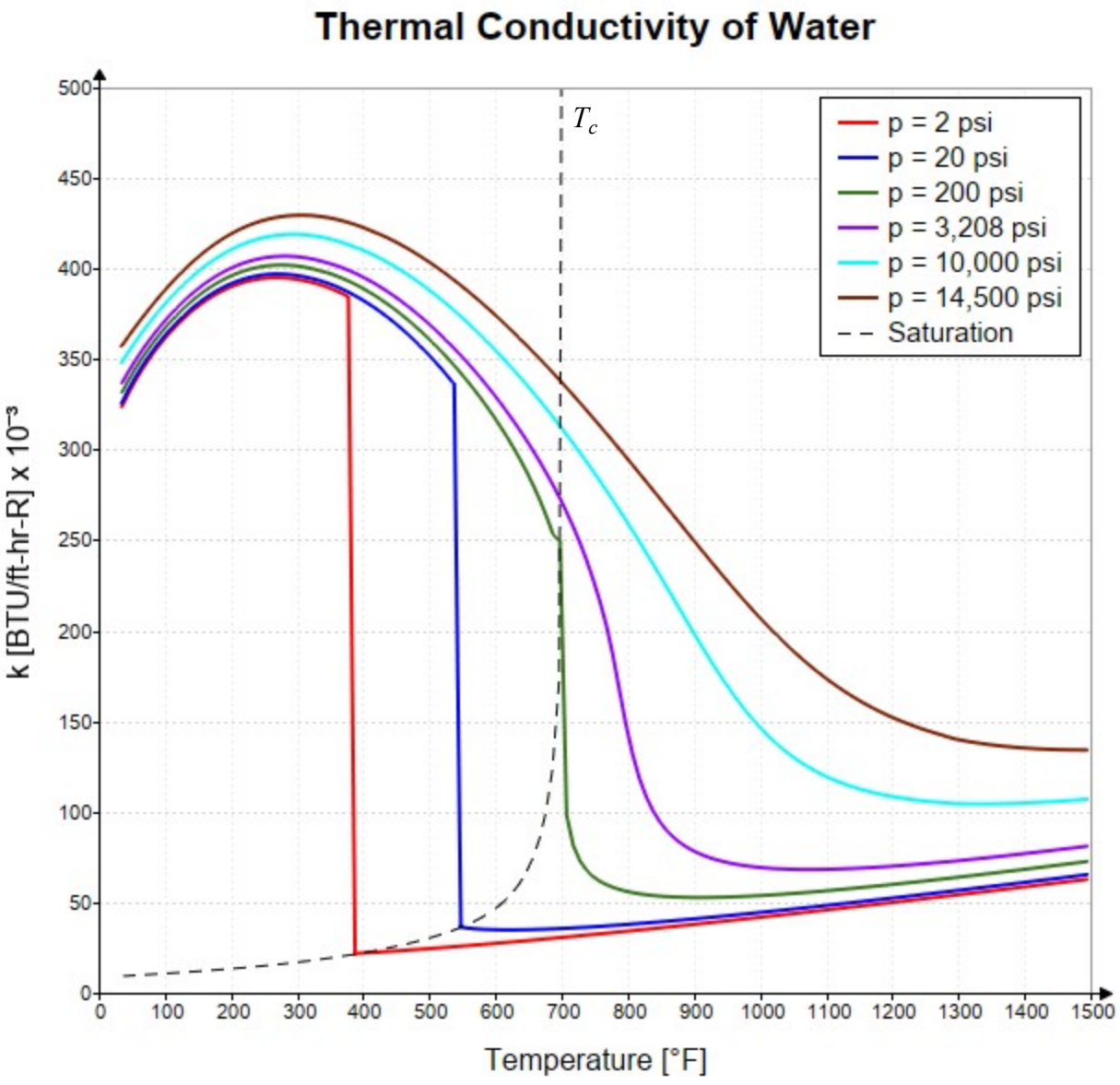
$p2 := 1000 \cdot psi$ $p5 := 10000 \cdot psi$

$p3 := 3000 \cdot psi$ $p6 := 14500 \cdot psi$

Temperatures (°F)

$t1 := 35\ °F, 45\ °F .. 1500\ °F$

$T_c := \text{rp\_tcrit}(fluid, 0) \cdot K$



4.11 Viscosity, μ

Viscosity is a property derived out of the study of Newtonian fluid mechanics. To analyze fluid motion, it is necessary to express the resistive forces in terms of velocity. The resistance to the shearing motion of flow is given by viscosity. It is used in determining dimensionless parameters (i.e. Reynold's number, etc.) as well as in many empirical correlations for heat transfer and fluid flow.

Functions that return Viscosity

Mathcad user functions are shown here to calculate viscosity (μ) with the application of units. These functions are defined in the include file, *Refprop_units.mcdx*.

$rp\_mutp(fluid, t, p)$	Obtains absolute (dynamic) viscosity of superheated vapor or compressed liquid in μPa-sec as a function of temperature (t) in K and Pressure (p) in MPa.
$rp\_muft(fluid, t)$	Obtains absolute (dynamic) viscosity of saturated liquid in μPa-sec as a function of Temperature (t) in K.
$rp\_mufp(fluid, p)$	Obtains absolute (dynamic) viscosity of saturated liquid in μPa-sec as a function of Pressure (p) in MPa.
$rp\_mugt(fluid, t)$	Obtains absolute (dynamic) viscosity of saturated vapor in μPa-sec as a function of Temperature (t) in K.
$rp\_mugp(fluid, p)$	Obtains absolute (dynamic) viscosity of saturated vapor in μPa-sec as a function of Pressure (p) in MPa.

NOTE : The first parameter in all of these functions is a **"fluid string"** variable that identifies the fluid material to be used for the property calculations.

Application of Units

$$\mu_{tp}(fluid, T, P) := \text{rp\_mutp}\left(fluid, \frac{T}{K}, \frac{P}{MPa}\right) \cdot \mu Pa \cdot s$$

$$\mu_{ft}(fluid, T) := \text{rp\_muft}\left(fluid, \frac{T}{K}\right) \cdot \mu Pa \cdot s \qquad \mu_{gt}(fluid, T) := \text{rp\_mugt}\left(fluid, \frac{T}{K}\right) \cdot \mu Pa \cdot s$$

$$\mu_{fp}(fluid, P) := \text{rp\_mufp}\left(fluid, \frac{P}{MPa}\right) \cdot \mu Pa \cdot s \qquad \mu_{gp}(fluid, P) := \text{rp\_mugp}\left(fluid, \frac{P}{MPa}\right) \cdot \mu Pa \cdot s$$

Example

$fluid := \text{“WATER.fld”}$

Pressures

$p1 := 200 \cdot psi$ $p4 := 5000 \cdot psi$

$p2 := 1000 \cdot psi$ $p5 := 10000 \cdot psi$

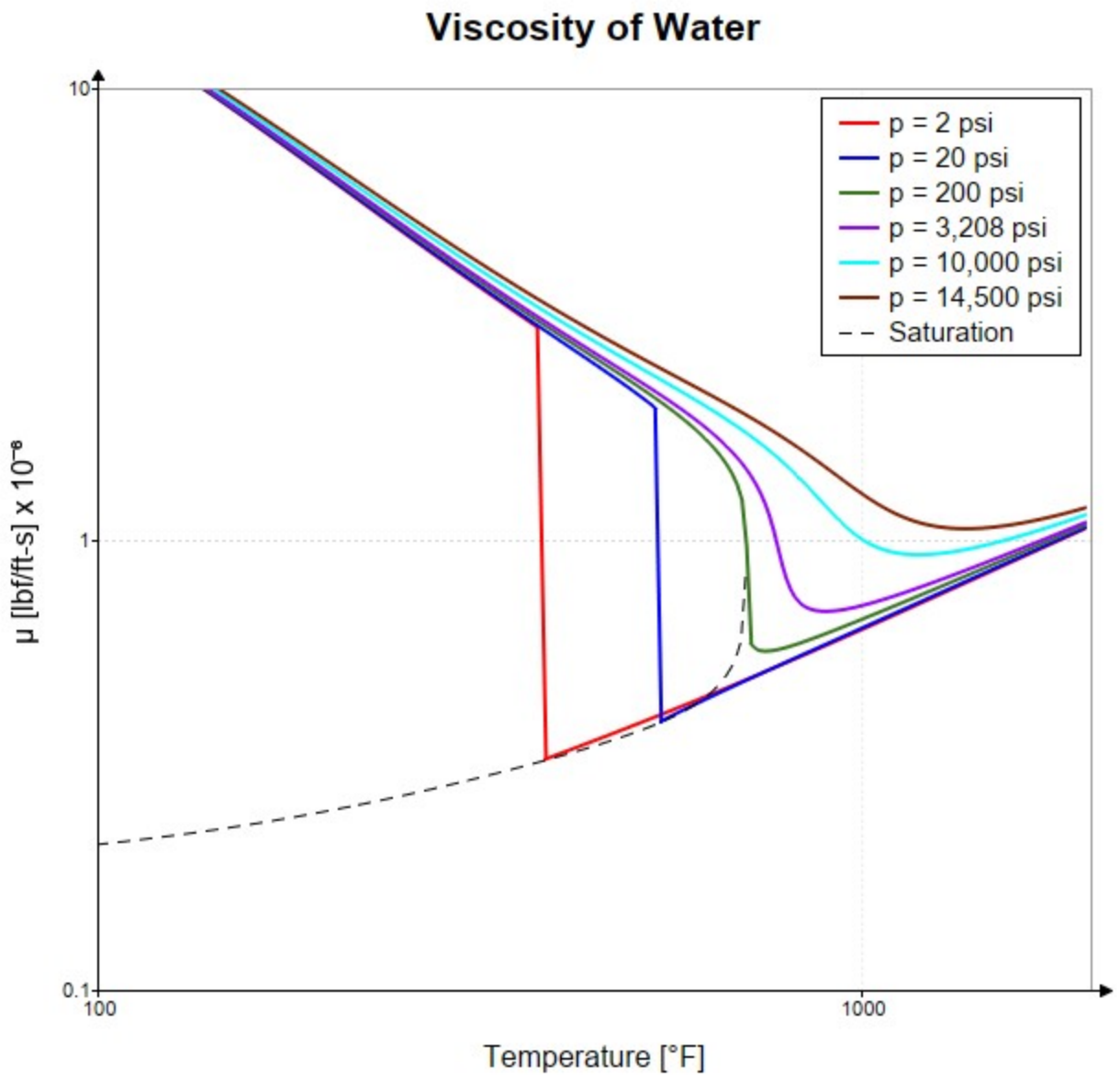
$p3 := 3200 \cdot psi$ $p6 := 14500 \cdot psi$

Temperatures

$t1 := 35\ °F, 45\ °F .. 5000\ °F$

$T_c := \text{rp\_tcrit}(fluid, 0) \cdot K$

$T_c = 705.103\ °F$



Appendix B - RefProp Function Verification

The ***RefProp*** functions have been tested and verified against other RefProp implementations and published data. Verification is based on the following criteria.

Continuity Check

All supplied functions have been plotted over the valid property limits for all of the fluids in the NIST RefProp Library to ensure continuity of the data with no areas of non-convergence of the underlying NIST RefProp functions. Water, CO2, and most of the room temperature gasses have been shown to be continuous over the full Temperature and Pressure limits of the models. This includes ranges tightly focused around the critical point (Figure 1).

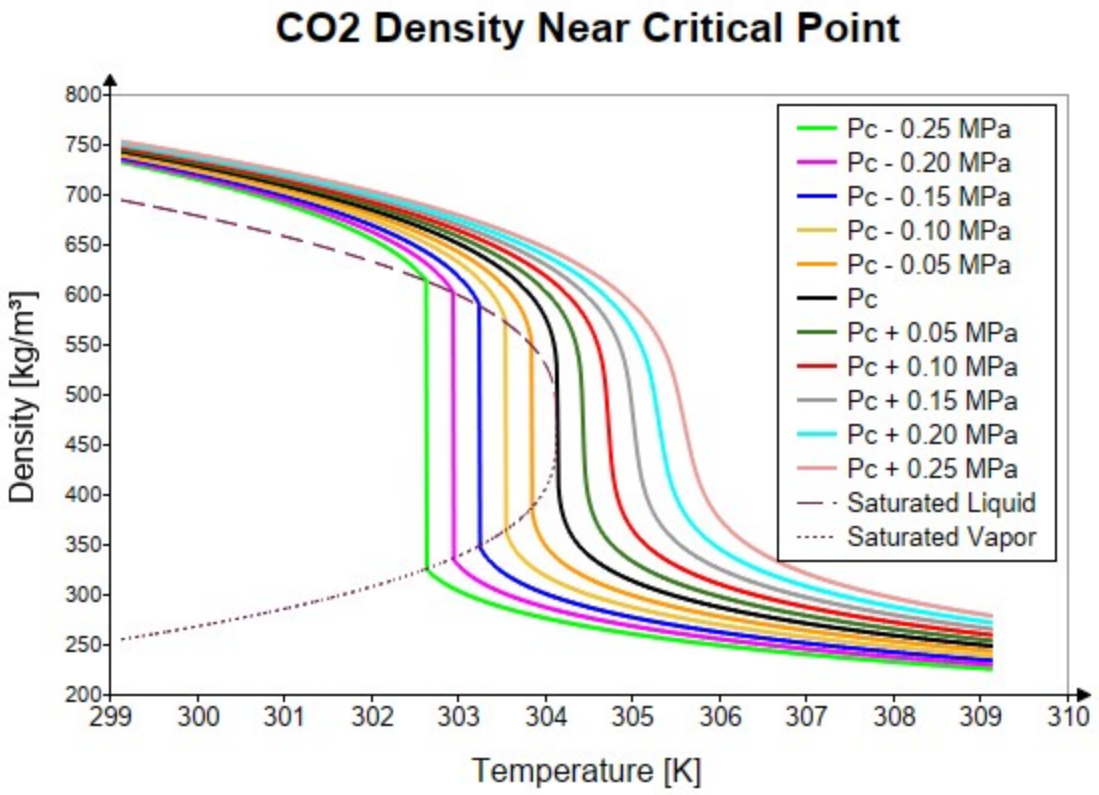


Figure 1 : Continuity of Calculated Density of Carbon Dioxide in the Vicinity of the Critical Point

When using any fluid from RefProp (or any properties library), care should be taken to verify the continuity over the region of interest, and accuracy of the saturation curve as it approaches the critical point; similar to Figure 1 and Figure 2.

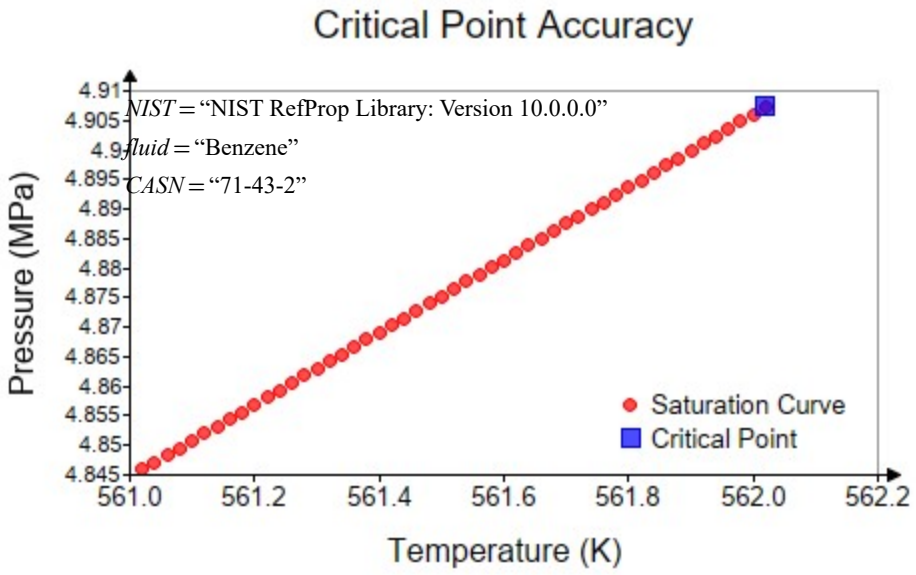


Figure 2 : Continuity of Benzene Saturation Curve up to the Critical Point

Accuracy

Comparisons have been made for CO_2 property results from the Mathcad DLL and the NIST provided Excel Add-in at selected Temperature and Pressure point locations. These comparisons (Figure 3) show that the two implementations agree to 5 significant figures or better on the calculated thermodynamic and transport properties. This comparison provides verification that the Mathcad Add-In is at least as accurate as the NIST Library, but makes no claim for the accuracy of the NIST Library itself.

Error =

	ρ	H	S	U	Cp	Cv	w	μ	k
	1	2	3	4	5	6	7	8	9
1	$4.3 \cdot 10^{6}$	$-1.6 \cdot 10^{5}$	$-1.6 \cdot 10^{5}$	$-1.6 \cdot 10^{5}$	$-1.6 \cdot 10^{5}$	$-1.6 \cdot 10^{5}$	$9.9 \cdot 10^{8}$	$1.3 \cdot 10^{6}$	$-1.5 \cdot 10^{5}$
2	$4.3 \cdot 10^{6}$	$-1.6 \cdot 10^{5}$	$-1.6 \cdot 10^{5}$	$-1.6 \cdot 10^{5}$	$-1.6 \cdot 10^{5}$	$-1.6 \cdot 10^{5}$	$5.1 \cdot 10^{8}$	$1.3 \cdot 10^{6}$	$-1.5 \cdot 10^{5}$
3	$5.3 \cdot 10^{6}$	$-1.6 \cdot 10^{5}$	$-1.6 \cdot 10^{5}$	$-1.6 \cdot 10^{5}$	$-1.4 \cdot 10^{5}$	$-1.5 \cdot 10^{5}$	$-2.5 \cdot 10^{7}$	$1.9 \cdot 10^{6}$	$-1.4 \cdot 10^{5}$
4	$1.3 \cdot 10^{5}$	$-2 \cdot 10^{5}$	$-1.9 \cdot 10^{5}$	$-2 \cdot 10^{5}$	$-1.2 \cdot 10^{4}$	$-3.1 \cdot 10^{5}$	$1.9 \cdot 10^{5}$	$1.5 \cdot 10^{5}$	$-3.2 \cdot 10^{5}$
5	$3.3 \cdot 10^{6}$	$-1.6 \cdot 10^{5}$	$-1.6 \cdot 10^{5}$	$-1.6 \cdot 10^{5}$	$-1.7 \cdot 10^{5}$	$-1.6 \cdot 10^{5}$	$1.4 \cdot 10^{6}$	$2.4 \cdot 10^{6}$	$-1.4 \cdot 10^{5}$
6	$4.4 \cdot 10^{6}$	$-1.6 \cdot 10^{5}$	$-1.6 \cdot 10^{5}$	$-1.6 \cdot 10^{5}$	$-1.6 \cdot 10^{5}$	$-1.6 \cdot 10^{5}$	$1.4 \cdot 10^{7}$	$1.5 \cdot 10^{6}$	$-1.5 \cdot 10^{5}$

$$\overrightarrow{\text{min\_sig\_digits}(\text{Error})} = \begin{pmatrix} 6 & 5 & 5 & 5 & 5 & 5 & 8 & 6 & 5 \\ 6 & 5 & 5 & 5 & 5 & 5 & 8 & 6 & 5 \\ 6 & 5 & 5 & 5 & 5 & 5 & 7 & 6 & 5 \\ 5 & 5 & 5 & 5 & 4 & 5 & 5 & 5 & 5 \\ 6 & 5 & 5 & 5 & 5 & 5 & 6 & 6 & 5 \\ 6 & 5 & 5 & 5 & 5 & 5 & 7 & 6 & 5 \end{pmatrix}$$

$$\text{RMS}_{\text{error}} = 2.2 \times 10^{-5}$$

$$\text{precision} = 5$$

The Mathcad and Excel implementations of the NIST Refprop DLL agree to within 5 significant figures over the comparison points chosen.

Figure 3 : Mathcad Add-In Accuracy Relative to the NIST MS-Excel Add-In

Since the RefProp property models for Water are taken from the IAPWS-95 Steam/ Water Property Formulation for General and Scientific Use, the Mathcad predicted water properties were compared directly with the IAPWS-95 published verification values. Thus providing verification that the Mathcad Add-In implementation functioning as expected. The IAPWS-95 release document provides a table of property values over a matrix of state points (Figure 4) lying in all liquid/vapor/super-critical phase regions of the phase diagram.

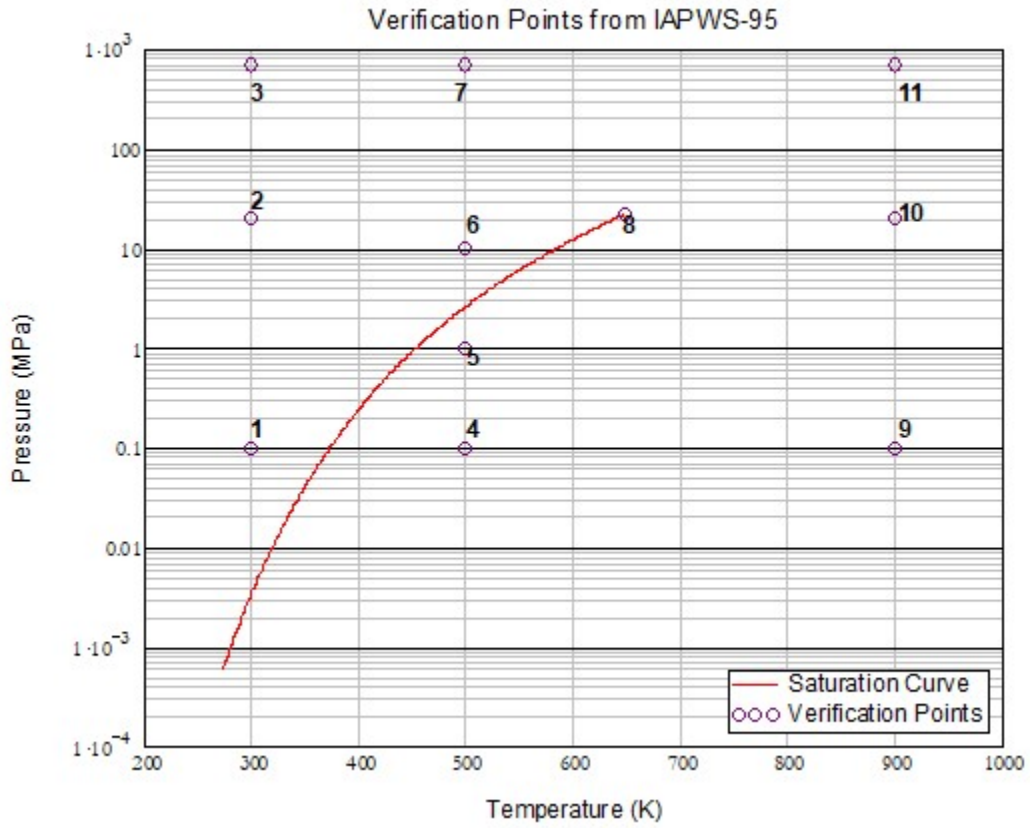


Figure 4 : IAPWS-95 Recommended Verification Points

Using the direct functions of Temperature and Density, the predicted pressures agree with the published verification values to 6 significant figures or better. All other properties (C_v, w, and S) agreed to 9 significant figures or better, which is the precision of the published values.

Indirectly using the functions of Temperature and Pressure to calculate Density, requires an iterative procedure within the NIST RefProp Library that solves the Pressure equation for Density to a selected tolerance. This function evaluates Density to ***6 significant figures*** (or better) of the published values. Since this Density calculation is used as a first step in the calculation of all other properties as a function of Temperature and Pressure, the remaining properties are also limited to an accuracy of ***6 significant figures***. According to the IAPWS-95 documentation, the formulation itself (and hence the verification values) is only accurate to 5 significant figures or less in the liquid region, and 4 significant figures or less in the vapor and super-critical regions.

Appendix C Unit Functions Reference

NIST RefProp Functions with Appropriately Applied Units

(For Mathcad Prime and Version 2.0 or later of the RefProp Add-In)

Most property routines listed below require a fluid string, fl, to specify the fluid properties to use. Make calls to $rp\_property$ functions with one of the RefProp fluids (e.g. $fl := \text{“CO2.fld”}$) or mixtures. Fluid names are not case sensitive.

Add-in Utility Functions

Function to get the version string:

$Refprop\_Addin\_Version(i) := \text{rp\_getvers}(i)$ $isV10? := \text{rp\_getRPnum}(0) \geq 10 = 1$

$NIST\_Refprop\_Version(i) := \text{rp\_getNIST}(i)$

Legacy API Calls

Fluid Information Functions

$\text{Name}(fl, component) := \text{rp\_getname}(fl, component)$ Returns a string with the full fluid name

$\text{CASN}(fl, component) := \text{rp\_getcasn}(fl, component)$ Returns a string containing the CAS number

The following functions all retrieve specific properties and constants. The parameters to each function (unless it's a component number) should have appropriate Mathcad units applied. Each of these functions will convert the passed parameters to the units expected by the RefProp functions, and return values with appropriate units applied. Temperatures should always be passed with absolute temperature units of K or R.

Basic Physical Constants by individual component (component = 1 for a pure fluid)

Molecular Weight $WM(fl, comp) := rp\_wmol(fl, comp) \cdot \frac{gm}{mol}$

$R_g(fl, comp) := rp\_rgas(fl, comp) \cdot J \cdot mol^{-1} \cdot K^{-1}$

Critical Point

Temperature $T_c(fl, component) := rp\_tcrit(fl, component) \cdot K$

Pressure $P_c(fl, component) := rp\_pcrit(fl, component) \cdot MPa$

Density $\rho_c(fl, component) := rp\_rhocrit(fl, component) \cdot \frac{kg}{m^3}$

Triple Point

$T_t(fl, component) := rp\_ttrip(fl, component) \cdot K$

$P_t(fl, component) := rp\_ptrip(fl, component) \cdot MPa$

Temperature and Pressure Functions along the Saturation Curve

$T_{sat}(fl, P) := rp\_tsatp\left(fl, \frac{P}{MPa}\right) \cdot K$

$P_{sat}(fl, T) := rp\_psatt\left(fl, \frac{T}{K}\right) \cdot MPa$

$T_{satf}(fl, P) := rp\_tsatpf\left(fl, \frac{P}{MPa}\right) \cdot K$

$P_{satf}(fl, T) := rp\_psattf\left(fl, \frac{T}{K}\right) \cdot MPa$

$T_{satg}(fl, P) := rp\_tsatpg\left(fl, \frac{P}{MPa}\right) \cdot K$

$P_{satg}(fl, T) := rp\_psattg\left(fl, \frac{T}{K}\right) \cdot MPa$

Cricondentherm and Cricondenbar Temperature and Pressure Functions along the Saturation Curve

$T_{maxT}(fl) := rp\_maxX(fl, \text{“T”})_0 \, K$

$T_{maxP}(fl) := rp\_maxX(fl, \text{“P”})_0 \, K$

$P_{maxT}(fl) := rp\_maxX(fl, \text{“T”})_1 \, MPa$

$P_{maxP}(fl) := rp\_maxX(fl, \text{“P”})_1 \, MPa$

$$\overrightarrow{LIMITS(fl, mStr) := rp\_limits(fl, mStr)} \begin{bmatrix} K \\ K \\ kg\ m^{-3} \\ MPa \end{bmatrix}$$

Thermodynamic Properties as functions of Temperature and Pressure

Fluid function subscripts indicate the phase region and the independent parameter to be used.
A subscript of **f** indicates saturated liquid (fluid) and a subscript of **g** indicates saturated vapor (*gas*). If neither **f** or **g** is used, the function returns either subcooled liquid or superheated vapor values. These functions are only defined between the triple point and the critical point, and will return an error otherwise.

A second subscript of **t**, **p**, or both indicates the independent parameter that should be passed. Thus, ρ_{fp} would return the saturated liquid density as a function of the saturation pressure, p. Alternatively, ρ_{tp} will return the general subcooled liquid or superheated vapor density as a function of both temperature, t, and pressure, p.

Density, ρ

Saturated Liquid as a function of temperature:

$$\rho_{ft}(fl,T) := \text{rp\_rhoft}\left(fl, \frac{T}{K}\right)\cdot\frac{kg}{m^3}$$

Saturated Vapor as a function of temperature:

$$\rho_{gt}(fl,T) := \text{rp\_rhogt}\left(fl, \frac{T}{K}\right)\cdot\frac{kg}{m^3}$$

Saturated Liquid as a function of temperature:

$$\rho_{fp}(fl,P) := \text{rp\_rhofp}\left(fl, \frac{P}{MPa}\right)\cdot\frac{kg}{m^3}$$

Saturated Vapor as a function of temperature:

$$\rho_{gp}(fl,P) := \text{rp\_rhogp}\left(fl, \frac{P}{MPa}\right)\cdot\frac{kg}{m^3}$$

Subcooled Liquid or Superheated Vapor as a function of both temperature and pressure:

$$\rho_{tp}(fl,T,P) := \text{rp\_rhotp}\left(fl, \frac{T}{K}, \frac{P}{MPa}\right)\cdot\frac{kg}{m^3}$$

Enthalpy, h

$$h_{ft}(fl,T) := \text{rp\_hft}\left(fl, \frac{T}{K}\right)\cdot\frac{kJ}{kg}$$

$$h_{gt}(fl,T) := \text{rp\_hgt}\left(fl, \frac{T}{K}\right)\cdot\frac{kJ}{kg}$$

$$h_{fp}(fl,P) := \text{rp\_hfp}\left(fl, \frac{P}{MPa}\right)\cdot\frac{kJ}{kg}$$

$$h_{gp}(fl,P) := \text{rp\_hgp}\left(fl, \frac{P}{MPa}\right)\cdot\frac{kJ}{kg}$$

$$h_{tp}(fl,T,P) := \text{rp\_htp}\left(fl, \frac{T}{K}, \frac{P}{MPa}\right)\cdot\frac{kJ}{kg}$$

Entropy, s

$$s_{ft}(fl,T) := \text{rp\_sft}\left(fl, \frac{T}{K}\right)\cdot\frac{kJ}{kg\cdot K}$$

$$s_{gt}(fl,T) := \text{rp\_sgt}\left(fl, \frac{T}{K}\right)\cdot\frac{kJ}{kg\cdot K}$$

$$s_{fp}(fl,P) := \text{rp\_sfp}\left(fl, \frac{P}{MPa}\right)\cdot\frac{kJ}{kg\cdot K}$$

$$s_{gp}(fl,P) := \text{rp\_sgp}\left(fl, \frac{P}{MPa}\right)\cdot\frac{kJ}{kg\cdot K}$$

$$s_{tp}(fl,T,P) := \text{rp\_stp}\left(fl, \frac{T}{K}, \frac{P}{MPa}\right)\cdot\frac{kJ}{kg\cdot K}$$

Internal Energy, u

$$u_{fp}(fl,P) := \text{rp\_ufp}\left(fl, \frac{P}{MPa}\right)\cdot\frac{kJ}{kg}$$

$$u_{gp}(fl,P) := \text{rp\_ugp}\left(fl, \frac{P}{MPa}\right)\cdot\frac{kJ}{kg}$$

$$u_{ft}(fl,T) := \text{rp\_uft}\left(fl, \frac{T}{K}\right)\cdot\frac{kJ}{kg}$$

$$u_{gt}(fl,T) := \text{rp\_ugt}\left(fl, \frac{T}{K}\right)\cdot\frac{kJ}{kg}$$

$$u_{tp}(fl,T,P) := \text{rp\_utp}\left(fl, \frac{T}{K}, \frac{P}{MPa}\right)\cdot\frac{kJ}{kg}$$

Isobaric Specific Heat, **Cp**

$$Cp_{fp}(fl,P) := rp\_cpfp\left(fl, \frac{P}{MPa}\right)\cdot\frac{kJ}{kg\cdot K} \qquad Cp_{gp}(fl,P) := rp\_cpgp\left(fl, \frac{P}{MPa}\right)\cdot\frac{kJ}{kg\cdot K}$$

$$Cp_{ft}(fl,T) := rp\_cpft\left(fl, \frac{T}{K}\right)\cdot\frac{kJ}{kg\cdot K} \qquad Cp_{gt}(fl,T) := rp\_cpgt\left(fl, \frac{T}{K}\right)\cdot\frac{kJ}{kg\cdot K}$$

$$Cp_{tp}(fl,T,P) := rp\_cptp\left(fl, \frac{T}{K}, \frac{P}{MPa}\right)\cdot\frac{kJ}{kg\cdot K}$$

Isochoric Specific Heat, **Cv**

$$Cv_{fp}(fl,P) := rp\_cvfp\left(fl, \frac{P}{MPa}\right)\cdot\frac{kJ}{kg\cdot K} \qquad Cv_{gp}(fl,P) := rp\_cvgp\left(fl, \frac{P}{MPa}\right)\cdot\frac{kJ}{kg\cdot K}$$

$$Cv_{ft}(fl,T) := rp\_cvft\left(fl, \frac{T}{K}\right)\cdot\frac{kJ}{kg\cdot K} \qquad Cv_{gt}(fl,T) := rp\_cvgt\left(fl, \frac{T}{K}\right)\cdot\frac{kJ}{kg\cdot K}$$

$$Cv_{tp}(fl,T,P) := rp\_cvtp\left(fl, \frac{T}{K}, \frac{P}{MPa}\right)\cdot\frac{kJ}{kg\cdot K}$$

Sonic Velocity, w

$$w_{fp}(fl,P) := rp\_wfp\left(fl, \frac{P}{MPa}\right)\cdot\frac{m}{s} \qquad w_{gp}(fl,P) := rp\_wgp\left(fl, \frac{P}{MPa}\right)\cdot\frac{m}{s}$$

$$w_{ft}(fl,T) := rp\_wft\left(fl, \frac{T}{K}\right)\cdot\frac{m}{s} \qquad w_{gt}(fl,T) := rp\_wgt\left(fl, \frac{T}{K}\right)\cdot\frac{m}{s}$$

$$w_{tp}(fl,T,P) := rp\_wtp\left(fl, \frac{T}{K}, \frac{P}{MPa}\right)\cdot\frac{m}{s}$$

Transport Properties as Functions of Temperature and Pressure

Thermal Conductivity, **k**

$k_{fp}(fl,P) := rp\_kfp\left(fl, \frac{P}{MPa}\right) \cdot \frac{W}{m \cdot K}$

$k_{gp}(fl,P) := rp\_kgp\left(fl, \frac{P}{MPa}\right) \cdot \frac{W}{m \cdot K}$

$k_{ft}(fl,T) := rp\_kft\left(fl, \frac{T}{K}\right) \cdot \frac{W}{m \cdot K}$

$k_{gt}(fl,T) := rp\_kgt\left(fl, \frac{T}{K}\right) \cdot \frac{W}{m \cdot K}$

$k_{tp}(fl,T,P) := rp\_ktp\left(fl, \frac{T}{K}, \frac{P}{MPa}\right) \cdot \frac{W}{m \cdot K}$

Fluid Viscosity, **μ**

$\mu_{fp}(fl,P) := rp\_mufp\left(fl, \frac{P}{MPa}\right) \cdot \mu Pa \cdot s$

$\mu_{gp}(fl,P) := rp\_mugp\left(fl, \frac{P}{MPa}\right) \cdot (\mu Pa \cdot s)$

$\mu_{ft}(fl,T) := rp\_muft\left(fl, \frac{T}{K}\right) \cdot (\mu Pa \cdot s)$

$\mu_{gt}(fl,T) := rp\_mugt\left(fl, \frac{T}{K}\right) \cdot (\mu Pa \cdot s)$

$\mu_{tp}(fl,T,P) := rp\_mutp\left(fl, \frac{T}{K}, \frac{P}{MPa}\right) \cdot (\mu Pa \cdot s)$

Some Additional Property Functions

Surface Tension

$$\sigma_t(fl,T) := \text{rp\_surften}\left(fl, \frac{T}{K}\right)\cdot\frac{N}{m}$$

Latent Heat

$$h_{fg}(fl,t) := h_{gt}(fl,t) - h_{ft}(fl,t)$$

Vapor Quality

$$x_{th}(f,t,h) := min\left(\max\left(\frac{h-h_{ft}(f,t)}{h_{gt}(f,t)-h_{ft}(f,t)},0\right),1.0\right)$$

$$x_{ph}(f,p,h) := min\left(\max\left(\frac{h-h_{fp}(f,p)}{h_{gp}(f,p)-h_{fp}(f,p)},0\right),1.0\right)$$

$$x_{ts}(f,t,s) := min\left(\max\left(\frac{s-s_{ft}(f,t)}{s_{gt}(f,t)-s_{ft}(f,t)},0\right),1.0\right)$$

$$x_{ps}(f,p,s) := min\left(\max\left(\frac{s-s_{fp}(f,p)}{s_{gp}(f,p)-s_{fp}(f,p)},0\right),1.0\right)$$

$$x_{t\rho}(f,t,\rho) := min\left(\max\left(\frac{\rho-\rho_{ft}(f,t)}{\rho_{gt}(f,t)-\rho_{ft}(f,t)},0\right),1.0\right)$$

$$x_{p\rho}(f,p,\rho) := min\left(\max\left(\frac{\rho-\rho_{fp}(f,p)}{\rho_{gp}(f,p)-\rho_{fp}(f,p)},0\right),1.0\right)$$

$$x_{tu}(f,t,u) := min\left(\max\left(\frac{u-u_{ft}(f,t)}{u_{gt}(f,t)-u_{ft}(f,t)},0\right),1.0\right)$$

$$x_{pu}(f,p,u) := min\left(\max\left(\frac{u-u_{fp}(f,p)}{u_{gp}(f,p)-u_{fp}(f,p)},0\right),1.0\right)$$

2-phase Properties

$$h_{tx}(f,t,x) := h_{ft}(f,t)\cdot(1-x) + h_{gt}(f,t)\cdot x$$

$$h_{px}(f,p,x) := h_{fp}(f,p)\cdot(1-x) + h_{gp}(f,p)\cdot x$$

$$s_{tx}(f,t,x) := s_{ft}(f,t)\cdot(1-x) + s_{gt}(f,t)\cdot x$$

$$s_{px}(f,p,x) := s_{fp}(f,p)\cdot(1-x) + s_{gp}(f,p)\cdot x$$

$$u_{tx}(f,t,x) := u_{ft}(f,t)\cdot(1-x) + u_{gt}(f,t)\cdot x$$

$$u_{px}(f,p,x) := u_{fp}(f,p)\cdot(1-x) + u_{gp}(f,p)\cdot x$$

$$\rho_{tx}(f,t,x) := \left(\rho_{ft}(f,t)^{-1}\cdot(1-x) + \rho_{gt}(f,t)^{-1}\cdot x\right)^{-1}$$

$$\rho_{px}(f,p,x) := \left(\rho_{fp}(f,p)^{-1}\cdot(1-x) + \rho_{gp}(f,p)^{-1}\cdot x\right)^{-1}$$

$$\Delta\rho_t(f,t) := \rho_{ft}(f,t) - \rho_{gt}(f,t)$$

$$\Delta\rho_p(f,p) := \rho_{fp}(f,p) - \rho_{gp}(f,p)$$

Derived Quantities

Morton Number

$$Mo(f,P) := g \cdot \mu_{fp}(f,P)^4 \cdot \frac{\Delta\rho_p(f,P)}{\rho_{fp}(f,P)^2 \cdot \sigma_t(f, T_{sat}(f,P))^3}$$

Dynamic Viscosity

$$\nu_{fp}(f,P) := \frac{\mu_{fp}(f,P)}{\rho_{fp}(f,P)} \qquad \nu_{gp}(f,P) := \frac{\mu_{gp}(f,P)}{\rho_{gp}(f,P)} \qquad \nu_{tp}(f,T,P) := \frac{\mu_{tp}(f,T,P)}{\rho_{tp}(f,T,P)}$$

$$\nu_{ft}(f,T) := \frac{\mu_{ft}(f,T)}{\rho_{ft}(f,T)} \qquad \nu_{gt}(f,T) := \frac{\mu_{gt}(f,T)}{\rho_{gt}(f,T)}$$

Prandtl Number

$$Pr(f,T,P) = \frac{\mu_{tp}(f,T,P) \cdot Cp_{tp}(f,T,P)}{k_{tp}(f,T,P)}$$

⇐ *This function and the next few are calculated in the add-in (C++ code) as of v. 2.1*

$$Pr_{fp}(f,P) := rp\_prfp\left(f, \frac{P}{MPa}\right) \qquad Pr_{gp}(f,P) := rp\_prgp\left(f, \frac{P}{MPa}\right) \qquad Pr_{tp}(f,T,P) := rp\_prtp\left(f, \frac{T}{K}, \frac{P}{MPa}\right)$$

$$Pr_{ft}(f,T) := rp\_prft\left(f, \frac{T}{K}\right) \qquad Pr_{gt}(f,T) := rp\_prgt\left(f, \frac{T}{K}\right)$$

Coefficient of Thermal Expansion

$$\beta_{fp}(f,P) := rp\_betafp\left(f, \frac{P}{MPa}\right) K^{-1} \qquad \beta_{gp}(f,P) := rp\_betagp\left(f, \frac{P}{MPa}\right) K^{-1} \qquad \beta_{tp}(f,T,P) := rp\_betatp\left(f, \frac{T}{K}, \frac{P}{MPa}\right) K^{-1}$$

$$\beta_{ft}(f,T) := rp\_betaft\left(f, \frac{T}{K}\right) K^{-1} \qquad \beta_{gt}(f,T) := rp\_betagt\left(f, \frac{T}{K}\right) K^{-1} \qquad \beta_c(fl,T,P) := \frac{-1}{\rho_{tp}(fl,T,P)} \cdot \left(\frac{d}{dT}\rho_{tp}(fl,T,P)\right)$$

Isentropic Exponent

$$\gamma_{fp}(f,P) := rp\_gammafp\left(f, \frac{P}{MPa}\right) \qquad \gamma_{gp}(f,P) := rp\_gammagp\left(f, \frac{P}{MPa}\right) \qquad \gamma_{tp}(f,T,P) := rp\_gammatp\left(f, \frac{T}{K}, \frac{P}{MPa}\right)$$

$$\gamma_{ft}(f,T) := rp\_gammaft\left(f, \frac{T}{K}\right) \qquad \gamma_{gt}(f,T) := rp\_gammagt\left(f, \frac{T}{K}\right) \qquad \gamma_c(fl,T,P) := \frac{Cp_{tp}(fl,T,P)}{Cv_{tp}(fl,T,P)}$$

Compressibility

$$Z_{tp}(fl,T,P) := rp\_ztp\left(fl, \frac{T}{K}, \frac{P}{MPa}\right) \qquad Z_{ft}(fl,T) := rp\_zft\left(fl, \frac{T}{K}\right) \qquad Z_{fp}(fl,P) := rp\_zfp\left(fl, \frac{P}{MPa}\right)$$

$$Z_c(fl, comp) = rp\_zcrit(fl, comp) \qquad Z_{gt}(fl,T) := rp\_zgt\left(fl, \frac{T}{K}\right) \qquad Z_{gp}(fl,P) := rp\_zgp\left(fl, \frac{P}{MPa}\right)$$

Reverse Functions of Pressure/Temperature and Enthalpy/Entropy

$$T_{ph}(fl,p,h) := \text{rp\_tph}\left(fl, \frac{p}{MPa}, h\cdot\frac{kg}{kJ}\right)\cdot K$$

$$T_{hs}(fl,h,s) := \text{rp\_ths}\left(fl, h\cdot\frac{kg}{kJ}, s\cdot\frac{kg\cdot K}{kJ}\right)\cdot K$$

$$T_{ps}(fl,p,s) := \text{rp\_tps}\left(fl, \frac{p}{MPa}, s\cdot\frac{kg\cdot K}{kJ}\right)\cdot K$$

$$P_{hs}(fl,h,s) := \text{rp\_phs}\left(fl, h\cdot\frac{kg}{kJ}, s\cdot\frac{kg\cdot K}{kJ}\right)\cdot MPa$$

$$P_{th}(fl,t,h,r) := \text{rp\_pth}\left(fl, \frac{t}{K}, h\cdot\frac{kg}{kJ}, r\right)\cdot MPa$$

$$P_{ts}(fl,t,s) := \text{rp\_pts}\left(fl, \frac{t}{K}, s\cdot\frac{kg\cdot K}{kJ}\right)\cdot MPa$$

$$\rho_{th}(fl,t,h,r) := \text{rp\_rhoth}\left(fl, \frac{t}{K}, h\cdot\frac{kg}{kJ}, r\right)\cdot\frac{kg}{m^3}$$

$$\rho_{ts}(fl,t,s) := \text{rp\_rhots}\left(fl, \frac{t}{K}, s\cdot\frac{kg\cdot K}{kJ}\right)\cdot\frac{kg}{m^3}$$

$$\rho_{ph}(fl,p,h) := \text{rp\_rhoph}\left(fl, \frac{p}{MPa}, h\cdot\frac{kg}{kJ}\right)\cdot\frac{kg}{m^3}$$

$$\rho_{ps}(fl,p,s) := \text{rp\_rhops}\left(fl, \frac{p}{MPa}, s\cdot\frac{kg\cdot K}{kJ}\right)\cdot\frac{kg}{m^3}$$

$$h_{ps}(fl,p,s) := \text{rp\_hps}\left(fl, \frac{p}{MPa}, s\cdot\frac{kg\cdot K}{kJ}\right)\cdot\frac{kJ}{kg}$$

$$s_{ph}(fl,p,h) := \text{rp\_sph}\left(fl, \frac{p}{MPa}, h\cdot\frac{kg}{kJ}\right)\cdot\frac{kJ}{kg\cdot K}$$

$$h_{ts}(fl,t,h) := \text{rp\_hts}\left(fl, \frac{t}{K}, h\cdot\frac{kg\cdot K}{kJ}\right)\cdot\frac{kJ}{kg}$$

$$s_{th}(fl,t,h,r) := \text{rp\_sth}\left(fl, \frac{t}{K}, h\cdot\frac{kg}{kJ}, r\right)\cdot\frac{kJ}{kg\cdot K}$$

Some Additional Unit Definitions

$kJ \equiv 1000\ J$ $\quad debye \equiv 1 \cdot 10^{-18}\ dyne^{0.5}\ cm^{2}$ $\quad \mu Pa \equiv 10^{-6}\ Pa$

These are set globally because they are needed in the above definitions.

Some Additional Utility Functions

Utility function that inputs a two-column matrix of fluids and mole fractions and outputs a formatted custom fluid string for the mixtures.

Use to create Legacy API mixture strings.

```
MixString(M) :=
    O ← ORIGIN
    n ← rows(M) − (1 − O)
    s ← ""
    y ← 0
    for i ∈ O .. n
        y ← y + M_{i, O+1}
        x ← concat("[", num2str(M_{i, O+1}), "]")
        s ← concat(s, M_{i, O}, x)
        if i < n
            s ← concat(s, "&")
    msg ← "Composition does not sum to unity."
    if |1 − y| < 0.000001
        s
    else
        error(msg)
```

Use to create High-Level API mixture strings.

```
MixString10(M) :=
    O ← ORIGIN
    n ← rows(M) − (1 − O)
    s ← ""
    y ← 0
    for i ∈ O .. n
        y ← y + M_{i, O+1}
        x ← concat(";", num2str(M_{i, O+1}))
        s ← concat(s, M_{i, O}, x)
        if i < n
            s ← concat(s, ";")
    msg ← "Composition does not sum to unity."
    if |1 − y| < 0.000001
        s
    else
        error(msg)
```

Usage:

$$C\_test := \begin{bmatrix} \text{"Nitrogen"} & 0.4 \\ \text{"Oxygen"} & 0.3 \\ \text{"Argon"} & 0.3 \end{bmatrix}$$

$\text{MixString}(C\_test) = \text{"Nitrogen[0.4]\&Oxygen[0.3]\&Argon[0.3]"}$ Legacy API strings

$\text{MixString10}(C\_test) = \text{"Nitrogen;0.4;Oxygen;0.3;Argon;0.3"}$ High-Level API strings

Utility function that takes a mixture file name, and returns a two column matrix consisting of the individual component names and the mole fractions for each.

```
Composition(fluid) :=
    mf ← rp_getx(fluid)
    for i ∈ ORIGIN .. last(mf)
        cn_i ← rp_getname(fluid, i + (1 − ORIGIN))
    return augment(cn, mf)
```

REFPROP multi-purpose wrapper function (only available in REFPROP10)

Utility functions to split strings, process units, and compress Output arrays are in the collapsed areas below.

```
REFPROP(hFld, hIn, hOut, a, b) :=
  if rp_getRPnum(0) ≥ 10
    hFld ← if(hFld = " ", "", hFld)
    vOut ← StrSplit(hOut)
    nOut ← MultiOut(vOut)
    "//*****Call the correct REFPROP function*****//"
    if (rp_IsChar(hOut) = 1)
      if nOut = 1
        try
          Ret ← rp_REFPROPc(hFld, hIn, hOut, a, b)
        on error
          error(rp_ERRMSG(0))
        return Ret
      else
        error("Only one string result can be retrieved at a time.")
    else
      inputs ← iCount(hIn)
      if inputs > 0
        a ← rp_scale(substr(hIn, ORIGINc, 1), a)
        if inputs > 1
          b ← rp_scale(substr(hIn, ORIGINc + 1, 1), b)
      if (nOut = 1) ∧ (hFld = "")
        try
          Ret ← rp_REFPROP1(hIn, hOut, a, b)
        on error
          error(rp_ERRMSG(0))
      else
        try
          Ret ← rp_REFPROP(hFld, hIn, hOut, a, b)
        on error
          error(rp_ERRMSG(0))
      "//**Add Units to return values and compress**//"
      iUlist ← rp_UNITS(hOut)
      Ret ← Ret · rp_Uval(iUlist)
      Ret ← rp_Comp(vOut, Ret)
    "//***********Return the result(s)***********//"
    if IsArray(Ret) ∧ (length(Ret) = 1)
      return Ret_ORIGIN
    else
      return Ret
  else
    error("ERROR: Function requires REFPROP 10 or later.")
```

Appendix D - RefProp Speed Up

Minimize NIST Fluid File Requests

Each RefProp call passes the name of the desired fluid to the underlying NIST DLL code. The NIST DLL then opens the appropriate fluid property file and extracts the needed physical constants and coefficients to properly calculate the physical properties for the requested fluid.

The NIST code is written in such a way, however, that it checks to see if the requested fluid file is already open. If it is, then no file I/O is necessary and the property calculations proceed directly. That is to say, 20 property calls for the same fluid, say "CO2.fld", only results in a single property file I/O request during the first property call.

Users can enhance the efficiency of the RefProp Add-In by avoiding repetitive calls to alternating fluids. For example, if properties for CO2 and Water are desired within a programming loop, it would be more efficient to make all of the property calls for Water, followed by all the property calls for CO2; resulting in just two property file I/O calls per programming loop.

Alternatively, making twenty property calls in matched pairs (one call for each fluid) would result in 20 times that many fluid property file I/O requests (40) per programming loop and would be ill advised.

Alternatively, it is possible to load multiple fluids at once (like an ad-hoc mixture) and then set the "Pure Fluid" REFPROP Flag to the component number for which pure-fluid calculations should be performed.

High-Level vs. Legacy API Calls

High-Level API calls require Fortran string handling, which is slow. If making single, or a dozen, property calls, wrapper function performance will not be noticeable. Call timing will be on the order of fractions of a millisecond. However, if many thousands of calls are being made, say for generating plots or optimization of thermodynamic cycles:

1. Make calls directly to the *rp_REFPROP* function, without units applied (40% improvement over calling the REFPROP user function).

2. If requesting single property requests, call *rp_REFPROP1* directly (80% improvement over calling the REFPROP user function).

3. For ~5X to 10X performance increase over using the REFPROP unit wrapper function, call the Legacy API functions with or without the unit wrapper functions.