Computer Subroutines for the Calculation of the Properties of Steam and Water

Paul Bannister
Energy Research Centre
Research School of Physical Sciences and Engineering
Australian National University
Canberra, Australia

March 1991

Report Number EP-RR-56

Computer Subroutines for the Calculation of the Properties of Steam and Water

Introduction

In any calculations related to thermal systems, such as the ANU solar thermal power system, accurate knowledge of the thermodynamic and transport properties of the working fluid is essential. This report documents a set of externally callable subroutines for the thermodynamic and transport properties of steam and water. The source code is written in VAX Pascal, enabling integration with applications written in VAX Pascal and Fortran. A Turbo Pascal version is provided. Also included within this report are some routines for certain thermodynamic processes. Sample programs are provided to illustrate the operation of the module (Appendix 1)

The subroutines for the properties of steam and water are based on a simple linear interpolation between given data points read from the data file THERMO.DAT. In the case of points near the saturation line, the interpolation is carried out taking into account the location of the saturation lines and the relevant property values. The accuracy of the property values returned is 1% or (generally) better, and the properties are fully continuous functions of pressure and temperature in the subcooled and superheated regions. In the saturated steam region (including the saturation lines) the properties are continuous functions of vapour quality and pressure/temperature.

Throughout the program the unit convention is generally SI, with the exception that pressure is in bar, specific energy in kJ/kg and specific entropy in kJ/kgK. The detailed internal operations of the program are documented within the code (Appendix 2).

Thermodynamic and Transport Properties of Steam

Each of the thermodynamic property subroutines is a global function contained within the module PROP.PAS. The functions available are detailed in Table 1.

Thermodynamic Processes

The subroutines for thermodynamic process calculations are also contained in the module PROP.PAS. The subroutines are as follows:

1. isobaric_heating(p,q,T,x). This procedure models the effect of adding heat q kJkg^{-1} (where q can be greater or less than zero) to fluid at pressure p bar, temperature T°C, and vapour quality x. The resultant state is returned as new values of T and x.
2. T_from_S_and_p(s,p,T,x). This procedure calculates the temperature T and vapour quality x corresponding to the entropy s and pressure p.
3. T_from_h_and_p(h,p,T,x). This procedure calculates the temperature T and vapour quality x corresponding to the enthalpy h and pressure p.
4. T_from_u_and_p(u,p,T,x). This procedure calculates the temperature T and vapour quality x corresponding to the internal energy u and pressure p.
5. find_sat_state_from_entropy(s,p,T). This procedure finds the intersection of the isentrope s with the $x = 1$ saturation line and finds the pressure p and temperature T at this point.

Initialisation of the Module

The VAX Pascal version of PROP.PAS should be initialised by calling the procedure read_in_data(var filename:text) with filename = THERMO at the beginning of the application program. Failure to do so causes the module to return an error message to the screen and, ultimately, it will fail. An example of a typical VAX Pascal program using the thermodynamic functions is given in Appendix 1.

The Turbo Pascal version of PROP.PAS is self initialising and does not require any external call to the procedure read_in_data. It is worth noting that the program requires hardware numeric evaluation and, under certain circumstances, also requires that the stack size be expanded to 24000. Both of these settings are found in the Option/Compiler menu. An example of a typical Turbo Pascal program using the thermodynamic functions is given in Appendix 1.

Summary

The module PROP.PAS (detailed in Appendix 2) contains a wide range of thermodynamic property and process subroutines that should be of considerable aid to other workers in the field of energy systems involving steam.

Property	Function Call	Pressure Range (bar)	Temperature Range (°C)
enthalpy ($kJkg^{-1}$)	enthalpy(p, T, x)	0.1–1 1–100	25–420 15–1000
entropy ($kJkg^{-1}K^{-1}$)	entropy(p, T, x)	0.1–1 1–100	25–420 15–1000
exergy ($kJkg^{-1}$)	exergy(p, T, n)	0.1–1 1–100	25–420 15–1000
internal energy ($kJkg^{-1}$)	internal_energy(p, T, x)	0.1–1 1–100	25–420 25–1000
density (kgm^{-3})	density(p, T, x)	0.1–1 1–100	25–420 15–1000
viscosity ($kgm^{-1}\ s^{-1}$)	viscosity(p, T, x)	1–100	25–800
thermal conductivity ($WK^{-1}\ m^{-1}$)	conductivity(p, T, x)	1–100	25–800
Prandtl number	Prandtl(p, T, x)	1–100	25–800
saturation temperature (°C)	saturation_temperature(p)	0.1–100	45.82 – 311.0
saturation pressure (bar)	saturation_pressure(T)	0.1–100	45.82–311.0
latent heat ($kJkg^{-1}$)	latent_heat(p)	0.1–100	n/a
surface tension	surface_tension(T)	n/a	all water states

Table 1: Thermodynamic and transport properties of steam available in the module PROP.PAS. All returned function values and argument values are real numbers. The argument variables are pressure (p) in bar, temperature (T) in degrees C, and vapour quality (x).

Appendix 1 : Program EXAMPLE.PAS VAX Pascal

```
{This is an example of a typical VAX Pascal program setup using some of the
routines described in the documentation.}

program example(input,output,outfile,thermo);

{Note that the data file THERMO must be declared in the first line so that the
program recognises it.}

var p, T, x, h, dens, T_s  : real;
    thermo,  outfile : text;
    ch : char;
{The file THERMO must also be declared as a text file variable.}

{The external procedures are declared first, as follows. It is very important
to get an exact correspondence in the variable declarations between the
module and the main program, as this is NOT checked by the link process and
can cause subtle and baffling errors.}

[external]procedure read_in_data(var w : text);extern;
[external]function enthalpy(p, T, x : real):real;extern;
[external]function density(p, T, x : real):real;extern;
[external]function saturation_temperature(p : real):real;extern;

{If there are any internal procedures they are declared next.}

procedure greetings;
begin
 writeln('Welcome to the instant thermodynamic quantity program!');
end;

procedure partings;
begin
 writeln('Goodbye, hope you will call again!');
end;


{The main program is now started}

begin
  greetings;
  rewrite(outfile);
  read_in_data(thermo);
  {It is very important to initialise the module as otherwise the program will
   fail as soon as it is called.}
  repeat
    write(' Pressure (bar) : ');
    readln(p);
    write('Temperature (C) : ');
    readln(T);
    T_s:=saturation_temperature(p);
    if T=T_s then
     begin
      writeln('Pressure and Temperture indicate saturation');
      write(' Vapour Quality : ');
      readln(x);
     end
    else
     begin
      if T<T_s then x:=0 else x:=1;
      writeln('Vapour Quality has been set at ',x:2:1);
     end;
    h:=enthalpy(p, T, x);
    dens:=density(p, T, x);
    writeln('Enthalpy is ',h:6:2,' kJ/kg');
    writeln(' Density is ',dens:6:2,' kg/m3');
    writeln(outfile,p:6:2,' ',T:6:2,' ',x:6:4,' ',h:8:2,' ',dens:8:2);
    write('Again? (y/n)');
    readln(ch);
  until ch in ['n','N'];
 partings;
end.
```

Appendix 1 : Program EXAMPLE2.PAS Turbo Pascal

```
{This is the Turbo Pascal version of the same program.}

program example;
uses property_program;
{This indicates the use of the external procedures from PROPERTY.PAS}

var p, T, x, h, dens, T_s  : real;
    thermo,  outfile : text;
    ch : char;

procedure greetings;
begin
 writeln('Welcome to the instant thermodynamic quantity program!');
end;

procedure partings;
begin
 writeln('Goodbye, hope you will call again!');
end;


{The main program is now started}

begin
  greetings;
  assign(outfile,'outfile.dat');
  rewrite(outfile);
  repeat
    write(' Pressure (bar) : ');
    readln(p);
    write('Temperature (C) : ');
    readln(T);
    T_s:=saturation_temperature(p);
    if T=T_s then
     begin
      writeln('Pressure and Temperture indicate saturation');
      write(' Vapour Quality : ');
      readln(x);
     end
    else
     begin
      if T<T_s then x:=0 else x:=1;
      writeln('Vapour Quality has been set at ',x:2:1);
     end;
    h:=enthalpy(p, T, x);
    dens:=density(p, T, x);
    writeln('Enthalpy is ',h:6:2,' kJ/kg');
    writeln(' Density is ',dens:6:2,' kg/m3');
    writeln(outfile,p:6:2,' ',T:6:2,' ',x:6:4,' ',h:8:2,' ',dens:8:2);
    write('Again? (y/n)');
    readln(ch);
 until ch in ['n','N'];
 partings;
 flush(outfile);
end.
```

Appendix 2 : Unit PROPERTY.PAS
Turbo Pascal

```
unit property_program;

{This module contains global procedures  designed to calculate, by
interpolation, the values of the thermodynamic functions enthalpy, entropy,
conductivity, Prandl Number, viscosity and density. Units of values returned
are unmodified S.I. units, except for energies, whiach are in kJ eg,kJ/kg,
kW/Km etc. The ordinate functions temperature and pressure are read in Celcius
and bar.

Module Organisation

To as large an extent as possible, the module has been organised into a series
of levels, with level zero being the highest, level one containing procedures
and functions used only by level zero, level two containing those used by
either level one or level zero, and so on. Variables with a common function
or meaning throughout these different levels have been suffixed with the level
number appropriate to the routine, which allows easier identification.

Algorithm

This program uses linear interpolation between data points read in from the
file THERMO.DAT. The bulk of the data is in the form of an array of values
corresponding to evaluation of the property at a grid of temperature and
pressure values. These grid values can be different for each property, thus
allowing the optimal choice of points to be made for each property. The data
includes values for both liquid and vapour phases
as well as the saturation values for both zero and 100% vapour quality. The
values for points sufficiently far away from saturation are found by simply
interpolating from the grid points that form a box around the point. Values
close to saturation are found by interpolating between the saturation line
and the gridlines.

Data Organisation

The properties to be evaluated  are stored as records of type 'property',
consisting of records of the grid temperatures and  pressures at which the
property is evaluated, and a two dimensional array of values 'table_value' that
holds the evaluations of the property at these grid points.

Temperature and pressure are stored as subrecords of type 'ordinate',
consisting of an array 'evalpts' of real values that holds the actual
evaluation points of the ordinate, and an integer,'N_evalpts', less than the
constant Max_N, giving the number of evaluation points for that axis.


Data Entry Format

The data should be entered into file THERMO.DAT in the
following fashion. Explanitory comments in square brackets [] shown here
should NOT be included in the actual data file. Care should be taken to
make sure that the units of the data are appropriate for the output procedures.
The program is currently set to read values as follows:temperatures:Celcius,
pressures:bar, enthalpy:kJ/kg, entropy:kJ/kgK, density:kg/m3,
conductivity:1e-3W/Km, viscosity 1e-6*kg/sm.

[beginning of file]
5 [=saturation.Nevalpts]

2 11 [=saturation.pressure[1],satuation.temperature[1]]
3 12 [                "              [2],          "          [3]]
4 13 [etc]
5 14
6 15

[Each property is now entered in the following fashion]
3 [=property.pressure.N_evalpts]

2 4 6  [=property.pressure.evalpt[1], [2], [3] ]

3 [=property.temperature.N_evalpts]

10 15 20 [=property.temperature.evalpt[1-3] ]

[The next set of data contains information on the saturation conditions, listed
individually for each property.pressure[i]. The order of information on each
line is: property.sat_val[lo], property.sat_val[hi]]
110 150
120 160
130 170

[The next array contains the grid of property values, with each line being the
series of property values associated with one pressure evaluation. Thus the
first row contains a list of property values at pressure =
property.pressure.evalpt[1], with the entries listed across the page in the
same order as the property.temperature.evalpts.]
100 200 300

110 210 310

120 220 320

[The other properties are listed similarly]
        .
        .
        .
[end of datafile]

Program Structure Illustration

level:[global]    enthalpy,entropy,Prandtl,viscosity,
                  conductivity,density,exergy
                          \    /
level:0                   value----------------------.
                        __/   \___                    \
level:1      saturation_          standard_property    error_check
             property                 |        \
               |                      |         \
level:2        |                      |---------- property_at_grid_T
               |          ____________/|           |          |
               |         /             |           |          |
level:3   saturation_limit  saturation_temperature saturation_pressure
                                 [global]              [global]

level:4   interpolation, find_brackets (these routines are used by most of the
          above)

Constants

Max_N : the maximum number of ordinate evaluation points.

dead_T, dead_p : the temperature and pressure of the 'dead' state, i.e. normal
         atmospheric conditions.

Variables

temperature : a record consisting of an array, temperature.evalpt[i] that
              lists, in ascending order, the temperatures at which the functions
              are given (in the data file THERMO.DAT) and an +ve integer (less
              than the constant Max_N) giving the number of points in the array.

pressure    : as for temperature, but this time using the pressure values at
              which the functions are given.
```

enthalpyprop: a record consisting of a two dimensional array enthalpy.table_value that contains the values of the entahlpy function at the grid temperatures and pressures given in *.evalpt[i], and another two dimensional array, containing the enthalpy values at saturation, indexed by pressure in the first index, and by vapour quality in the second (lo=>x=0, hi=>x=1). Values in both arrays are read from the input data file THERMO.DAT. Also included are records enthalpyprop.temperature and enthalpyprop.pressure containing the temperature and pressure evaluation points used for enthalpy.

entropyprop, densityprop, conductprop, viscosprop, Prandtlprop : as for enthalpyprop, but this time for the functions entropy, density conductivity, viscosity and Prandtl.

inconsistency: boolean variable used to keep tabs on inconsistenies in the pressure, temperature and vapour quality inputs to the module. Specifically, if the temperature is not the saturation temperature but the vapour quality is not zero or one, then inconsistency is set true. Note that under such conditions the pressure and temperature data overrides the vapour quality data.

x_error : boolean variable set to true if the temperature indicates saturation but the vapour quality is less than zero or greater than one. Under these conditions, the property value returned is the error default value, -9999.

t_error : boolean variable set by procedure find_brackets if the input temperature lies outside the range of the data in THERMO.DAT. Setting of this variable causes the property value returned to be -9999.

p_error : as for t_error, but for the input pressure information.

saturation : a record containing two one dimensional arrays listing a series of saturation pressures and temperatures for determination of the saturation line position, and the single integer N_evalpts giving the number of evaluation points used. Note that that through this record, with values from THERMO.DAT, the saturation line is determined for the entire module, irrespective of the choice of which property is being evaluated and how many pressure evaluations are associated with that property.

p_ind_* : a two component vector containing the indice numbers of the pressure grid points on either side of the input pressure point. The indice of the grid point that is higher than p is given by p_ind_*[hi], and the lower point by p_ind_*[lo].

t_ind_* : as for t_ind_*, but this time for the the temperature grid points bracketing the input pressure point.

whichprop_* : variable determining which of the properties enthalpyprop and entropyprop is being looked up.

whichord_* : variable determining which of the ordinates for the grid, i.e. pressure or temperature, is under consideration.

whichval_* : variable determining whether the saturation limit of interest is x=0 or x=1.

p, p_* : the input pressure value at which the property has to be evaluated.

t, t_* : the input temperature at which the property has to be evaluated.

x, x_* : the input vapour quality at which the property has to be evaluated.

```
{*****************************************************************************
                                DECLARATIONS
*****************************************************************************}
interface
function enthalpy(p, T, x : single): single;
function entropy(p, T, x : single): single;
function density(p, T, x : single):single;
function conductivity(p, T, x : single): single;
function viscosity(p, T, x : single): single;
function Prandtl(p, T, x : single): single;
function volume(p, T, x : single): single;
function exergy(p, T, x : single): single;
function surface_tension(T : single): single;
function int_energy(p, T, x : single): single;
function saturation_temperature(p_3 : single): single;
function saturation_pressure(T_3 : single): single;
function latent_heat(p : single): single;
procedure isobaric_heating(p, heatadded : single; var T, x : single);
procedure T_from_S_and_p(S, p : single; var T, x : single);
procedure T_from_h_and_p(h, p : single; var T, x : single);
procedure T_from_u_and_p(u, p : single; var T, x : single);
procedure find_sat_state_from_entropy(s : single; var p, T : single);

implementation

const Max_N = 30;
      dead_T= 298.15;
      dead_p= 1.0;

type    brackets = (lo, hi);

        ordinate_index = 1..Max_N;

        ordinate_vector = array[ordinate_index] of single;
        ordinate = record
                     evalpt : ordinate_vector;
                     N_evalpts : ordinate_index;
                   end;

        table_of_values = array[ordinate_index, ordinate_index] of single;

        property = record
                     tag : 1..11;
                     temperature, pressure : ordinate;
                     table_value : table_of_values;
                     sat_val : array[ordinate_index, brackets] of single;
                   end;

        index_bracket = array[brackets] of ordinate_index;

        real_bracket  = array[brackets] of single;

        double_brackets = array[brackets, brackets] of single;


var     saturation          : record
                                temperature, pressure : ordinate_vector;
                                N_evalpts : ordinate_index;
                              end;
```

```
          enthalpyprop, entropyprop, viscosprop, Prandtlprop, conductprop,
          densityprop, lo_enthalpyprop, lo_entropyprop, lo_densityprop,
          int_energyprop, lo_int_energyprop : property;

          inconsistency,
          p_error, t_error,
          x_error : boolean;

          saturation_T, saturation_p    : ordinate_vector;

          thermo                  : text;

          dead_state_availability, data_check : single;

{@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                   LEVEL 4
{@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

          {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

function interpolation(x1, x2, y1, y2, xx : single) : single;

{Performs a linear interpolation}

begin
 interpolation:=(xx-x1)*(y2-y1)/(x2-x1)+y1;
end;

          {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure find_brackets(vector : ordinate_vector; top_index : ordinate_index;
                        point : single;
                        var B_2 : index_bracket; var out_of_range : boolean);

{Finds the grid points bracketing the ordinate value 'point' of interest. If
the value lies outside the range of the table of values read in from THERMO.DAT
then the error flag 'out_of_range' is set to true.}

var newpt : ordinate_index;

begin
   if (point>vector[top_index]) or (point<vector[1]) then
      begin
        out_of_range:=true;
        B_2[hi]:=1;
        B_2[lo]:=1;
      end
     else
      begin
        B_2[hi]:=top_index;
        B_2[lo]:=1;
        repeat
            newpt:=(B_2[hi]+B_2[lo]) div 2;
            if (point-vector[newpt])<0 then B_2[hi]:= newpt
                                        else B_2[lo]:= newpt;
          until (B_2[hi]-B_2[lo])=1;
      end;
end;

{@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                   LEVEL 3
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

function saturation_temperature;

var p_ind_3 : index_bracket;
    error_3 : boolean;

{Interpolates between the known saturation temperatures at grid pressures to
find the saturation temperature at any pressure.}

begin
 error_3:=false;
 with saturation do
  begin
     find_brackets(pressure, N_evalpts, p_3, p_ind_3, error_3);
     if error_3 then saturation_temperature:=-1
              else saturation_temperature:=interpolation(pressure[p_ind_3[lo]],
                                 pressure[p_ind_3[hi]], temperature[p_ind_3[lo]],
                                 temperature[p_ind_3[hi]], p_3);
  end;
end;

          {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

function saturation_pressure;

{Interpolates between known saturation temperature values to find the
saturation pressure at any temperature.}

var p_ind_3  : index_bracket;
    error_3  : boolean;

begin
  error_3:=false;
  with saturation do
    begin
      find_brackets(temperature, N_evalpts, T_3, p_ind_3, error_3);
      if error_3 then saturation_pressure:=-1
             else saturation_pressure:=interpolation(temperature[p_ind_3[lo]],
                     temperature[p_ind_3[hi]], pressure[p_ind_3[lo]],
                     pressure[p_ind_3[hi]], T_3);
    end;
end;

          {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

function saturation_limit(whichprop_3 : property; whichval_3 : brackets;
                          p_ind_3 : index_bracket; p_3 : single): single;

{Interpolates between known values at grid pressures to find the x=0 or
x=1 limit of the property concerned at saturation.}

begin
 with whichprop_3.pressure do
   saturation_limit:=interpolation(evalpt[p_ind_3[lo]], evalpt[p_ind_3[hi]],
        whichprop_3.sat_val[p_ind_3[lo],whichval_3],
        whichprop_3.sat_val[p_ind_3[hi],whichval_3], p_3);
end;

{@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                   LEVEL 2
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}
```

```
function property_at_grid_T(whichprop_2 : property; T_index : ordinate_index;
        p_ind_2 : index_bracket; sat_T_2, p_2 : single): single;

{Interpolates between known points, (including saturation values when the
saturation line crosses the temperature grid line between the grid pressures
that immediately bracket the pressure point of interest) to find the value
of the property at a grid temperature, indexed by T_index.}

var grid_T, sat_p : single;
    whichside : brackets;

begin
 with whichprop_2 do
   begin
    grid_T:=temperature.evalpt[T_index];
    sat_p:=saturation_pressure(grid_T);

    {First we determine whichside of the saturation line we are on.}

    if sat_T_2>grid_T then whichside:=lo else whichside:=hi;

    {Depending on where the saturation line crosses the grid temperature line
    we now choose appropriate points,and intrpolate between them to find the
    property value.}

    with pressure do
      if (sat_p>=evalpt[p_ind_2[lo]]) and (sat_p<=p_2) then
         property_at_grid_T:=interpolation(sat_p, evalpt[p_ind_2[hi]],
                    saturation_limit(whichprop_2, whichside, p_ind_2, sat_p),
                    table_value[p_ind_2[hi], T_index], p_2)

       else if (sat_p<=evalpt[p_ind_2[hi]]) and (sat_p>=p_2) then
         property_at_grid_T:=interpolation(evalpt[p_ind_2[lo]], sat_p,
                    table_value[p_ind_2[lo], T_index],
                    saturation_limit(whichprop_2, whichside, p_ind_2, sat_p),
                    p_2)

       else property_at_grid_T:=interpolation(evalpt[p_ind_2[lo]],
                    evalpt[p_ind_2[hi]], table_value[p_ind_2[lo], T_index],
                    table_value[p_ind_2[hi], T_index], p_2);
   end;
end;


{@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                              LEVEL 1
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}


function standard_property(whichprop_1 : property; p_ind_1 : index_bracket;
                    sat_T_1, p_1, T_1 : single): single;

{Organises the evaluation of the property under superheated or sub cooled
conditions. The property value is found by interpolating along a constant
pressure line, between the smallest bracket that can be formed by the grid
pressures and the saturation line.}

var T_ind_1 : index_bracket;

begin
 with whichprop_1.temperature do
  begin
   find_brackets(evalpt, N_evalpts, T_1, T_ind_1, t_error);
   if T_1>evalpt[N_evalpts] then
     standard_property:=interpolation(evalpt[N_evalpts-1],
                    evalpt[N_evalpts], property_at_grid_T(whichprop_1,
                    N_evalpts-1, p_ind_1, sat_T_1, p_1),
                    property_at_grid_T(whichprop_1, N_evalpts, p_ind_1,
                    sat_T_1, p_1), T_1)
   else if (sat_T_1<=T_1) and (sat_T_1>=evalpt[T_ind_1[lo]]) then
        standard_property:=interpolation(sat_T_1, evalpt[T_ind_1[hi]],
                    saturation_limit(whichprop_1, hi, p_ind_1, p_1),
                    property_at_grid_T(whichprop_1, T_ind_1[hi],
                    p_ind_1, sat_T_1, p_1), T_1)

      else if (sat_T_1>=T_1) and (sat_T_1<=evalpt[T_ind_1[hi]]) then
       standard_property:=interpolation(evalpt[T_ind_1[lo]], sat_T_1,
                    property_at_grid_T(whichprop_1, T_ind_1[lo],
                    p_ind_1, sat_T_1, p_1),
                    saturation_limit(whichprop_1, lo, p_ind_1, p_1), T_1)

      else standard_property:=interpolation(evalpt[T_ind_1[lo]],
                    evalpt[T_ind_1[hi]],
                    property_at_grid_T(whichprop_1, T_ind_1[lo],
                    p_ind_1, sat_T_1, p_1), property_at_grid_T(whichprop_1,
                    T_ind_1[hi], p_ind_1, sat_T_1, p_1), T_1);
  end;
end;

     {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}


function saturation_property(whichprop_1 : property; p_ind_1 : index_bracket;
                    p_1, x_1 : single): single;

{Evaluates the property under saturation conditions.}

begin
 if (x_1>=0) and (x_1<=1) then
   saturation_property:=saturation_limit(whichprop_1, lo, p_ind_1, p_1)*(1-x_1)
                 +saturation_limit(whichprop_1, hi, p_ind_1, p_1)*x_1
  else
   begin
     saturation_property:=-9999;
     x_error:=true;
   end;
end;

     {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure check_for_bad_input_data(whichprop_1 : property; var p_1, T_1 : single);

var hold_p, hold_T : single;

begin
  with whichprop_1.pressure do
    begin
      hold_p:=p_1;
      if hold_p>evalpt[N_evalpts] then p_1:=evalpt[N_evalpts]
       else if hold_p<evalpt[1] then p_1:=evalpt[1];
    end;
  if hold_p<>p_1 then
    begin
      write('Pressure ',hold_p:4:2,' is out of range.');
      writeln('Proceeding with p = ',p_1:6:2);
    end;
  with whichprop_1.temperature do
    begin
      hold_T:=T_1;
      if hold_T>evalpt[N_evalpts] then T_1:=evalpt[N_evalpts]
       else if hold_T<evalpt[1] then T_1:=evalpt[1];
    end;
```

```
    if hold_T<>T_1 then
      begin
        write('Temperature ',hold_T:4:2,' is out of range.');
        writeln('Proceeding with T = ',T_1:6:2);
      end;
  end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure error_check(p_1, T_1, x_1 : single);

begin
 if p_error then writeln('Error in finding bracket for pressure value',p_1);
 if t_error then writeln('Error in finding bracket for temperature value',T_1);
 if x_error then writeln('Error in vapour quality value',x_1);
 if inconsistency then writeln('Data is inconsistent: x= ',x_1,
                               ' but temperature does not indicate saturation');
end;

{@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                 LEVEL 0
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

function value(whichprop_0 : property; p_0, T_0, x_0 : single): single;

{Sorts out whether the input conditions are saturation conditions or not
and organises the evaluation of the property accordingly.}

var p_ind_0 : index_bracket;
    sat_T_0, pp_0, TT_0 : single;

begin
 p_error:=false;
 t_error:=false;
 x_error:=false;
 inconsistency:=false;
 pp_0:=p_0;
 TT_0:=T_0;
 sat_T_0:=saturation_temperature(p_0);
 check_for_bad_input_data(whichprop_0, pp_0, TT_0);
 with whichprop_0.pressure do
   begin
     find_brackets(evalpt, N_evalpts, pp_0, p_ind_0, p_error);
     if p_error  then value:=-9999
       else if (abs(TT_0-sat_T_0)/TT_0<0.001) or ((x_0>0) and (x_0<1)) then
         begin
           value:=saturation_property(whichprop_0, p_ind_0, pp_0, x_0);
           if (abs(TT_0-sat_T_0)/TT_0>0.001) and ((x_0>0) and (x_0<1)) then
               inconsistency:=true;
         end
       else value:=standard_property(whichprop_0, p_ind_0, sat_T_0, pp_0, TT_0);
   end;
 error_check(p_0, T_0, x_0);
end;

{@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                LEVEL [GLOBAL]
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

function enthalpy;
begin
if p>=1 then enthalpy:=value(enthalpyprop, p, T, x)
        else enthalpy:=value(lo_enthalpyprop, p, T, x);
```

```
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

function entropy;
begin
if p>=1 then entropy:=value(entropyprop, p, T, x)
        else entropy:=value(lo_entropyprop, p, T, x);
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

function Prandtl;
begin
Prandtl:=value(Prandtlprop, p, T, x);
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

function viscosity;
begin
viscosity:=value(viscosprop, p, T, x)*1e-6;
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

function conductivity;
begin
conductivity:=value(conductprop, p, T, x)*1e-3;
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

function density;
begin
if p>=1 then
  begin
    density:=value(densityprop, p, T, x);
    if (x<1) and (x>0) then
      density:=1/(x/value(densityprop,p,T,1)+(1-x)/value(densityprop,p,T,0));
  end
 else
  begin
    density:=value(lo_densityprop, p, T, x);
    if (x<1) and (x>0) then density:=1/
       (x/value(lo_densityprop,p,T,1)+(1-x)/value(lo_densityprop,p,T,0));
  end;
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

function volume;
begin
  if p>=1 then
    volume:=x/value(densityprop,p,T,1)+(1-x)/value(densityprop,p,T,0)
   else
    volume:=x/value(lo_densityprop,p,T,1)+(1-x)/value(lo_densityprop,p,T,0)
end;
        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

function exergy;
begin
if p>=0 then exergy:=value(enthalpyprop, p ,T ,x )-
                     dead_T*value(entropyprop, p, T, x)-dead_state_availability
        else exergy:=value(lo_enthalpyprop, p, T, x)-
                     dead_t*value(lo_entropyprop, p, T, x)-
```

```
                    dead_state_availability;
end;

      {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

function latent_heat;
var T_sat_gl : single;
begin
 T_sat_gl :=saturation_temperature(p);
 if p>=1 then latent_heat:=value(enthalpyprop, p, T_sat_gl, 1)
                -value(enthalpyprop, p, T_sat_gl, 0)
         else latent_heat:=value(lo_enthalpyprop, p, T_sat_gl, 1)
                -value(lo_enthalpyprop, p, T_sat_gl, 0);
end;

      {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

function surface_tension;
const T_star = 647.15;
      Big_B  = 0.2358;
      Lil_b  = -0.625;
      mu     = 1.256;
begin
 surface_tension:=Big_B*exp(mu*ln(1-T/T_star))*(1+Lil_b*(1-T/T_star));
end;

      {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

function int_energy;
begin
 if p>=1 then int_energy:=value(int_energyprop, p, T, x)
         else int_energy:=value(lo_int_energyprop, p, T, x);
end;

      {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    SEE ALSO LEVEL 3 FOR GLOBAL FUNCTIONS SATURATION_TEMPERATURE AND
                        SATURATION PRESSURE
      @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

{@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
      PROCEDURES FOR THE READING OF DATA AND SETUP OF THE MODULE
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure read_in_ordinate(var whichord_1 : ordinate; var location_1 : text);

{Reads in the data for an ordinate from a text file}

var index1 : ordinate_index;

begin
 with whichord_1 do
   begin
     readln(location_1, N_evalpts);
     readln(location_1);
     for index1:=1 to N_evalpts do read(location_1, evalpt[index1]);
     readln(location_1);
   end;
end;

      {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure read_in_property_table(var whichprop_1 : property;
                                 var location_1 : text);

{Reads in the table of property values at grid temperatures and pressures from
a text file.}

var index1, index2 : ordinate_index;

begin
 with whichprop_1 do
     for index1:=1 to pressure.N_evalpts do
       begin
         for index2:=1 to temperature.N_evalpts do
                read(location_1, table_value[index1, index2]);
         readln(location_1);
       end;
end;

      {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure read_in_sat_vals(var whichprop_1 : property; var location_1 : text);

{Reads in the table of saturation temperature and property values from a text
file.}

var index1 : ordinate_index;

begin
 with whichprop_1 do
   begin
     for index1:=1 to pressure.N_evalpts do
          readln(location_1, sat_val[index1,lo], sat_val[index1,hi]);
     readln(location_1);
   end;
end;

      {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure read_saturation_line(var location_0 : text);

var index1 : ordinate_index;

begin
  with saturation do
    begin
      readln(location_0, N_evalpts);
      readln(location_0);
      for index1:=1 to N_evalpts do
          readln(location_0, pressure[index1], temperature[index1] );
      readln(location_0);
    end;
end;

      {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure read_in_property(var whichprop_0 : property; var location_0 : text);
begin
 with whichprop_0 do
   begin
     read_in_ordinate(pressure, location_0);
     read_in_ordinate(temperature, location_0);
     read_in_sat_vals(whichprop_0, location_0);
     read_in_property_table(whichprop_0, location_0);
   end;
end;

      {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}
```

```
procedure read_in_data(var location : text);

{Reads in the data from a text file.}

begin
 reset(location);
 readln(location,data_check);
 readln(location);
 read_saturation_line(location);
 read_in_property(enthalpyprop, location);
 read_in_property(entropyprop, location);
 read_in_property(viscosprop, location);
 read_in_property(Prandtlprop, location);
 read_in_property(conductprop, location);
 read_in_property(densityprop, location);
 read_in_property(lo_enthalpyprop,location);
 read_in_property(lo_entropyprop, location);
 read_in_property(lo_densityprop, location);
 read_in_property(lo_int_energyprop, location);
 read_in_property(int_energyprop, location);
 enthalpyprop.tag:=1;
 entropyprop.tag:=2;
 densityprop.tag:=3;
 lo_enthalpyprop.tag:=4;
 lo_entropyprop.tag:=5;
 lo_densityprop.tag:=6;
 int_energyprop.tag:=7;
 lo_int_energyprop.tag:=8;
 viscosprop.tag:=9;
 Prandtlprop.tag:=10;
 conductprop.tag:=11;
 dead_state_availability:=value(enthalpyprop, dead_p, (dead_T-273.15), 0)-
                  dead_T*value(entropyprop, dead_p, (dead_T-273.15), 0);
 writeln('READING OF DATA FROM FILE THERMO.DAT COMPLETED');
end;

       {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure T_from_property_and_pressure(whichprop_0 : property;
                            p_0, propval : single;var T_0, x_0 : single);

var p_ind_0, B_0 : index_bracket;
    sat_T_0, hi_sat, lo_sat, lo_T_prop, hi_T_prop : single;
    newpt : ordinate_index;

begin
 with whichprop_0 do
  begin
   find_brackets(pressure.evalpt,pressure.N_evalpts, p_0, p_ind_0, p_error);
   hi_sat:=saturation_limit(whichprop_0, hi, p_ind_0, p_0);
   lo_sat:=saturation_limit(whichprop_0, lo, p_ind_0, p_0);
   sat_T_0:=saturation_temperature(p_0);
   if (lo_sat<=propval) and (hi_sat>=propval) then
     begin
      T_0:=sat_T_0;
      x_0:=(propval-lo_sat)/(hi_sat-lo_sat);
     end
    else
     with temperature do
      begin
       B_0[hi]:=N_evalpts;
       B_0[lo]:=1;

       repeat
          newpt:=(B_0[hi]+B_0[lo])div 2;
          if (propval-
                property_at_grid_T(whichprop_0, newpt, p_ind_0, sat_T_0, p_0)<0)
                then B_0[hi]:=newpt
                else B_0[lo]:=newpt;
       until (B_0[hi]-B_0[lo])=1;

       lo_T_prop:=property_at_grid_T(whichprop_0,B_0[lo],p_ind_0,sat_T_0, p_0);
       hi_T_prop:=property_at_grid_T(whichprop_0,B_0[hi],p_ind_0,sat_T_0, p_0);
       if (lo_sat>=propval) and (lo_sat<=hi_T_prop) then
         T_0:=interpolation(lo_T_prop,lo_sat, evalpt[B_0[lo]],sat_T_0, propval)

        else if (hi_sat<=propval) and (hi_sat>=lo_T_prop) then
         T_0:=interpolation(hi_sat, hi_T_prop,
              sat_T_0, evalpt[B_0[hi]], propval)
        else
         T_0:=interpolation(lo_T_prop, hi_T_prop, evalpt[B_0[lo]],
                                         evalpt[B_0[hi]], propval);
         if propval>hi_sat then x_0:=1 else x_0:=0;
      end;
    end;
 end;

       {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure isobaric_heating;
begin
 if p>=1 then T_from_property_and_pressure(enthalpyprop, p,
                 (value(enthalpyprop, p, T, x)+heatadded), T, x)
         else T_from_property_and_pressure(lo_enthalpyprop, p,
                 (value(lo_enthalpyprop, p, T, x)+heatadded), T, x);
end;

       {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure T_from_S_and_p;
begin
 if p>=1 then T_from_property_and_pressure(entropyprop, p, s, T, x)
        else T_from_property_and_pressure(lo_entropyprop, p, s, T, x);
end;

       {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure T_from_h_and_p;
begin
 if p>=1 then T_from_property_and_pressure(enthalpyprop, p, h, T, x)
        else T_from_property_and_pressure(lo_enthalpyprop, p, h, T, x)

end;
       {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure T_from_u_and_p;
begin
 if p>=1 then T_from_property_and_pressure(int_energyprop, p, u, T, x)
        else T_from_property_and_pressure(lo_int_energyprop, p, u, T, x)

end;

       {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure find_sat_state_from_S_and_x(propval_00, x_00 : single;
                                       var p_00, T_00 : single);
var B_00 : index_bracket;
    newpt : ordinate_index;
    loval_00, hival_00, testval_00 : single;
```

```
begin

 B_00[hi]:=saturation.N_evalpts;
 B_00[lo]:=1;

 repeat
  newpt:=(B_00[lo]+B_00[hi]) div 2;
  if saturation.pressure[newpt]<1 then
         testval_00:=value(lo_entropyprop, saturation.pressure[newpt],
                  saturation.temperature[newpt], x_00)
    else testval_00:=value(entropyprop, saturation.pressure[newpt],
                  saturation.temperature[newpt], x_00);
  if (propval_00-testval_00)>0 then B_00[hi]:=newpt
                               else B_00[lo]:=newpt;

 until (B_00[hi]-B_00[lo])=1;

  if saturation.pressure[B_00[lo]]<1 then
        loval_00:=value(lo_entropyprop, saturation.pressure[B_00[lo]],
                  saturation.temperature[B_00[lo]], x_00)
   else loval_00:=value(entropyprop, saturation.pressure[B_00[lo]],
                  saturation.temperature[B_00[lo]], x_00);

  if saturation.pressure[B_00[hi]]<1 then
        hival_00:=value(lo_entropyprop, saturation.pressure[B_00[hi]],
                  saturation.temperature[B_00[hi]], x_00)
   else hival_00:=value(entropyprop, saturation.pressure[B_00[hi]],
                  saturation.temperature[B_00[hi]], x_00);

 p_00:=interpolation(loval_00, hival_00, saturation.pressure[B_00[lo]],
                     saturation.pressure[B_00[hi]], propval_00);
 T_00:=saturation_temperature(p_00);
 if p_00>saturation.pressure[saturation.N_evalpts] then
   begin
     writeln('ERROR : Failed to correctly find value on saturation line');
     writeln('Entropy value ',propval_00);
   end;
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure find_sat_state_from_entropy;
begin
  find_sat_state_from_S_and_x(s,1,p,T);
end;

begin
 assign(thermo,'thermo.dat');
 read_in_data(thermo);
end.{of module}
```

Appendix 2 : Module PROP.PAS
VAX Pascal

module standard_functions(input,output,thermo);

{This module contains global procedures designed to calculate, by interpolation, the values of the thermodynamic functions enthalpy, entropy, conductivity, Prandl Number, viscosity, internal energy and density. Units of values returned are unmodified S.I. units, except for energies, which are in kJ eg kJ/kg, kW/Km etc. The ordinate functions temperature and pressure are read in Celcius and bar.

Module Organisation

To as large an extent as possible, the module has been organised into a series of levels, with level zero being the highest, level one containing procedures and functions used only by level zero, level two containing those used by either level one or level zero, and so on. Variables with a common function or meaning throughout these different levels have been suffixed with the level number appropriate to the routine, which allows easier identification.

Algorithm

This program uses linear interpolation between data points read in from the file THERMO.DAT. The bulk of the data is in the form of an array of values corresponding to evaluation of the property at a grid of temperature and pressure values. These grid values can be different for each property, thus allowing the optimal choice of points to be made for each property. The data includes values for both liquid and vapour phases
as well as the saturation values for both zero and 100% vapour quality. The values for points sufficiently far away from saturation are found by simply interpolating from the grid points that form a box around the point. Values close to saturation are found by interpolating between the saturation line and the gridlines.

Data Organisation

The properties to be evaluated are stored as records of type 'property', consisting of records of the grid temperatures and pressures at which the property is evaluated, and a two dimensional array of values 'table_value' that holds the evaluations of the property at these grid points.

Temperature and pressure are stored as subrecords of type 'ordinate', consisting of an array 'evalpts' of real values that holds the actual evaluation points of the ordinate, and an integer,'N_evalpts', less than the constant Max_N, giving the number of evaluation points for that axis.

Data Entry Format

The data should be entered into file THERMO.DAT in the following fashion. Explanitory comments in square brackets [] shown here should NOT be included in the actual data file. Care should be taken to make sure that the units of the data are appropriate for the output procedures. The program is currently set to read values as follows:temperatures:Celcius, pressures:bar, enthalpy:kJ/kg, entropy:kJ/kgK, density:kg/m3, conductivity:1e-3W/Km, viscosity 1e-6*kg/sm.

```
[beginning of file]
-999.99[This is a number inserted at the beginning of the data file ONLY to
       [indicate to the module that the datafile has been read]

5 [=saturation.Nevalpts]

2 11 [=saturation.pressure[1],satuation.temperature[1]]
3 12 [          "          [2],          "         [3]]
4 13 [etc]
5 14
6 15

[Each property is now entered in the following fashion]
3 [=property.pressure.N_evalpts]

2 4 6  [=property.pressure.evalpt[1], [2], [3] ]

3 [=property.temperature.N_evalpts]

10 15 20 [=property.temperature.evalpt[1-3] ]

[The next set of data contains information on the saturation conditions, listed
individually for each property.pressure[i]. The order of information on each
line is: property.sat_val[lo], property.sat_val[hi]]
110 150
120 160
130 170

[The next array contains the grid of property values, with each line being the
series of property values associated with one pressure evaluation. Thus the
first row contains a list of property values at pressure =
property.pressure.evalpt[1], with the entries listed across the page in the
same order as the property.temperature.evalpts.]
100 200 300

110 210 310

120 220 320

[The other properties are listed similarly]
         .
         .
         .
[end of datafile]
```

Program Structure Illustration

```
level:[global]    enthalpy,entropy,Prandtl,viscosity,
                  conductivity,density,exergy,int_energy
                           \    /
level:0                    value-----------------------.
                        ___/   \___                     \
level:1      saturation_           standard_property    error_check
             property                    |        \
                 |                       |         \
level:2          |                       |---------- property_at_grid_T
                 |                ______/|                 |
                 |               /       |                 |
level:3     saturation_limit  saturation_temperature saturation_pressure
                                   [global]                [global]

level:4     interpolation, find_brackets (these routines are used by most of the
            above)
```

Constants

Max_N : the maximum number of ordinate evaluation points.

dead_T, dead_p : the temperature and pressure of the 'dead' state, i.e. normal atmospheric conditions.

Variables

temperature : a record consisting of an array, temperature.evalpt[i] that lists, in ascending order, the temperatures at which the functions are given (in the data file THERMO.DAT) and an +ve integer (less than the constant Max_N) giving the number of points in the array.

pressure : as for temperature, but this time using the pressure values at which the functions are given.

enthalpyprop: a record consisting of a two dimensional array enthalpy.table_value that contains the values of the entahlpy function at the grid temperatures and pressures given in *.evalpt[i], and another two dimensional array, containing the enthalpy values at saturation, indexed by pressure in the first index, and by vapour quality in the second (lo=>x=0, hi=>x=1). Values in both arrays are read from the input data file THERMO.DAT. Also included are records enthalpyprop.temperature and enthalpyprop.pressure containing the temperature and pressure evaluation points used for enthalpy.

entropyprop, densityprop, conductprop, viscosprop, Prandtlprop, int_energyprop : as for enthalpyprop, but this time for the functions entropy, density, conductivity, viscosity, Prandtl number and internal energy.

lo_enthalpyprop, lo_densityprop, lo_entropyprop, lo_int_energyprop: these are standard property records for low pressures. They exist separately of the normal *prop records, but are called from the same global routine. This reduces the amount of time spent searching by the program and improves the flexibility of the data file, as the lower pressures normally require a different choice of temperature.evalpt[]'s to that which would be used otherwise.

inconsistency: boolean variable used to keep tabs on inconsistenies in the pressure, temperature and vapour quality inputs to the module. Specifically, if the temperature is not the saturation temperature but the vapour quality is not zero or one, then inconsistency is set true. Note that under such conditions the pressure and temperature data overrides the vapour quality data.

x_error : boolean variable set to true if the temperature indicates saturation but the vapour quality is less than zero or greater than one. Under these conditions, the property value returned is the error default value, -9999.

t_error : boolean variable set by procedure find_brackets if the input temperature lies outside the range of the data in THERMO.DAT. Setting of this variable causes the property value returned to be -9999.

p_error : as for t_error, but for the input pressure information.

saturation : a record containing two one dimensional arrays listing a series of saturation pressures and temperatures for determination of the saturation line position, and the single integer N_evalpts giving the number of evaluation points used. Note that that through this record, with values from THERMO.DAT, the saturation line is determined for the entire module, irrespective of the choice of which property is being evaluated and how many pressure evaluations are associated with that property.

p_ind_* : a two component vector containing the indice numbers of the pressure grid points on either side of the input pressure point. The indice of the grid point that is higher than p is given by p_ind_*[hi], and the lower point by p_ind_*[lo].

t_ind_* : as for t_ind_*, but this time for the the temperature grid points bracketing the input pressure point.

whichprop_* : variable determining which of the properties enthalpyprop and entropyprop is being looked up.

whichord_* : variable determining which of the ordinates for the grid, i.e. pressure or temperature, is under consideration.

whichval_* : variable determining whether the saturation limit of interest is x=0 or x=1.

p, p_* : the input pressure value at which the property has to be evaluated.

t, t_* : the input temperature at which the property has to be evaluated.

x, x_* : the input vapour quality at which the property has to be evaluated.

```
*******************************************************************************
                              DECLARATIONS
*******************************************************************************}

const Max_N = 30;
      dead_T= 298.15;
      dead_p= 1.0;

type    brackets = (lo, hi);

        ordinate_index = 1..Max_N;

        ordinate_vector = array[ordinate_index] of real;
        ordinate = record
                     evalpt : ordinate_vector;
                     N_evalpts : ordinate_index;
                   end;

        table_of_values = array[ordinate_index, ordinate_index] of real;

        property = record
                     tag : 1..10;
                     temperature, pressure : ordinate;
                     table_value : table_of_values;
                     sat_val : array[ordinate_index, brackets] of real;
                   end;

        index_bracket = array[brackets] of ordinate_index;

        real_bracket  = array[brackets] of real;

        double_brackets = array[brackets, brackets] of real;


var     saturation              : record
                                    temperature, pressure : ordinate_vector;
                                    N_evalpts : ordinate_index;
                                  end;

        enthalpyprop, entropyprop, viscosprop, Prandtlprop, conductprop,
        densityprop, lo_enthalpyprop, lo_entropyprop, lo_densityprop,
        int_energyprop, lo_int_energyprop : property;

        inconsistency,
        p_error, t_error,
        x_error : boolean;

        saturation_T, saturation_p    : ordinate_vector;
```

```
        thermo              : text;

        dead_state_availability, data_check : real;

{@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                   LEVEL 4
{@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure check_for_data;
{This procedure checks whether the file THERMO.DAT has been read. If it hasn't,
 a warning message is broadcast. The program will either fail or produce
 nonsensical answers if the file has not been read.}
begin
 if data_check<>-999.99 then
   begin
     writeln('       *********************************************');
     writeln('                      Module prop.pas');
     writeln('  FATAL ERROR : DATAFILE THERMO.DAT HAS NOT BEEN READ');
     writeln('To rectify, call procedure read_in_data at start of program');
     writeln('       *********************************************');
   end;
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

function interpolation(x1, x2, y1, y2, xx : real) : real;

{Performs a linear interpolation}

begin
 interpolation:=(xx-x1)*(y2-y1)/(x2-x1)+y1;
end;


        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}


procedure find_brackets(vector : ordinate_vector; top_index : ordinate_index;
                        point : real;
                        var B_2 : index_bracket; var out_of_range : boolean);

{Finds the grid points bracketing the ordinate value 'point' of interest. If
the value lies outside the range of the table of values read in from THERMO.DAT
then the error flag 'out_of_range' is set to true.}

var newpt : ordinate_index;

begin
   if (point>vector[top_index]) or (point<vector[1]) then
       begin
         out_of_range:=true;
         B_2[hi]:=1;
         B_2[lo]:=1;
       end
      else
       begin
         B_2[hi]:=top_index;
         B_2[lo]:=1;
         repeat
             newpt:=(B_2[hi]+B_2[lo]) div 2;
             if (point-vector[newpt])<0 then B_2[hi]:= newpt
                                         else B_2[lo]:= newpt;
           until (B_2[hi]-B_2[lo])=1;
       end;
end;

{@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                   LEVEL 3
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}


[global]function saturation_temperature(p_3 : real): real;

var p_ind_3 : index_bracket;
    error_3 : boolean;

{Interpolates between the known saturation temperatures at grid pressures to
find the saturation temperature at any pressure.}

begin
 check_for_data;
 error_3:=false;
 with saturation do
  begin
     find_brackets(pressure, N_evalpts, p_3, p_ind_3, error_3);
     if error_3 then saturation_temperature:=-1
              else saturation_temperature:=interpolation(pressure[p_ind_3[lo]],
                             pressure[p_ind_3[hi]], temperature[p_ind_3[lo]],
                             temperature[p_ind_3[hi]], p_3);
  end;
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}


[global]function saturation_pressure(T_3 : real): real;

{Interpolates between known saturation temperature values to find the
saturation pressure at any temperature.}

var p_ind_3  : index_bracket;
    error_3  : boolean;

begin
  check_for_data;
  error_3:=false;
  with saturation do
   begin
     find_brackets(temperature, N_evalpts, T_3, p_ind_3, error_3);
     if error_3 then saturation_pressure:=-1
            else saturation_pressure:=interpolation(temperature[p_ind_3[lo]],
                       temperature[p_ind_3[hi]], pressure[p_ind_3[lo]],
                       pressure[p_ind_3[hi]], T_3);
   end;
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}


function saturation_limit(whichprop_3 : property; whichval_3 : brackets;
                          p_ind_3 : index_bracket; p_3 : real): real;

{Interpolates between known values at grid pressures to find the x=0 or
x=1 limit of the property concerned at saturation.}

begin
 with whichprop_3.pressure do
   saturation_limit:=interpolation(evalpt[p_ind_3[lo]], evalpt[p_ind_3[hi]],
        whichprop_3.sat_val[p_ind_3[lo],whichval_3],
        whichprop_3.sat_val[p_ind_3[hi],whichval_3], p_3);
```

```
end;

{@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                              LEVEL 2
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

function property_at_grid_T(whichprop_2 : property; T_index : ordinate_index;
         p_ind_2 : index_bracket; sat_T_2, p_2 : real): real;

{Interpolates between known points, (including saturation values when the
saturation line crosses the temperature grid line between the grid pressures
that immediately bracket the pressure point of interest) to find the value
of the property at a grid temperature, indexed by T_index.}

var grid_T, sat_p : real;
    whichside : brackets;

begin
 with whichprop_2 do
   begin
     grid_T:=temperature.evalpt[T_index];
     sat_p:=saturation_pressure(grid_T);

     {First we determine whichside of the saturation line we are on.}

     if sat_T_2>grid_T then whichside:=lo else whichside:=hi;

     {Depending on where the saturation line crosses the grid temperature line
     we now choose appropriate points,and intrpolate between them to find the
     property value.}

     with pressure do
       if (sat_p>=evalpt[p_ind_2[lo]]) and (sat_p<=p_2) then
          property_at_grid_T:=interpolation(sat_p, evalpt[p_ind_2[hi]],
                       saturation_limit(whichprop_2, whichside, p_ind_2, sat_p),
                       table_value[p_ind_2[hi], T_index], p_2)

        else if (sat_p<=evalpt[p_ind_2[hi]]) and (sat_p>=p_2) then
          property_at_grid_T:=interpolation(evalpt[p_ind_2[lo]], sat_p,
                       table_value[p_ind_2[lo], T_index],
                       saturation_limit(whichprop_2, whichside, p_ind_2, sat_p),
                       p_2)

        else property_at_grid_T:=interpolation(evalpt[p_ind_2[lo]],
                       evalpt[p_ind_2[hi]], table_value[p_ind_2[lo], T_index],
                       table_value[p_ind_2[hi], T_index], p_2);
   end;
end;

{@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                              LEVEL 1
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}


function standard_property(whichprop_1 : property; p_ind_1 : index_bracket;
                           sat_T_1, p_1, T_1 : real): real;

{Organises the evaluation of the property under superheated or sub cooled
conditions. The property value is found by interpolating along a constant
pressure line, between the smallest bracket that can be formed by the grid
pressures and the saturation line.}

var T_ind_1 : index_bracket;

begin
 with whichprop_1.temperature do
  begin
   find_brackets(evalpt, N_evalpts, T_1, T_ind_1, t_error);
   if T_1>evalpt[N_evalpts] then
     standard_property:=interpolation(evalpt[N_evalpts-1],
                        evalpt[N_evalpts], property_at_grid_T(whichprop_1,
                        N_evalpts-1, p_ind_1, sat_T_1, p_1),
                        property_at_grid_T(whichprop_1, N_evalpts, p_ind_1,
                        sat_T_1, p_1), T_1)
    else if (sat_T_1<=T_1) and (sat_T_1>=evalpt[T_ind_1[lo]]) then
         standard_property:=interpolation(sat_T_1, evalpt[T_ind_1[hi]],
                        saturation_limit(whichprop_1, hi, p_ind_1, p_1),
                        property_at_grid_T(whichprop_1, T_ind_1[hi],
                        p_ind_1, sat_T_1, p_1), T_1)

        else if (sat_T_1>=T_1) and (sat_T_1<=evalpt[T_ind_1[hi]]) then
         standard_property:=interpolation(evalpt[T_ind_1[lo]], sat_T_1,
                        property_at_grid_T(whichprop_1, T_ind_1[lo],
                        p_ind_1, sat_T_1, p_1),
                        saturation_limit(whichprop_1, lo, p_ind_1, p_1), T_1)

        else standard_property:=interpolation(evalpt[T_ind_1[lo]],
                        evalpt[T_ind_1[hi]],
                        property_at_grid_T(whichprop_1, T_ind_1[lo],
                        p_ind_1, sat_T_1, p_1), property_at_grid_T(whichprop_1,
                        T_ind_1[hi], p_ind_1, sat_T_1, p_1), T_1);
  end;
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}


function saturation_property(whichprop_1 : property; p_ind_1 : index_bracket;
                             p_1, x_1 : real): real;

{Evaluates the property under saturation conditions.}

begin
 if (x_1>=0) and (x_1<=1) then
   saturation_property:=saturation_limit(whichprop_1, lo, p_ind_1, p_1)*(1-x_1)
                 +saturation_limit(whichprop_1, hi, p_ind_1, p_1)*x_1
  else
   begin
     saturation_property:=-9999;
     x_error:=true;
   end;
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure check_for_bad_input_data(whichprop_1 : property; var p_1, T_1 : real);

var hold_p, hold_T : real;

begin
  with whichprop_1.pressure do
    begin
      hold_p:=p_1;
      if hold_p>evalpt[N_evalpts] then p_1:=evalpt[N_evalpts]
       else if hold_p<evalpt[1] then p_1:=evalpt[1];
    end;
  if hold_p<>p_1 then
    begin
      write('Pressure ',hold_p:4:2,' is out of range.');
      writeln('Proceeding with p = ',p_1:6:2);
```

```
      end;
    with whichprop_1.temperature do
      begin
        hold_T:=T_1;
        if hold_T>evalpt[N_evalpts] then T_1:=evalpt[N_evalpts]
          else if hold_T<evalpt[1] then T_1:=evalpt[1];
      end;
    if hold_T<>T_1 then
      begin
        write('Temperature ',hold_T:4:2,' is out of range.');
        writeln('Proceeding with T = ',T_1:6:2);
      end;
  end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}


procedure error_check(p_1, T_1, x_1 : real);

begin
  if p_error then writeln('Error in finding bracket for pressure value',p_1);
  if t_error then writeln('Error in finding bracket for temperature value',T_1);
  if x_error then writeln('Error in vapour quality value',x_1);
  if inconsistency then writeln('Data is inconsistent: x= ',x_1,
                          ' but temperature does not indicate saturation');
end;

{@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                 LEVEL 0
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}


function value(whichprop_0 : property; p_0, T_0, x_0 : real): real;

{Sorts out whether the input conditions are saturation conditions or not
and organises the evaluation of the property accordingly.}

var p_ind_0 : index_bracket;
    sat_T_0, pp_0, TT_0 : real;

begin
check_for_data;
 p_error:=false;
 t_error:=false;
 x_error:=false;
 inconsistency:=false;
 pp_0:=p_0;
 TT_0:=T_0;
 sat_T_0:=saturation_temperature(p_0);
 check_for_bad_input_data(whichprop_0, pp_0, TT_0);
 with whichprop_0.pressure do
   begin
     find_brackets(evalpt, N_evalpts, pp_0, p_ind_0, p_error);
     if p_error  then value:=-9999
       else if (abs(TT_0-sat_T_0)/TT_0<0.001) or ((x_0>0) and (x_0<1)) then
         begin
           value:=saturation_property(whichprop_0, p_ind_0, pp_0, x_0);
           if (abs(TT_0-sat_T_0)/TT_0>0.001) and ((x_0>0) and (x_0<1)) then
              inconsistency:=true;
         end
       else value:=standard_property(whichprop_0, p_ind_0, sat_T_0, pp_0, TT_0);
   end;
 error_check(p_0, T_0, x_0);
end;

{@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                 LEVEL [GLOBAL]
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

[global] function enthalpy(p, T, x : real):real;
begin
if p>=1 then enthalpy:=value(enthalpyprop, p, T, x)
        else enthalpy:=value(lo_enthalpyprop, p, T, x);
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

[global] function entropy(p, T, x : real):real;
begin
if p>=1 then entropy:=value(entropyprop, p, T, x)
        else entropy:=value(lo_entropyprop, p, T, x);
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

[global] function Prandtl(p, T, x : real):real;
begin
Prandtl:=value(Prandtlprop, p, T, x);
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

[global] function viscosity(p, T, x : real):real;
begin
viscosity:=value(viscosprop, p, T, x)*1e-6;
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

[global] function conductivity(p, T, x : real):real;
begin
conductivity:=value(conductprop, p, T, x)*1e-3;
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

[global] function density(p, T, x : real):real;
begin
if p>=1 then
  begin
    density:=value(densityprop, p, T, x);
    if (x<1) and (x>0) then
      density:=1/(x/value(densityprop,p,T,1)+(1-x)/value(densityprop,p,T,0));
  end
 else
  begin
    density:=value(lo_densityprop, p, T, x);
    if (x<1) and (x>0) then density:=1/
        (x/value(lo_densityprop,p,T,1)+(1-x)/value(lo_densityprop,p,T,0));
  end;
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

[global]function volume(p, T, x : real):real;
begin
  if p>=1 then
    volume:=x/value(densityprop,p,T,1)+(1-x)/value(densityprop,p,T,0)
   else
    volume:=x/value(lo_densityprop,p,T,1)+(1-x)/value(lo_densityprop,p,T,0)
end;
```

```
        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

[global]function exergy(p, T, x : real):real;
begin
if p>=0 then exergy:=value(enthalpyprop, p ,T ,x )-
                  dead_T*value(entropyprop, p, T, x)-dead_state_availability
        else exergy:=value(lo_enthalpyprop, p, T, x)-
                  dead_t*value(lo_entropyprop, p, T, x)-
                  dead_state_availability;
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

[global] function latent_heat(p : real): real;
var T_sat_gl : real;
begin
 T_sat_gl :=saturation_temperature(p);
 if p>=1 then latent_heat:=value(enthalpyprop, p, T_sat_gl, 1)
                -value(enthalpyprop, p, T_sat_gl, 0)
         else latent_heat:=value(lo_enthalpyprop, p, T_sat_gl, 1)
                -value(lo_enthalpyprop, p, T_sat_gl, 0);
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

[global]function surface_tension(T : real): real;
const T_star = 647.15;
      Big_B  = 0.2358;
      Lil_b  = -0.625;
      mu     = 1.256;
begin
 surface_tension:=Big_B*((1-T/T_star)**mu)*(1+Lil_b*(1-T/T_star));
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

[global]function int_energy(p,T,x : real):real;
begin
 if p>=1 then int_energy:=value(int_energyprop, p, T, x)
        else int_energy:=value(lo_int_energyprop, p, T, x);
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
     SEE ALSO LEVEL 3 FOR GLOBAL FUNCTIONS SATURATION_TEMPERATURE AND
                      SATURATION_PRESSURE
        @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

{@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
      PROCEDURES FOR THE READING OF DATA AND SETUP OF THE MODULE
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure read_in_ordinate(var whichord_1 : ordinate; var location_1 : text);

{Reads in the data for an ordinate from a text file}

var index1 : ordinate_index;

begin
 with whichord_1 do
   begin
     readln(location_1, N_evalpts);
     readln(location_1);
     for index1:=1 to N_evalpts do read(location_1, evalpt[index1]);
     readln(location_1);
   end;
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure read_in_property_table(var whichprop_1 : property;
                                  var location_1 : text);

{Reads in the table of property values at grid temperatures and pressures from
a text file.}

var index1, index2 : ordinate_index;

begin
 with whichprop_1 do
     for index1:=1 to pressure.N_evalpts do
       begin
         for index2:=1 to temperature.N_evalpts do
                read(location_1, table_value[index1, index2]);
         readln(location_1);
       end;
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure read_in_sat_vals(var whichprop_1 : property; var location_1 : text);

{Reads in the table of saturation temperature and property values from a text
file.}

var index1 : ordinate_index;

begin
 with whichprop_1 do
   begin
     for index1:=1 to pressure.N_evalpts do
          readln(location_1, sat_val[index1,lo], sat_val[index1,hi]);
     readln(location_1);
   end;
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure read_saturation_line(var location_0 : text);

var index1 : ordinate_index;

begin
  with saturation do
    begin
      readln(location_0, N_evalpts);
      readln(location_0);
      for index1:=1 to N_evalpts do
           readln(location_0, pressure[index1], temperature[index1] );
      readln(location_0);
    end;
end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure read_in_property(var whichprop_0 : property; var location_0 : text);
begin
 with whichprop_0 do
   begin
```

```
      read_in_ordinate(pressure, location_0);
      read_in_ordinate(temperature, location_0);
      read_in_sat_vals(whichprop_0, location_0);
      read_in_property_table(whichprop_0, location_0);
    end;
 end;

        {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

 [global] procedure read_in_data(var location : text);

 {Reads in the data from a text file.}

 begin
  reset(location);
  readln(location,data_check);
  readln(location);
  read_saturation_line(location);
  read_in_property(enthalpyprop, location);
  read_in_property(entropyprop, location);
  read_in_property(viscosprop, location);
  read_in_property(Prandtlprop, location);
  read_in_property(conductprop, location);
  read_in_property(densityprop, location);
  read_in_property(lo_enthalpyprop,location);
  read_in_property(lo_entropyprop, location);
  read_in_property(lo_densityprop, location);
  read_in_property(lo_int_energyprop, location);
  read_in_property(int_energyprop, location);
  enthalpyprop.tag:=1;
  entropyprop.tag:=2;
  densityprop.tag:=3;
  lo_enthalpyprop.tag:=4;
  lo_entropyprop.tag:=5;
  lo_densityprop.tag:=6;
  int_energyprop.tag:=7;
  lo_int_energyprop.tag:=8;
  viscosprop.tag:=9;
  Prandtlprop.tag:=10;
  conductprop.tag:=11;
  dead_state_availability:=value(enthalpyprop, dead_p, (dead_T-273.15), 0)-
                  dead_T*value(entropyprop, dead_p, (dead_T-273.15), 0);
  writeln('READING OF DATA FROM FILE THERMO.DAT COMPLETED');
 end;

       {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}
                       {ANALYTICAL PROCEDURES}
       {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure T_from_property_and_pressure(whichprop_0 : property;
                         p_0, propval : real;var T_0, x_0 : real);

{This procedure finds the temperature (and vapour quality) from a given
property whichprop_0 with value propval and pressure p_0.}

var p_ind_0, B_0 : index_bracket;
    sat_T_0, hi_sat, lo_sat, lo_T_prop, hi_T_prop : real;
    newpt : ordinate_index;

begin
 with whichprop_0 do
  begin
   find_brackets(pressure.evalpt,pressure.N_evalpts, p_0, p_ind_0, p_error);
   hi_sat:=saturation_limit(whichprop_0, hi, p_ind_0, p_0);
   lo_sat:=saturation_limit(whichprop_0, lo, p_ind_0, p_0);
```

```
   sat_T_0:=saturation_temperature(p_0);
   if (lo_sat<=propval) and (hi_sat>=propval) then
    begin
     T_0:=sat_T_0;
     x_0:=(propval-lo_sat)/(hi_sat-lo_sat);
    end
   else
    with temperature do
     begin
      B_0[hi]:=N_evalpts;
      B_0[lo]:=1;

      repeat
        newpt:=(B_0[hi]+B_0[lo])div 2;
        if (propval-
              property_at_grid_T(whichprop_0, newpt, p_ind_0, sat_T_0, p_0)<0)
             then B_0[hi]:=newpt
             else B_0[lo]:=newpt;
      until (B_0[hi]-B_0[lo])=1;

      lo_T_prop:=property_at_grid_T(whichprop_0,B_0[lo],p_ind_0,sat_T_0, p_0);
      hi_T_prop:=property_at_grid_T(whichprop_0,B_0[hi],p_ind_0,sat_T_0, p_0);
      if (lo_sat>=propval) and (lo_sat<=hi_T_prop) then
       T_0:=interpolation(lo_T_prop,lo_sat, evalpt[B_0[lo]],sat_T_0, propval)

      else if (hi_sat<=propval) and (hi_sat>=lo_T_prop) then
       T_0:=interpolation(hi_sat, hi_T_prop,
            sat_T_0, evalpt[B_0[hi]], propval)
      else
       T_0:=interpolation(lo_T_prop, hi_T_prop, evalpt[B_0[lo]],
                                        evalpt[B_0[hi]], propval);
       if propval>hi_sat then x_0:=1 else x_0:=0;
    end;
  end;
end;

       {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

[global]procedure isobaric_heating(p, heatadded : real; var T, x : real);
{This procedure takes an initial condition p, T, x and finds the final
condition p, T, x for an isobaric heating process with heat addition
heatadded.}
begin
 if p>=1 then T_from_property_and_pressure(enthalpyprop, p,
              (value(enthalpyprop, p, T, x)+heatadded), T, x)
        else T_from_property_and_pressure(lo_enthalpyprop, p,
              (value(lo_enthalpyprop, p, T, x)+heatadded), T, x);
end;

       {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

[global]procedure T_from_s_and_p(s, p : real; var T, x : real);
{This procedure finds the temperature T and vapour quality x corresponding
to a given entropy s and pressure p.}
begin
 if p>=1 then T_from_property_and_pressure(entropyprop, p, s, T, x)
        else T_from_property_and_pressure(lo_entropyprop, p, s, T, x);
end;

       {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

[global]procedure T_from_h_and_p(h, p : real; var T, x : real);
{This procedure finds the temperature T and vapour quality x corresponding
to a given enthalpy h and pressure p.}
begin
 if p>=1 then T_from_property_and_pressure(enthalpyprop, p, h, T, x)
```

```
         else T_from_property_and_pressure(lo_enthalpyprop, p, h, T, x)

 end;
         {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

[global]procedure T_from_u_and_p(u, p : real; var T, x : real);
{This procedure finds the temperature T and vapour quality x corresponding
to a given internal energy u and pressure p.}
begin
 if p>=1 then T_from_property_and_pressure(int_energyprop, p, u, T, x)
         else T_from_property_and_pressure(lo_int_energyprop, p, u, T, x)

end;

         {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

procedure find_sat_state_from_S_and_x(propval_00, x_00 : real;
                                     var p_00, T_00 : real);
{This procedure finds the pressure p_00 and temperature T_00 corresponding to
a saturated state of entropy propval_00 and vapour quality x_00.}
var B_00 : index_bracket;
    newpt : ordinate_index;
    loval_00, hival_00, testval_00 : real;

begin

 B_00[hi]:=saturation.N_evalpts;
 B_00[lo]:=1;

 repeat
  newpt:=(B_00[lo]+B_00[hi]) div 2;
  if saturation.pressure[newpt]<1 then
         testval_00:=value(lo_entropyprop, saturation.pressure[newpt],
                   saturation.temperature[newpt], x_00)
    else testval_00:=value(entropyprop, saturation.pressure[newpt],
                   saturation.temperature[newpt], x_00);
  if (propval_00-testval_00)>0 then B_00[hi]:=newpt
                                else B_00[lo]:=newpt;

 until (B_00[hi]-B_00[lo])=1;

  if saturation.pressure[B_00[lo]]<1 then
        loval_00:=value(lo_entropyprop, saturation.pressure[B_00[lo]],
                  saturation.temperature[B_00[lo]], x_00)
   else loval_00:=value(entropyprop, saturation.pressure[B_00[lo]],
                  saturation.temperature[B_00[lo]], x_00);

  if saturation.pressure[B_00[hi]]<1 then
        hival_00:=value(lo_entropyprop, saturation.pressure[B_00[hi]],
                  saturation.temperature[B_00[hi]], x_00)
   else hival_00:=value(entropyprop, saturation.pressure[B_00[hi]],
                  saturation.temperature[B_00[hi]], x_00);

 p_00:=interpolation(loval_00, hival_00, saturation.pressure[B_00[lo]],
                  saturation.pressure[B_00[hi]], propval_00);
 T_00:=saturation_temperature(p_00);
 if p_00>saturation.pressure[saturation.N_evalpts] then
   begin
     writeln('ERROR : Failed to correctly find value on saturation line');
     writeln('Entropy value ',propval_00);
   end;
end;

         {@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@}

[global] procedure find_sat_state_from_entropy(s : real; var  p, T : real);
{This procedure finds the intersection of an isentrope of entropy s with
the saturation line x=1.}
begin
  find_sat_state_from_S_and_x(s,1,p,T);
end;


end.{of module}
```

Appendix 3 : Data File THERMO.DAT

```
-999.99

27

0.1 45.817
0.2 60.073
0.3 69.114
0.4 75.877
0.5 81.339
0.6 85.949
0.75 91.783
0.9 96.713
1   99.632
1.2 104.811
1.5 111.378
1.7 115.177
2   120.241
2.5 127.443
3   133.555
4   143.643
5   151.866
7   164.983
10  179.916
15  198.327
20  212.417
25  223.989
36  244.220
50  263.977
66  281.910
76  291.482
100 311.031

15

1 1.5 2 3 4 5 7 10 15 25 36 50 66 76 100

19

15 25 50 75 100 125 175 225 275 325 375 450 500 560 620 700 800 900 1000

417.51  2675.1
467.18  2693.4
504.8   2706.5
561.61  2725.3
604.9   2738.5
640.38  2748.6
697.35  2763.3
762.88  2777.7
844.86  2791.5
961.98  2802.2
1057.43 2802.3
1154.22 2793.7
1246.11 2776.9
1297.19 2763.6
1407.33 2724.5


63.01 104.84 209.40 314.01 2675.9 2726.3 2825.5 2924.3 3023.8 3124.4 3226.4 3382.3
3488.2 3617.5 3749.2 3928.8 4159.7 4397.5 4642.0

63.06 104.89 209.45 314.05 419.10 2721.6 2822.5 2922.3 3022.3 3123.3 3225.6 3381.7
3487.7 3617.0 3748.9 3928.5 4159.4 4379.3 4641.9

63.11 104.93 209.49 314.09 419.14 2716.6 2819.6 2920.3 3020.9 3122.2 3224.7 3381.0
3487.1 3616.6 3748.5 3928.3 4159.2 4397.1 4641.7
```

63.20 105.03 209.58 314.17 419.21 525.11 2813.5 2916.2 3017.9 3120.0 3223.0 3379.7
3486.1 3615.7 3747.8 3927.7 4158.8 4396.7 4641.4

63.30 105.12 209.66 314.25 419.29 525.18 2807.2 2912.1 3015.0 3117.8 3221.2 3378.5
3485.0 3614.9 3747.1 3927.1 4158.3 4396.4 4641.1

63.39 105.21 209.75 314.33 419.36 525.25 2800.7 2907.9 3012.0 3115.5 3219.4 3377.2
3483.9 3614.0 3746.3 3926.5 4157.8 4396.0 4640.8

63.58 105.40 209.92 314.49 419.51 525.39 2787.2 2899.3 3005.9 3111.0 3215.9 3374.6
3481.8 3612.2 3744.9 3925.3 4156.9 4395.3 4640.2

63.87 105.67 210.18 314.73 419.74 525.60 741.28 2885.8 2996.6 3104.1 3210.5 3370.7
3478.6 3609.6 3742.7 3923.6 4155.5 4394.2 4639.3

64.35 106.14 210.61 315.14 420.11 525.94 741.54 2861.6 2980.5 3092.3 3201.5 3364.2
3473.1 3605.2 3739.0 3920.6 4153.2 4392.3 4637.9

65.30 107.06 211.47 315.94 420.87 526.63 742.07 2805.4 2945.6 3067.7 3182.8 3350.9
3462.2 3596.2 3731.6 3914.7 4148.6 4388.7 4634.9

66.35 108.08 212.42 316.83 421.69 527.39 742.65 966.91 2902.4 3038.8 3161.5 3335.9
3449.9 3586.3 3723.3 3908.1 4143.5 4384.6 4631.6

67.69 109.37 213.63 317.96 422.75 528.36 743.40 967.25 2837.5 2998.7 3132.9 3316.3
3433.9 3573.4 3712.7 3899.7 4137.0 4379.4 4627.4

69.21 110.84 215.00 319.25 423.95 529.47 744.25 967.66 1209.85 2947.4 3098.0 3293.2
3415.3 3558.5 3700.4 3889.9 4129.5 4373.5 4622.7

70.16 111.76 215.86 320.06 424.71 530.17 744.79 967.92 1209.43 2911.6 3074.9 3278.3
3403.3 3549.0 3692.7 3883.8 4124.8 4369.8 4619.7

72.44 113.97 217.93 322.00 426.52 531.84 746.09 968.56 1208.52 2808.1 3014.7 3241.1
3374.0 3525.8 3673.8 3869.0 4113.5 4360.9 4612.5

15

1 1.5 2 3 4 5 7 10 15 25 36 50 66 76 100

20

15 25 35 50 75 100 125 175 225 275 325 375 450 500 560 620 700 800 900 1000

1.30273 7.3589
1.43376 7.2232
1.53035 7.1272
1.67211 6.9921
1.77700 6.8961
1.86104 6.8214
1.99254 6.7079
2.13885 6.5859
2.31496 6.4438
2.55439 6.2560
2.73992 6.1125
2.92013 5.9725
3.08455 5.8425
3.17336 5.7705
3.35918 5.6140

0.22422 0.36693 0.50493 0.70370 1.01549 7.3609 7.4918 7.7263 7.9353 8.1257 8.3014
8.4652 8.6927 8.8342 8.9953 9.1480 9.3405 9.5662 9.7781 9.9781

0.22421 0.36692 0.50491 0.70368 1.01546 1.30685 7.2952 7.5342 7.7452 7.9366 8.1129
8.2770 8.5049 8.6466 8.8077 8.9605 9.1531 9.3789 9.5908 9.7908

0.22421 0.36690 0.50489 0.70365 1.01543 1.30681 7.1527 7.3963 7.6094 7.8018 7.9787
8.1433 8.3714 8.5133 8.6746 8.8274 9.0201 9.2460 9.4579 9.6580

0.22419 0.36688 0.50486 0.70361 1.01536 1.30673 1.58142 7.1987 7.4161 7.6107 7.7888
7.9541 8.1830 8.3252 8.4867 8.6397 8.8325 9.0585 9.2705 9.4706

0.22417 0.36685 0.50482 0.70356 1.01530 1.30665 1.58132 7.0551 7.2770 7.4739 7.6533
7.8194 8.0489 8.1914 8.3531 8.5063 8.6993 8.9254 9.1375 9.3377

0.22416 0.36683 0.50479 0.70351 1.01524 1.30657 1.58123 6.9408 7.1676 7.3668 7.5475
7.7144 7.9446 8.0873 8.2493 8.4027 8.5959 8.8221 9.0342 9.2345

0.22413 0.36677 0.50472 0.70342 1.01511 1.30642 1.58104 6.7617 6.9990 7.2032 7.3865
7.5550 7.7866 7.9300 8.0924 8.2462 8.4396 8.6661 8.8784 9.0788

0.22408 0.36670 0.50461 0.70328 1.01492 1.30618 1.58076 2.09091 6.8135 7.0257 7.2133
7.3842 7.6180 7.7622 7.9254 8.0797 8.2736 8.5005 8.7130 8.9136

0.22401 0.36657 0.50444 0.70305 1.01461 1.30579 1.58029 2.09024 6.5885 6.8161 7.0114
7.1867 7.4242 7.5699 7.7343 7.8894 8.0841 8.3116 8.5246 8.7254

0.22385 0.36630 0.50409 0.70259 1.01399 1.30502 1.57935 2.08892 6.2625 6.5312 6.7444
6.9293 7.1746 7.3235 7.4905 7.6473 7.8436 8.0724 8.2862 8.4876

0.22368 0.36601 0.50371 0.70208 1.01330 1.30416 1.57833 2.08748 2.5617 6.3007 6.5390
6.7360 6.9908 7.1432 7.3131 7.4719 7.6701 7.9003 8.1151 8.3171

0.22345 0.36564 0.50321 0.70144 1.01243 1.30308 1.57703 2.08565 2.5590 6.0532 6.3352
6.5507 6.8187 6.9760 7.1498 7.3112 7.5117 7.7438 7.9598 8.1626

0.22319 0.36522 0.50265 0.70070 1.01144 1.30185 1.57555 2.08357 2.5559 3.0188 6.1390
6.3811 6.6663 6.8296 7.0080 7.1725 7.3757 7.6100 7.8274 8.0312

0.22303 0.36495 0.50230 0.70024 1.01082 1.30108 1.57463 2.08229 2.5541 3.0157 6.0255
6.2883 6.5855 6.7528 6.9342 7.1007 7.3057 7.5414 7.7596 7.9640

0.22262 0.36431 0.50146 0.69914 1.00394 1.29924 1.57243 2.07922 2.5496 3.0083 5.7555
6.0882 6.4194 6.5971 6.7862 6.9577 7.1671 7.4062 7.6266 7.8324

7

1 5 10 25 50 75 100

13

25 50 75 100 150 200 250 300 400 500 600 700 800

283.5 12.27
179.7 14.08
149.4 15.07
118.8 16.63
99.9 18.14
89.4 19.27
81.7 20.33

890.8 547.1 378.4 12.28 14.19 16.18 18.22 20.29 24.45 28.57 32.61 36.55 40.38

890.7 547.1 378.5 282.4 182.0 16.07 18.15 20.25 24.44 28.58 32.63 36.57 40.39

890.6 547.2 378.6 282.6 182.1 15.93 18.07 20.20 24.42 28.58 32.64 36.59 40.42

890.3 547.5 379.0 283.0 182.5 133.9 17.83 20.06 24.39 28.61 32.70 36.66 40.50

889.8 547.9 379.6 283.6 183.2 134.5 106.1 19.86 24.38 28.67 32.81 36.79 40.63

889.3 548.3 380.2 284.3 183.8 135.1 106.8 19.74 24.40 28.77 32.93 36.93 40.78

888.9 548.7 380.9 284.9 184.4 135.7 107.5 86.42 24.49 28.89 33.09 37.09 40.94

7

1 5 10 25 50 75 100

15

25 50 75 100 150 200 250 300 350 400 450 500 600 700 800

1.761 1.000
1.138 1.022
0.977 1.058
0.851 1.177
0.831 1.382
0.872 1.599
0.950 1.839

6.137 3.555 2.378 1.000 0.974 0.960 0.950 0.941 0.932 0.924 0.917 0.911 0.899
0.890 0.882

6.133 3.553 2.377 1.753 1.151 0.984 0.964 0.950 0.939 0.929 0.921 0.913 0.901
0.891 0.882

6.128 3.551 2.377 1.752 1.150 1.028 0.987 0.965 0.949 0.936 0.926 0.917 0.902
0.891 0.883

6.113 3.546 2.374 1.751 1.150 0.903 1.096 1.021 0.982 0.958 0.941 0.928 0.908
0.894 0.884

6.088 3.538 2.371 1.750 1.149 0.902 0.825 1.173 1.057 1.002 0.969 0.946 0.916
0.898 0.886

6.063 3.529 2.367 1.748 1.148 0.900 0.821 1.466 1.162 1.055 1.000 0.966 0.925
0.901 0.888

6.039 3.521 2.364 1.746 1.147 0.899 0.817 0.890 1.312 1.118 1.035 0.988 0.933
0.904 0.889

7

1 5 10 25 50 75 100

14

25 50 75 100 150 200 250 300 350 400 500 600 700 800

679.0 25.05
681.7 31.88
673.4 36.43
646.4 45.03
603.8 55.20
564.2 64.85
528.0 75.97

607.2 643.6 666.8 25.08 28.85 33.28 38.17 43.42 48.96 54.76 66.97 79.89 93.37
107.3

607.4 643.7 667.0 679.3 682.1 34.93 39.18 44.09 49.44 55.13 67.25 80.13 93.59
107.5

607.6 644.0 667.2 679.6 682.4 37.21 40.51 44.95 50.06 55.61 67.60 80.44 93.87
107.8

608.3 644.7 668.0 680.4 683.4 664.2 45.16 47.82 52.06 57.15 68.71 81.39 94.75

```
108.5

609.4 645.8 669.2 681.8 685.1 666.4 622.7 53.86 55.99 60.06 70.74 83.13 96.34
109.9

610.5 647.0 670.5 683.2 686.8 668.6 625.9 63.11 61.06 63.56 73.03 85.04 98.08
111.5

611.7 648.2 671.7 684.5 688.5 670.7 629.0 550.9 68.10 67.89 75.61 87.14 99.97
113.2

22

1 1.5 2 3 4 5 7 8.5 10 12 15 20 25 30 36 42 50 58 66 76 88 100

15

15 25 75 125 175 225 275 325 375 450 520 600 700 800 900 1000

958.66 0.5902
949.94 0.8624
942.96 1.1289
931.84 1.6505
922.91 2.1624
915.31 2.6677
902.58 3.6655
894.45 4.4068
887.15 5.144
878.37 6.125
866.69 7.592
849.85 10.041
835.19 12.508
821.99 15.001
807.51 18.037
794.13 21.130
777.51 25.355
761.93 29.712
747.10 34.218
729.32 40.087
708.73 47.527
688.61 55.47

999.14 997.06 974.86 0.5503 0.48669 0.43681 0.39643 0.36299 0.33480 0.29992 0.27337
0.24827 0.22272 0.20194 0.18472 0.17020

999.16 997.08 974.89 0.8303 0.7325 0.6566 0.5955 0.5497 0.5026 0.4501 0.41020
0.37249 0.33413 0.30294 0.27709 0.25531

999.18 997.11 974.91 1.1138 0.9800 0.8774 0.7952 0.7275 0.6706 0.6004 0.54712
0.49677 0.44557 0.40396 0.36947 0.34042

999.23 997.15 974.95 939.11 1.4804 1.3219 1.1963 1.0935 1.0075 0.9015 0.8212
0.7455 0.6685 0.6061 0.5543 0.5107

999.28 997.20 975.00 939.16 1.9883 1.7705 1.5998 1.4610 1.3453 1.2033 1.0957
0.9945 0.8917 0.8082 0.7391 0.6809

999.32 997.24 975.04 939.21 2.5042 2.2232 2.0059 1.8302 1.6842 1.5056 1.3706
1.2437 1.1149 1.0105 0.9240 0.8512

999.42 997.33 975.13 939.31 3.5619 3.1418 2.8255 2.5732 2.3652 2.1120 1.9215
1.7428 1.5618 1.4152 1.2939 1.1919

999.49 997.40 975.20 939.39 4.3804 3.8429 3.4471 3.1348 2.8788 2.5685 2.3357
2.1177 1.8973 1.7189 1.5714 1.4474
```

999.56 997.47 975.26 939.46 892.41 4.5551 4.0748 3.7001 3.3948 3.0263 2.7508
2.4932 2.2331 2.0228 1.8491 1.7030

999.65 997.56 975.35 939.57 892.54 5.523 4.922 4.4598 4.087 3.6390 3.3056
2.9946 2.6813 2.4283 2.2194 2.0439

999.79 997.69 975.49 939.72 892.74 7.018 6.214 5.613 5.1330 4.5628 4.1407
3.7486 3.3546 3.0370 2.7752 2.5553

1000.02 997.92 975.71 939.97 893.06 9.640 8.432 7.571 6.900 6.115 5.541
5.010 4.4794 4.0531 3.7023 3.4082

1000.25 998.14 975.93 940.23 893.39 12.463 10.739 9.578 8.698 7.685 6.951
6.278 5.608 5.071 4.6305 4.2616

1000.49 998.36 979.06 940.48 893.71 834.29 13.148 11.639 10.528 9.271 8.371
7.551 6.739 6.091 5.560 5.116

1000.76 998.63 976.42 940.78 894.09 834.85 16.192 14.188 12.768 11.198 10.089
9.088 8.101 7.317 6.676 6.141

1001.04 998.90 976.68 941.09 894.48 835.40 19.782 16.829 15.061 13.151 11.823
10.633 9.468 8.546 7.794 7.167

1001.41 999.26 977.03 941.49 894.99 836.13 24.155 20.511 18.203 15.798 14.160
12.709 11.299 10.189 9.287 8.536

1001.78 999.61 977.38 941.89 895.50 836.85 29.474 24.402 21.454 18.497 16.526
14.801 13.138 11.836 10.782 9.907

1002.15 999.97 977.73 942.29 896.01 837.57 760.26 28.540 24.822 21.249 18.922
16.909 14.986 13.488 12.280 11.279

1002.60 1000.41 978.17 942.79 896.64 838.46 761.80 34.133 29.216 24.768 21.961
19.570 17.309 15.560 14.155 12.995

1003.15 1000.95 978.70 943.39 897.39 839.53 763.61 41.640 34.794 29.117 25.674
22.799 20.115 18.057 16.411 15.057

1003.70 1001.48 979.22 943.98 898.14 840.58 765.38 50.366 40.763 33.611 29.463
26.068 22.941 20.564 18.673 17.122

10

0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

15

25 35 45 55 65 75 85 95 105 120 180 240 300 360 420

191.83 2583.8
251.46 2608.9
289.28 2624.6
317.63 2636.1
340.54 2645.3
359.90 2653.0
376.73 2659.6
391.70 2665.3
405.20 2670.5
417.51 2675.1

104.76 146.59 188.42 2601.3 2620.4 2639.4 2658.4 2677.4 2696.5 2725.1 2840.3
2957.2 3076.2 3197.4 3320.8

104.77 146.60 188.42 230.24 2618.4 2637.7 2657.0 2676.2 2695.4 2724.1 2839.7
2956.9 3075.9 3197.2 3320.7

104.78 146.61 188.43 230.25 272.08 2636.1 2655.5 2674.9 2694.2 2723.2 2839.2
2956.5 3075.7 3197.0 3320.5

104.79 146.62 188.44 230.26 272.09 313.96 2654.1 2673.6 2693.1 2722.2 2838.6
2956.2 3075.4 3196.8 3320.4

104.80 146.63 188.45 230.27 272.10 313.97 2652.6 2672.3 2691.9 2721.2 2838.1
2955.8 3075.2 3196.6 3320.3

104.81 146.64 188.46 230.28 272.11 313.98 355.92 2671.0 2690.8 2720.3 2837.5
2955.4 3074.9 3196.4 3320.1

104.81 146.65 188.47 230.28 272.11 313.98 355.93 2669.7 2689.6 2719.3 2837.0
2955.1 3074.7 3196.2 3320.0

104.82 146.65 188.48 230.29 272.12 313.99 355.93 2668.3 2688.4 2718.3 2836.4
2954.7 3074.4 3196.1 3319.8

104.83 146.66 188.49 230.30 272.13 314.00 355.94 397.99 2687.3 2717.3 2835.9
2954.4 3074.2 3195.9 3319.7

104.84 146.67 188.49 230.31 272.14 314.01 355.95 397.99 2686.1 2716.3 2835.3
2954.0 3073.9 3195.7 3319.5

10

0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

15

25 35 45 55 65 75 85 95 105 120 180 240 300 360 420

0.64926 8.1482
0.83211 7.9068
0.94406 7.7672
1.02607 7.6688
1.09117 7.5928
1.14540 7.5310
1.19197 7.4789
1.23298 7.4339
1.26960 7.3943
1.30273 7.3589

0.36695 0.50496 0.63853 8.2024 8.2596 8.3151 8.3690 8.4214 8.4724 8.5466 8.8193
9.0615 9.2808 9.4818 9.6681

0.36695 0.50496 0.63853 0.76794 7.9352 7.9914 8.0459 8.0988 8.1502 8.2248 8.4984
8.7411 8.9606 9.1617 9.3480

0.36695 0.50495 0.63852 0.76794 0.89350 7.8005 7.8555 7.9089 7.9607 8.0358
8.3104 8.5535 8.7731 8.9743 9.1607

0.36695 0.50495 0.63852 0.76793 0.89350 1.01553 7.7195 7.7733 7.8255 7.9011
8.1767 8.4202 8.6400 8.8413 9.0278

0.36694 0.50495 0.63852 0.76793 0.89349 1.01552 7.6132 7.6676 7.7202 7.7962
8.0728 8.3167 8.5767 8.7381 8.9247

0.36694 0.50494 0.63851 0.76792 0.89349 1.01551 1.13429 7.5806 7.6336 7.7101
7.9877 8.2320 8.4522 8.6538 8.8404

0.36694 0.50494 0.63851 0.76792 0.89348 1.01551 1.13428 7.5066 7.5600 7.6370
7.9156 8.1603 8.3807 8.5824 8.7691

0.36694 0.50494 0.63850 0.76791 0.89348 1.01550 1.13428 7.4421 7.4960 7.5734

7.8531 8.0982 8.3188 8.5206 8.7073

0.36693 0.50493 0.63850 0.76791 0.89347 1.01549 1.13427 1.25005 7.4391 7.5171
7.7978 8.0433 8.2641 8.4660 8.6528

0.36693 0.50493 0.63849 0.76790 0.89347 1.01549 1.13426 1.25004 7.3880 7.4665
7.7482 7.9942 8.2152 8.4172 8.6040

10

0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

9

25 55 85 115 175 240 300 360 420

989.82 0.06815
983.13 0.13072
978.25 0.19121
974.31 0.25038
970.96 0.30856
968.01 0.36598
965.36 0.42276
962.95 0.47902
960.72 0.53482
958.66 0.5902

997.02 0.06621 0.06061 0.05589 0.048379 0.042240 0.037813 0.034227 0.031263

997.03 985.66 0.12143 0.11192 0.09682 0.08451 0.07564 0.06847 0.06253

997.03 985.66 0.18248 0.16809 0.14533 0.12682 0.11349 0.10272 0.09381

997.03 985.67 0.24375 0.22439 0.19390 0.16915 0.15136 0.13698 0.12510

997.04 985.67 0.30526 0.28084 0.24254 0.21152 0.18925 0.17125 0.15639

997.04 985.67 968.63 0.33744 0.29124 0.25392 0.22716 0.20553 0.18769

997.05 985.68 968.63 0.39418 0.34000 0.29635 0.26508 0.23983 0.21899

997.05 985.68 968.63 0.45107 0.38883 0.33881 0.30302 0.27413 0.25031

997.06 985.69 968.64 0.50811 0.43773 0.38131 0.34098 0.30845 0.28163

997.06 985.69 968.64 0.5653 0.48669 0.42384 0.37896 0.34278 0.31296

10

0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

15

25 35 45 55 65 75 85 95 105 120 180 240 300 360 420

191.82 2245.16
251.44 2455.90
289.25 2467.70
317.59 2476.34
340.49 2483.26
359.84 2489.06
376.66 2494.02
391.62 2498.29
405.11 2502.22
417.41 2505.67

104.75 146.58 188.41 2450.3 2464.7 2479.0 2493.4 2507.8 2522.2 2543.8 2631.3
2720.5 2811.7 2905.2 3001.0

104.75 146.58 188.40 230.22 2463.1 2477.7 2492.3 2506.8 2521.3 2543.1 2630.9
2720.2 2811.5 2905.1 3000.9

104.75 146.58 188.40 230.22 272.05 2476.4 2491.1 2505.8 2520.4 2542.4 2630.4
2719.9 2811.4 2904.9 3000.7

104.75 146.58 188.40 230.22 272.05 313.92 2490.0 2504.8 2519.5 2541.6 2630.0
2719.7 2811.2 2904.8 3000.6

104.75 146.58 188.40 230.22 272.05 313.92 2488.8 2503.7 2518.6 2540.9 2629.6
2719.4 2811.0 2904.6 3000.5

104.75 146.58 188.40 230.22 272.04 313.91 355.86 2502.7 2517.7 2540.1 2629.2
2719.1 2810.8 2904.5 3000.4

104.74 146.58 188.40 230.21 272.04 313.91 355.85 2501.7 2516.8 2539.3 2628.8
2718.9 2810.6 2904.4 3000.3

104.74 146.57 188.40 230.21 272.04 313.91 355.85 2500.6 2515.9 2538.6 2628.4
2718.6 2810.4 2904.2 3000.2

104.74 146.57 188.40 230.21 272.04 313.91 355.85 397.89 2515.0 2537.8 2627.9
2718.3 2810.2 2904.1 3000.1

104.74 146.57 188.39 230.21 272.04 313.90 355.85 397.89 2514.0 2537.0 2627.5
2718.1 2810.1 2904.0 3000.0

15

1 1.5 2 3 4 5 7 10 15 25 36 50 66 76 100

18

25 50 75 100 125 175 225 275 325 375 450 500 560 620 700 800 900 1000

417.41 2505.67
467.02 2519.47
504.59 2529.34
561.29 2543.54
604.47 2553.42
639.83 2561.17
696.57 2572.33
761.75 2583.30
843.13 2593.92
958.99 2602.33
1052.97 2602.71
1147.79 2596.50
1237.28 2584.02
1286.77 2574.01
1392.81 2544.22

104.74 209.30 313.90 2506.3 2544.6 2620.0 2695.3 2771.5 2848.9 2927.8 3048.9
3131.6 3233.2 3337.2 3479.8 3664.5 3856.1 4054.5

104.74 209.29 313.89 418.94 2540.9 2617.8 2693.8 2770.4 2848.1 2927.1 3048.4
3131.2 3232.9 3336.9 3479.6 3664.3 3856.0 4054.3

104.73 209.29 313.88 418.93 2537.1 2615.5 2692.3 2769.4 2847.3 2926.5 3047.9
3130.8 3232.5 3336.7 3479.4 3664.1 3855.8 4054.2

104.73 209.27 313.86 418.90 524.80 2610.8 2689.3 2767.2 2845.6 2925.2 3047.0
3130.1 3231.9 3336.1 3478.9 3663.8 3855.5 4054.0

104.72 209.26 313.84 418.87 524.76 2606.0 2686.1 2765.0 2844.0 2923.9 3046.0
3129.3 3231.2 3336.1 3478.5 3663.4 3855.2 4053.7

104.71 209.24 313.82 418.84 524.72 2601.1 2683.0 2762.7 2842.3 2922.6 3045.1
3128.5 3230.6 3335.0 3478.0 3663.0 3854.9 4053.4

104.70 209.21 313.77 418.78 524.65 2590.6 2676.5 2758.2 2838.9 2919.9 3043.2
3126.9 3229.2 3333.9 3477.1 3662.3 3854.3 4052.9

104.67 209.17 313.71 418.70 524.53 740.16 2666.2 2751.2 2833.8 2916.0 3040.3
3124.5 3227.2 3332.2 3475.7 3661.2 3853.4 4052.1

104.63 209.09 313.60 418.55 524.35 739.86 2647.9 2739.1 2825.1 2909.2 3035.4
3120.4 3223.9 3329.4 3473.4 3659.3 3851.8 4050.8

104.56 208.94 313.38 418.26 525.97 739.27 2604.8 2712.9 2806.7 2895.4 3025.5
3112.2 3217.2 3323.7 3468.8 3655.6 3848.8 4048.2

104.47 208.78 313.15 417.94 523.57 738.62 962.60 2680.1 2785.0 2879.5 3014.4
3103.0 3209.7 3317.5 3463.7 3651.5 3845.4 4045.4

104.36 208.58 312.85 417.54 523.05 737.81 961.27 2630.5 2754.9 2858.2 2999.8
3091.1 3200.0 3309.4 3457.1 3646.3 3841.0 4041.7

104.24 208.34 312.50 417.09 522.47 736.89 959.78 1201.17 2716.1 2832.1 2982.6
3077.1 3188.7 3300.0 3449.5 3640.2 3836.0 4037.5

104.17 208.20 312.29 416.80 522.11 736.32 958.85 1199.46 2688.9 2814.8 2971.4
3068.2 3181.5 3294.0 3444.7 3636.4 3832.9 4034.9

103.98 207.85 311.79 416.13 521.24 734.96 956.67 1195.6 2609.6 2769.4 2943.6
3046.2 3164.0 3279.5 3433.1 3627.2 3825.3 4028.5