INL/RPT-22-68471-Revision-0

Tabulated Fluid Properties Research Report

September 2022

Benjamin Thomas Spaude

Changing the World's Energy Future

INL is a U.S. Department of Energy National Laboratory operated by Battelle Energy Alliance, LLC

INL/RPT-22-68471-Revision-0

Tabulated Fluid Properties Research Report

Benjamin Thomas Spaude

September 2022

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

Tabulated Fluid Properties

Benjamin Spaude*
Department of Physics, St. Norbert College, DePere, WI 54115

(Dated: July 28th, 2022)

The Multiphysics Object-Oriented Simulation Environment (MOOSE) enables a wide range of advanced nuclear reactor simulations.[6] Under the guidance of MOOSE's Thermal Hydraulics Team, I worked to expand the capabilities of Tabulated Fluid Properties (TFP) in the fluid properties module. The fluid properties module allows the user to determine a variety of fluid properties by interpolating points between tabulated data. I implemented the ability to use bilinear interpolation instead of bicubic interpolation for interpolating tabulated data. I also changed the method of variable set inversions to use a 2-dimensional Newton's Method utility that I created. Variable set inversions are often done from (v,e) to (p,T), where v is specific volume, e is specific internal energy, p is pressure and T is temperature. New routines have also been added into TFP such that it can be used with more applications, such as the Navier Stokes and Thermal Hydraulics modules in MOOSE for Pronghorn[5] and RELAP-7[1] respectively. This work was spurred by interest from NASA in testing a Nuclear Thermal Propulsion (NTP) engine system. NTP engines have drastically different fluid properties throughout the engine and Tabulated Fluid Properties provides the flexibility needed to properly simulate and test these engines.

I. INTRODUCTION

While at the Idaho National Laboratory, I worked under the guidance of Dr. Guillaume Giudicelli on computational framework in MOOSE's [6] fluid properties module, specifically Tabulated Fluid Properties (TFP). Tabulated Fluid Properties allows for the calculation of various fluid properties by using tabulated data and interpolation methods to determine the values of fluid properties between data points. TFP makes use of many different variable sets and can invert between them before interpolation. If tabulated data doesn't exist, TFP will generate tabulated data with the aid of another fluid property object within the fluid properties module. I modified the bilinear interpolation utility in MOOSE to be used with TFP. Prior to this, bicubic interpolation was the only interpolation method used in TFP. When variable set inversions are initially performed, a 2-dimensional Newton's method is used to invert between variable sets. This led me to develop a Newton Method utility for the whole fluid properties module. The Newton Method utility can be used in 1-dimension as well as 2-dimensions.

II. TABULATED FLUID PROPERTIES OVERVIEW

Tabulated Fluid Properties (TFP) allows the user to compute a variety of fluid properties by interpolating between tabulated data points. TFP is especially useful when tabulated data already exists as it cuts down on computation time needed to generate tabulated data. If analytical formulas or functional fits exist for a fluid property, they should be used to compute the fluid properties instead of tabulated data. If tabulated data is not already present, it is generated with another fluid properties object. Bicubic or Bilinear Interpolation is used to determine the fluid properties at points in between the table of data points. The fluid properties module can use a variety of variable sets, but will perform a variable set inversion to a (p,T) variable set if (p,T) data does not already exist. There are numerous ways variable set inversions can be done, but for TFP I created a 2-dimensional Newton's Method utility due to its quick convergence (converges after about five iterations). Currently TFP can calculate 13 different fluid properties from a variety of variable sets. The 13 fluid properties used in TFP can be seen in Table I. The current variable sets used by TFP are (p,T), (v,e), (v,h), (h,s), (p,h), and (p,ρ), however the most commonly used variable sets are (p,T), (v,e), and (v,h). New routines have been added into TFP which allow it to be used in more applications, such as the Navier Stokes and Thermal Hydraulics modules in MOOSE for Pronghorn[5] and RELAP-7[1], respectively. Tabulated Fluid Properties has the flexibility needed to simulate situations where fluid properties can vary drastically.

Table I: Table showing the different fluid properties used by TFP.

Symbol	Fluid Property	Units
p	Pressure	Pa
T	Temperature	K
v	Specific Volume	m^3/kg
e	Specific Internal Energy	J/kg
h	Specific Enthalpy	J/kg
s	Specific Entropy	J/kg·K
g	Gibbs Free Energy	J
c_p	Specific Heat Capacity at Constant Pressure	J/kg·K
c_v	Specific Heat Capacity at Constant Volume	J/kg·K
ρ	Density	kg/m^3
c	Speed of Sound in Fluid	m/s
μ	Dynamic Viscosity	Pa·s
k	Thermal Conductivity	W/m·K

III. VARIABLE SETS

A variety of different variable sets are used to determine the desired fluid properties. Different variable sets are used as some experiments allow for specific variable sets to be measured easier than others. Most commonly, (p,T) variable sets are used as pressure and temperature are often the easiest variables to measure. Some applications may use other variable sets, such as (v,e) or (v,h). Fluid properties can vary drastically when different variable sets are used. An example of this can be seen in Figure 1 and 2, where the density is plotted as a function of (p,T) and (v,e). Certain applications do not

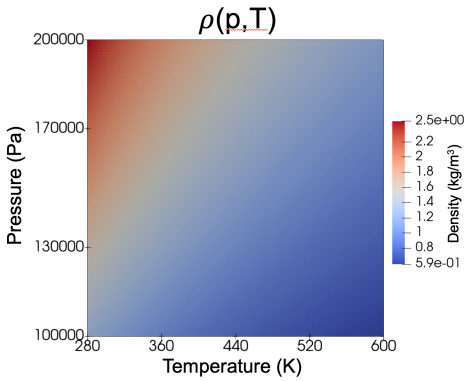

Figure 1: Density as a function of (p,T).

initially have access to (p,T) or other variable sets, in which case inversions between variable sets are needed. Inversions between variable sets are performed using a 2-dimensional (2D) Newton's Method. These variable set inversions come at a cost to computation time since Newton's Method is an iterative method. Fortunately, Newton's Method tends to converge quadratically (if the

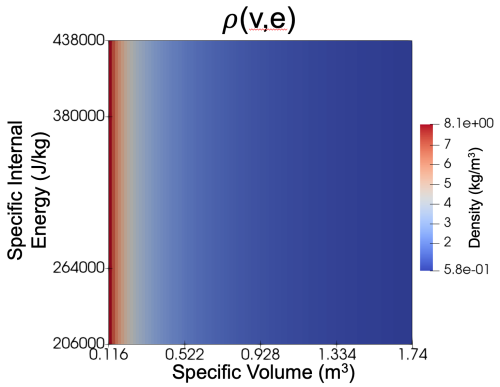

Figure 2: Density as a function of (v,e)

Jacobian is exact), making it an ideal option for these necessary variable set inversions. It is important to note that if variable sets other than (p,T) are used, a (p,T) variable set is constructed before tabulated data can be interpolated. As a result, when variable sets such as (v,e) are used, variable set inversions are always being done.

IV. GENERATING TABULATED DATA

TFP may be used to generate tabulations. Tabulations are generated when they do not exist in the data file given. To generate tabulated data, pressure and temperature ranges are divided in a regular grid using their maximum and minimum values. The user may choose how many subdivisions, but the default is 100. The process of dividing the data into equal segments is done using the following equation

$$\Delta x = \frac{x_{max} - x_{min}}{N - 1},$$

where N is the number of points, and x_{max} and x_{min} are the maximum and minimum values, respectively, of the relevant variable, x. Tabulated data can then be generated for each fluid property at the pressure and temperature points by evaluating the fluid properties routine at each point. Once tabulated data is generated, it is used to create interpolation tables.

It is important to note that the interpolation process varies slightly depending on the variable set. If (p,T) variable sets are used, interpolation is done immediately to determine the value for the desired fluid property. If a variable set such as (v,e) or (v,h) is used, we invert the variable set to (p,T) before interpolation. Upon initialization, inversion is done using a 2D Newton's Method, else calls to the fluid properties routines are made. Variable set inversions to (p,T) are desired due to (p,T) formulations being less complicated than other variable sets. The method of generating tabulated data is often better than using iterative methods as iterations tend to require more computation time and memory. As a result, it is preferred to use Tabulated Fluid Properties whenever possible.

V. INTERPOLATION METHODS

Tabulated Fluid Properties has two different interpolation methods that can be used, bicubic interpolation or bilinear interpolation. In most cases, bicubic interpolation is a better option as it has continuous derivatives, unlike bilinear interpolation. However, concerns about the monotonicity for the fluid properties during conversions between (p,T) and (v,e) justify the introduction of a monotonic bilinear interpolation. Interpolation methods are used after tabulated data has been generated to determine the fluid properties value between tabulated data points. Depending on the variable set used, the interpolation process may differ. When (p,T) is not the variable set used, TFP must first convert to a (p,T) variable set before interpolating the tabulated data. If a (p,T) variable set is available, TFP can interpolate the tabulated data immediately. When initializing tabulated data tables, TFP creates grids for the variable set used based on the maximum and minimum values of the data set. Then, variable set inversions can take place using 2D Newton's Method before interpolation occurs. If the tabulated data tables already exist for the fluid properties, Newton's Method is not needed and interpolation can be done without variable set inversions by making calls to the fluid properties routine.

I implemented bilinear interpolation as an alternative method to bicubic interpolation. With bicubic interpolation, there were concerns during variable set inversion where inversion to (p,T) gives values out of user-defined bounds. Because of this I added the capability of using bilinear interpolation instead of bicubic interpolation to avoid these concerns. In order to efficiently add this capability, I used polymorphism to avoid rewriting shared code. Polymorphism is a C++ technique that can be used to improve run time or compile time. Two polymorphism techniques were used to accomplish this: templating and inheritance.

A. Templating

Templating is a technique used when the same routines are used with different typenames within the same file. This allows the developer to avoid rewriting the same code within a file. It helps to improve the compile time of a code, however it adds to the run time of the code. When used in combination with inheritance, templating can cause some issues. Inheritance often makes use of virtual routines such that they can be overridden in the derived class. If a routine is templated in MOOSE, it

can not be made into a virtual routine, in which case the routine will not function as desired. This is a limitation of the C++ language, common to many other languages. This limits the use of templating in some instances, but not all. An example of templating can be seen in Figure 3.

Figure 3: An example of templating used in bicubic interpolation.

B. Inheritance

Inheritance is a technique used to create new classes from existing classes. The base class (existing class) is the primary class, while the derived classes are secondary classes (new classes). The derived classes will inherit the properties of the bass class, with the ability to add new features. A bass class is only one file, while each derived class has a file of their own. An example of inheritance can be seen in Figure 4, where inheritance in the fluid properties module is shown. The base class is fluid properties which has many branches of derived classes (Figure 4 is not a complete list of the derived classes in the fluid properties module). I split TFP between three files, "TabulatedFluidProperties", "TabulatedBicubicFluidProperties", and "TabulatedBilinearFluidProperties". TFP holds the routines and shared code between the interpolation methods, while the other two files just have the routines specific to the interpolation method.

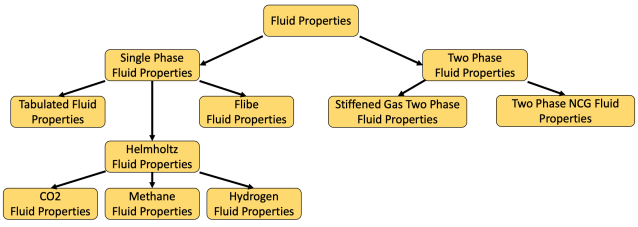

Figure 4: Inheritance system in the fluid properties module.

VI. NEWTON'S METHOD

As different applications use different variable sets, fluid properties are most often provided as functions of pressure and temperature. To evaluate the properties, a conversion to (p,T) must first occur. In order to save on computation time, I created a utility in the fluid properties module capable of doing Newton's Method in 2-dimensions (2D) as well as 1-dimension (1D). MOOSE already had a Newton's Method utility but it did not have the 2D capabilities needed for use in TFP. Newton's Method in 2D is given by the following iterative formula[2] ,

$$\vec{x}_{i+1} = \vec{x}_i - J^{-1} f(\vec{x}_i), \tag{1}$$

where $\vec{x}_{i+1}$ is the $i+1$ vector, $\vec{x}_i$ is the ith vector, and $f(\vec{x}_i)$ is a function at the point $\vec{x}_i$. The vectors are 2D vectors of the form $\begin{bmatrix} x \\ y \end{bmatrix}$. J is a 2x2 Jacobian Matrix given by

$$J = \begin{bmatrix} \frac{df}{dx} & \frac{df}{dy} \\ \frac{dg}{dx} & \frac{dg}{dy} \end{bmatrix}. \tag{2}$$

The Jacobian in Equation 2 is comprised of the derivatives of the function $f(x,y)$ and $g(x,y)$. These functions represent the properties we are converting from. For example, if we have a (v,e) variable set and want (p,T), $f(x,y)$ represents v(p,T) and $g(x,y)$ represents e(p,T). In equation 1, the function $f(\vec{x}_i)$ is obtained by making a call to the fluid property from the variable set the property is being converted from. Using the same example above, this would mean a call is made to the routine for v from (p,T). In C++, lambda functions were needed to accomplish this since different routines using Newton's Method have to make calls to properties from different variable sets. An example of the code can be seen in Figure 5.

```
void
SinglePhaseFluidProperties::p_T_from_v_e(const Real & v, //v value
                                         const Real & e, //e value
                                         const Real & p0,//initial guess
                                         const Real & T0,//intial guess
                                         Real & p, //returned pressure
                                         Real & T //returned temperature
                                         ) const
{
  // lambda function to get derivatives of v with respect to p and T
  auto v_lambda = [&](Real pressure, Real temperature, Real & new_v, Real & dv_dp, Real & dv_dT)
  {
    // call to v_from_p_T to fill v and its derivatives
    v_from_p_T(pressure, temperature, new_v, dv_dp, dv_dT);
  };
  // lambda function to get derivatives of e with respect to p and T
  auto e_lambda = [&](Real pressure, Real temperature, Real & new_e, Real & de_dp, Real & de_dT)
  {
    // call to e_from_p_T to fill e and its derivatives
    e_from_p_T(pressure, temperature, new_e, de_dp, de_dT);
  };
  // call to Newton Method to solve for p and T. Results saved to p and T
  NewtonMethod::NewtonSolve2D(v, e, p0, T0, p, T, _tolerance, v_lambda, e_lambda);
}
```

Figure 5: Newton's Method in 2D used to get (p,T) from (v,e). The results are stored as p and T, as seen in the last line of code.

When using Newton's Method, an initial guess is needed for $\vec{x}_i$ in order to start the first iteration ($i = 0$). In TFP, the user sets the initial guess in the input file for all calls to variable set inversions. If no initial guess is set, there are default values that will be used, however the default values may be non-physical for some fluid properties. It is recommended to set the initial guesses manually. Newton's Method will iterate until convergence is reached. Convergence occurs when a certain criterion has been reached. The criteria is based on a tolerance of the difference between the new and old solution. The tolerance defaults to 1e-8, but can be changed in the input file depending on the user's desired level of precision. Note, a higher level of precision requires more iterations and will cause a longer run-time. Newton's Method tends to converge quickly, usually only requiring approximately five iterations. If 100 iterations are reached, iteration stops and the solve does not converge. The result is then saved as a reference in "x_final" and "y_final". A snippet of the code for the Newton Method utility can be seen in Figure 6.

```
while (residual > tolerance) //iterate until residual is smaller than tolerance reached
{
  Real new_f, df_dx, df_dy, new_g, dg_dx, dg_dy;
  func1(current_vec[0], current_vec[1], new_f, df_dx, df_dy); //get new h and derivatives
  func2(current_vec[0], current_vec[1], new_g, dg_dx, dg_dy); //get new s and derivatives

  jacobian << df_dx, df_dy, //fill jacobian
              dg_dx, dg_dy;

  function << new_f, new_g; //fill function
  next_vec = current_vec - (jacobian.inverse() * ( function - target)); //2D Newton Method
  res1 = (current_vec[0] - next_vec[0]); //update residual 1
  res2 = (current_vec[1] - next_vec[1]); //update residual 2
  residual = pow(pow(res1, 2) + pow(res2, 2), 0.5); //update residual

  current_vec = next_vec; //update current_vec for next iteration
  ++iteration; //update iteration;

  if (iteration > 100)
    mooseError("2D Newton Solve, Convergence Failed");
}

x_final = current_vec[0]; //returned p
y_final = current_vec[1]; //returned y
}
```

Figure 6: The main block of code for 2D Newton's Method. Newton's Method iterates through until the convergence criteria is met and stores results.

Previously Tabulated Fluid Properties used a nearest-point method to invert variable sets. This method generated concerns as there is no guarantee that the data point is near any of the tabulated data. With Newton's Method this problem is avoided since the new data is determined directly from existing tabulated data. During the variable set inversion, it is possible that new data will be out of bounds (user sets (p,T) bounds in an input file). With Newton's Method, we were able to resolve this by setting the data equal to the maximum or minimum value of the property (the (p,T) maximum or minimum can be set in an input file). Ultimately, Newton's Method allowed for easy inversions between variable sets without the flaws of the nearest-point method. TFP currently uses Newton's method to invert to (p,T) from (v,e), (v,h), and (h,s). The (h,s) variable set is relatively rare, but extremely useful in isentropic thermal hydraulic applications where entropy is conserved, making for easy measurements of s.

Newton's Method in 1D is also given by Equation 1, however the Jacobian is different. In this instance, f is a function of one variable, and there is no function g. Thus, the Jacobian is equal to the derivative of the function and the 1D Newton's method takes the form:

$$x_{i+1} = x_i - \frac{f(x_i)}{\frac{\partial f}{\partial x}}.$$

In TFP, x represents the desired fluid property from some variable set (y, z), while $f(x)$ represents a fluid property as a function of x. For example, if we want temperature from (p,rho), x is the temperature, while $f(x)$ is $\rho(T)$. TFP currently only uses Newton Method in 1D for determining T(p,ρ) and T(p,h). These two formulations are difficult to obtain analytically, and so numerical methods are needed to obtain them. Newton's Method is ideal here due to its fast convergence. The utility for 1D Newton's Method is made using the same technique as in 2D, with one major difference. The 2D Newton's Method has a 2×2 Jacobian and requires the use of matrices. While the 1D utility used "Real" values, the 2D utility made use of "RealEigenMatrix" and "RealEigenVector", matrix formats defined by the Eigen[3] library from libmesh[4].

VII. CONCLUSION

During my internship at Idaho National Laboratory, I have improved the capabilities and documentation of the Tabulated Fluid Properties program. First, I added the ability to choose between bicubic or bilinear as the interpolation method. This was done using C++ templating and inheritance to maximise code reuse and maintainability. In order to make bilinear interpolation compatible with TFP, I added a routine to calculate derivatives in bilinear interpolation. Secondly, I added a variety of new routines to calculate fluid properties from new variable sets. This allowed for TFP to be used in other MOOSE modules, such as the Navier Stokes and Thermal Hydraulics Modules. Some of the new routines used a 1D Newton Method along with finite difference to determine the desired fluid property and its derivatives. This was the first time this was done in the fluid properties module. I also modified the method of variable set inversions by replacing the old method, nearest-point method, with a 2D Newton's Method. This was an efficient way of inverting between variable sets, however it required a fix for values out of user-defined bounds. Using Newton's Method for variable set inversions led me to develop a Newton Method utility that can be used throughout the fluid properties module. There is still minor work to be done on TFP, but I have ultimately gotten Tabulated

Fluid Properties to a point where it can be used for a variety of applications, especially when tabulated data is already present.

This internship allowed me to improve my programming skills in C++, my problem solving abilities, and helped me become more familiar with methods of numerical analysis. I was also able to learn a lot of professional development by engaging in department meetings and through intern-enrichment activities. My internship has also allowed me to obtain exposure to research that can help me achieve my goal of attending graduate school. Overall, I was able to have a positive internship experience where I made many connections and learned a lot about a career in research.

VIII. ACKNOWLEDGEMENTS

I would like to thank the Department of Energy and SULI program for funding my internship as well as the MOOSE team for allowing me the opportunity to work with them and for the help they provided during my internship. I would also like to give a special thanks to Dr. Guillaume Giudicelli for the research and career guidance he gave me.

REFERENCES

[1] David Andrs et al. "RELAP-7 Level 2 Milestone Report: Demonstration of a Steady State Single Phase PWR Simulation with RELAP-7". In: (May 2012). DOI: 10.2172/1047196. URL: https://www.osti.gov/biblio/1047196.

[2] A. Galántai. "The theory of Newton's method". In: *Journal of Computational and Applied Mathematics* 124.1 (2000). Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations, pp. 25–44. ISSN: 0377-0427. DOI: https://doi.org/10.1016/S0377-0427(00)00435-0. URL: https://www.sciencedirect.com/science/article/pii/S0377042700004350.

[3] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[4] Benjamin S Kirk et al. "libMesh: a C++ library for parallel adaptive mesh refinement/coarsening simulations". In: *Engineering with Computers* 22.3 (2006), pp. 237–254.

[5] April J. Novak et al. "Pronghorn Theory Manual". In: (Feb. 2018). DOI: 10.2172/1467396. URL: https://www.osti.gov/biblio/1467396.

[6] Cody J. Permann et al. "MOOSE: Enabling massively parallel multiphysics simulation". In: *SoftwareX* 11 (2020), p. 100430. ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2020.100430. URL: http://www.sciencedirect.com/science/article/pii/S2352711019302973.