The ITER1 and HYBRID1 iteration algorithms used in the PYroMat package

Christopher R. Martin
Assistant Professor of Mechanical Engineering
Penn State University

March 3, 2021

1 Introduction

In multi-phase substances, the general equation of state for the evaluation of thermodynamic properties of substances is almost always of the form, $A(T, \rho)$; the Helmholtz free energy expressed as a function of temperature and density. Ideal gases are almost always defined in terms of $c_p(T)$; where c_p is constant-pressure specific heat. For an ideal gas, enthalpy and specific heats do not depend on pressure, but entropy's pressure dependency is straightforward, so ideal gas properties are usually evaluated in terms of temperature and pressure instead of temperature and density.

In general, all of the substances modeled by PYroMatmay be thought of as having properties that are a function of any two of T, p, and ρ. As of version 2.1.0, virtually all property methods in PYroMataccept any two of these three as a constraint on the substance state. For two-phase mixtures (which are possible in the multi-phase collection) "quality" or x may be specified instead of one of these.

The underlying models on which PYroMatis based permit relatively straightforward algorithms for calculating properties in terms of these definitions of a substances "state." There are, however, a number of applications where it is essential to calculate the numerical inverse of one of these properties. As a simple example, the enthalpy of an ideal gas can be calculated directly from its temperature using the thermodynamic model for $h(T)$, but if enthalpy is known, an analytical expression for temperature is not available; numerical iteration is necessary.

1.1 Formulation

The generic problem of property inversion is two-dimensional. There are two arguments, to any property evaluation, so, in theory, specifying any two properties should be sufficient to specify the arguments. In this way, specifying enthalpy and entropy simultaneously should be enough to determine the state.

Like all ideas, this one has its limits too, though. For examples, liquids are so incompressible, that specifying their density doesn't actually do much for our ability to accurately calculate a pressure given a temperature. The uncertainty of such a calculation is huge. This is sometimes called a "numerically stiff" problem, so that a tiny error in the target density would have a huge impact on the calculated pressure. These situations must either be avoided by cautious use of iterative algorithms or mitigated with tremendous precision in the values provided.

In general, there are two property functions that are defined by formulae that cannot be algebraically inverted. In this case, the problem is of the form, given: y_1 and y_1, find: x_1 and x_2 so that

$$y_1 = f_1(x_1, x_2) \tag{1}$$
$$y_2 = f_2(x_1, x_2). \tag{2}$$

2 One-Dimensional Inversion

Many of the available methods restrict these to a powerful subset of the generic problem where one of the two independent variables is known. For example, $T_h()$ methods calculate temperature when entropy and either density or pressure is known. In ideal gas models, enthalpy is calculated as a function of temperature only, $h(T)$, and in multi-phase models, it is calculated as a function of temperature and density, $h(T, \rho)$. If density is known, this amounts to a one-dimensional inversion problem on temperature.

In these cases, the problem ammounts to given: y, find: x such that

$$y = f(x). \tag{3}$$

2.1 Introduction to Newton-Rhapson iteration

Simple Newton-Rhapson iteration is based on a linear approximation for the funciton, f, which assumes that its derivative is readily calculated. For a guess, x_k, the value y can be obtained by moving the guess a distance, Δx,

$$y = f(x_k) + f'(x_k)\Delta x. \tag{4}$$

Figure 1 shows an example iteration step using Newton-Rhapson iteration.

Newton iteration converges quickly on "well mannered" functions that are more or less parabolic in their shape, but there are significant problems when functions show more complicated shapes.

2.2 ITER1: Modified Newton-Rhapson Iteration

In many cases, there are hard boundaries above which no solution is possible. These conditions often occur along with numerically stiff functions like the one

Algorithm 1 Simple Newton-Rhapson iteration, $y = f(x)$

```
Require: A function and its derivative: $f(x)$, $f'(x)$
Require: An initial guess: $x_0$
Require: A target value, $y$, and acceptable error, $\epsilon$
  Initialize the current guess: $x_k \leftarrow x_0$
  Evaluate the function: $y_k \leftarrow f(x_k)$, $y'_k \leftarrow f'(x_k)$
  while Error is large: $|y - f(x_k)| < \epsilon$ do
    Calculate the next guess: $x_k \leftarrow x_k + (y - y_k)/y'_k$
    Evaluate the function: $y_k \leftarrow f(x_k)$, $y'_k \leftarrow f'(x_k)$
  end while
  return : $x_k$
```

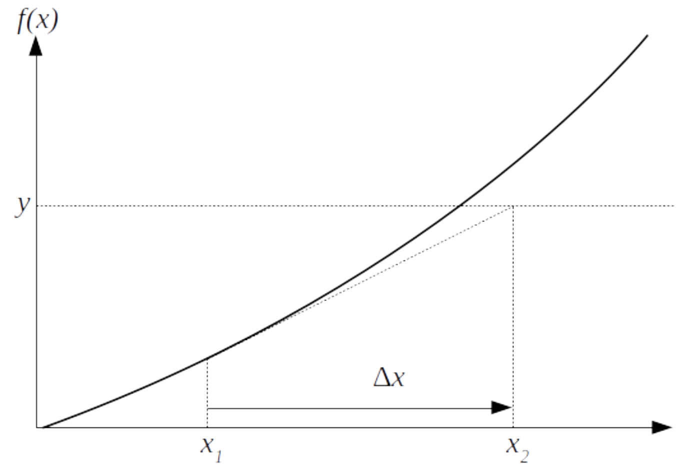

Figure 1: An example iteration step using simple Newton-Rhapson iteration.

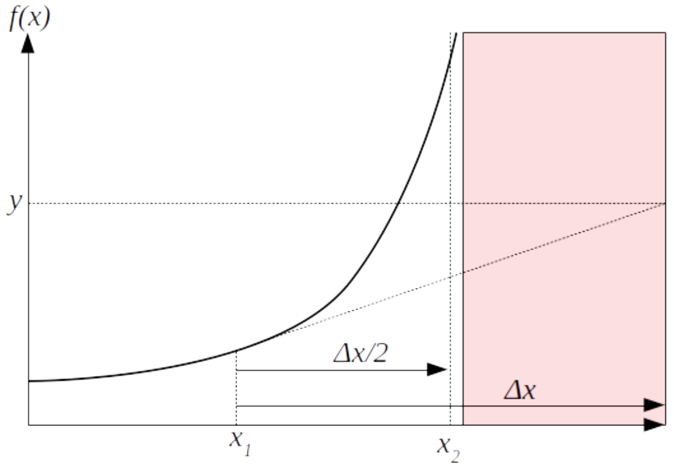

Figure 2: An example of an iteration step using the ITER1 method. The red box represents values beyond the legal solution boundary.

shown in Figure 2. Here the funciton diverges rapidly as it approaches an asymptote.

In many of these cases, the locations of those asymptotes can be known even if the exact solution to the problem is not. The ITER1 algorithm defined in PYroMatis a Newton-Rhapston solver modified to respect upper and lower bounds that limit the range over which an algorithm may stray. If a guess leaves this range, the algorithm halves the step, Δx, until the new guess is back in range.

In Figure 2, the first iteration produces a guess outside of the upper boundary, so Δx is adjusted until the new guess is in-bounds again. In this example, only one adjustment is necessary, but it could have continued many times.

This method is quite robust in the sense that it will either converge to an answer or it will toil on infinitely. Provided the limits on x are well selected, it cannot diverge wildly like the guess in Figure 2.

It should be emphasized, however, that Newton-Rhapson iteration is highly susceptible to infinite cycles when functions are shaped like Figure 3. This depicts an infinite cycle of guesses that will never converge to a solution. It could be disregarded as an esoteric curiosity were it not precisely the type of failure that occurs when iterating on properties near phase changes (or near the critical point of a fluid).

For this reason, the ITER1 algorithm is relegated to the well behaved curves found in ideal gas properties. A more sophisticated algorithm is needed for multi-phase properties. In practice, ITER1 usually converges in three or four iterations to .001% or better on ideal gas data. There are some species that use the Shomate equation that have small discontinuities in their specific heat curves. In extremely rare circumstances, this can cause strange numerical prob-

Algorithm 2 ITER1: Modified bounded Newton-Rhapson iteration, $y = f(x)$

```
Require: A function and its derivative: $f(x)$, $f'(x)$
Require: An initial guess: $x_0$
Require: A maximum and minimum boundary on $x$: $x_a < x < x_b$
Require: A target value, $y$, and acceptable error, $\epsilon$
  Initialize the current guess: $x_k \leftarrow x_0$
  Evaluate the function: $y_k \leftarrow f(x_k)$, $y'_k \leftarrow f'(x_k)$
  while Error is large: $|y - y_k| < \epsilon$ do
    Calculate a step size: $\Delta x \leftarrow (y - y_k)/y'_k$
    Calculate the next guess: $x_{k+1} \leftarrow x_k + \Delta x$
    while $x_{k+1}$ is out of bounds: $x_{k+1} > x_b$ or $x_{k+1} < x_a$ do
      Halve the step size: $\Delta x \leftarrow \Delta x/2$
      Calculate the next guess: $x_{k+1} \leftarrow x_k + \Delta x$
    end while
    Shift the guesses: $x_{k+1} \leftarrow x_k$
    Evaluate the function: $y_k \leftarrow f(x_k)$, $y'_k \leftarrow f'(x_k)$
  end while
  return $x_k$
```

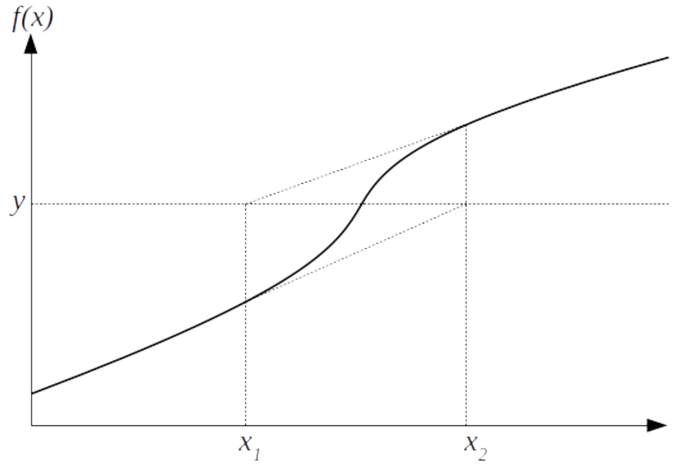

Figure 3: An example of an endless cycle using bisection or ITER1.

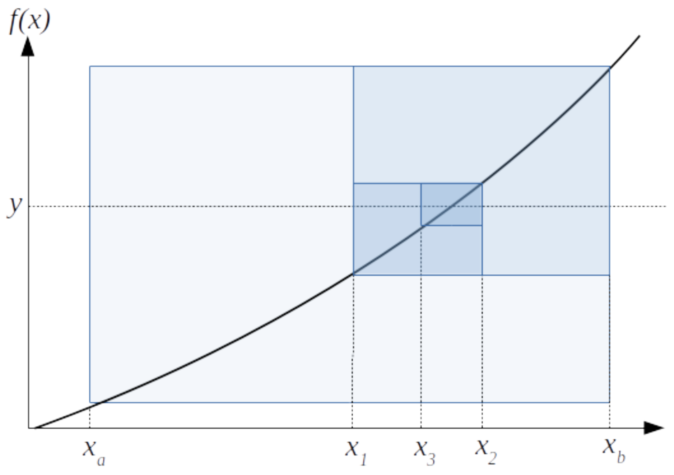

Figure 4: An example of three iteration steps using the bisection algorithm.

lems when the solution lies very near the discontinuity.

2.3 An introduction to bisection iteration

Given that failure to converge can be as bad as crashing, there is some incentive to implement a numerical routine that *must* converge. In one dimension, the bisection algorithm is an excellent choice when stability is valued over speed.

If upper and lower bounds, x_a and x_b are found so that $f(x_a) < y < f(x_b)$, then any continuous function must have a value somewhere between x_a and x_b such that $f(x) = y$. If we were to divide the domain in half $x_c = (x_a + x_b)/2$ and evaluate the function there, its value will either be above or below y, so one of the two values could be replaced by x_c. In this way, the domain between x_a and x_b is reliably cut in half with every iteration step, no matter how bizarrely $f(x)$ behaves.

This process just has to be repeated until the distance between the upper and lower bounds have shrunk to be so small that the numerical uncertainty is acceptable. Figure 4 shows three steps using this process. The vertical space occupied by each blue box represents the shrinking uncertainty for the value of $f(x)$ and the horizontal space represents the shrinking uncertainty in x.

While Newton iteration might converge in only a few steps, bisection can take tens of steps depending on the ratio of the initial domain and the acceptable error range. Since the domain is divided by two every time, it is easy to calculate the number of iterations to obtain a certain domain size.

$$N = \log_2 \frac{|x_b - x_a|}{\epsilon} \tag{5}$$

If the initial domain were a temperature range of 0 to 1000 Kelvin, and the

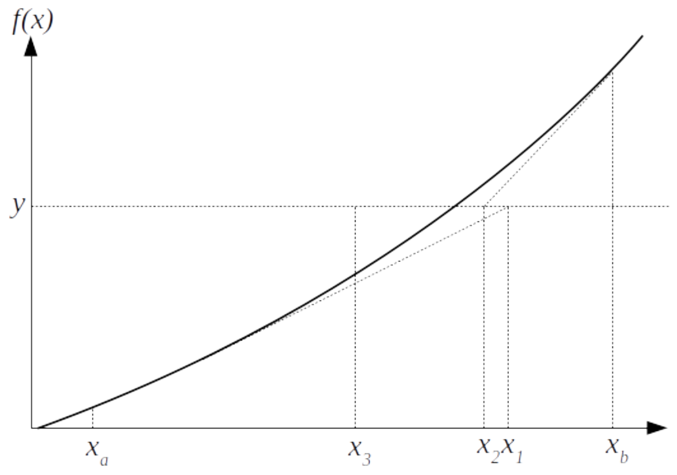

Figure 5: An example of a single iteration step using the HYBRID1 algorithm.

acceptable error were .01 Kelvin, 17 iterations are required. If the acceptable error were tightened to .001 Kelvin, 20 iterations are required!

2.4 HYBRID1

It is difficult to walk away from guaranteed convergence even if bisection iteration is very slow. It would be nice if there were some way to use a Newton-like algorithm that defaulted to a bisection algorithm when things go badly. The trouble is how to figure out when things are going badly without adding numerically expensive checks.

Like bisection, HYBRID1 begins with upper and lower bounds, x_a and x_b, which must bracket a solution. Unlike bisection, HYRBID1 does not assume that all points in the range are equally likely to be solutions. Newton iteration is used as much as possible, but (1) the two bounds must always bracket a solution, and (2) no guess should ever widen the bounds.

Each iteration step is allowed to select one of three candidate guesses. As depicted in Figure 5, two guesses are generated by performing a Newton iteration step at the upper and lower bounds, and a third bisects the domain. The HYBRID1 algorithm selects the middle (or median) of the three.

We may characterize each Newton iteration guess as lying in one of four segments of the x domain:

1. Below the domain, $x < x_a$
2. In the lower half of the domain, $x_a < x < x_3$
3. In the upper half of the domain, $x_3 < x < x_b$
4. Above the domain, $x > x_b$

That represents 16 possible combinations for where the Newton guesses could lie.

There are two cases where both Newton guesses lie in the same half of the domain. This means our next guess will be guaranteed to shrink the domain, and because the two Newton iterations are in agreement, there is an excellent probability that the real solution lies nearby. In this event, the median value will select the Newton iteration nearest the center of the domain - an excellent balance between wanting to shrink the domain as quickly as possible while still benefiting from the speed of Newton iteration.

There are eight cases in which the two Newton guesses lie on oposite sides of the domain center. In all of these situations, Newton iteration seems to provide us no better insight than bisection, so we should fall back on the certainty that the domain will be reduced by half no matter what. In this situation, the median value will be the bisection guess.

There are four cases where the two Newton guesses will be on the same side of the domain center, but one lies outside of the domain. In these situation, the Newton iterations agree on the direction where the solution should be found, but the median value will automatically disregard the one that attempts to leave the domain.

The only degenerate cases are the two where both Newton iterations are outside the domain and on the same side. This can be easily checked by verifying that the median value is inside the domain. If not, the bisection guess should be used instead.

The function and its derivative are only evaluated at the selected candidate value; the other two are left alone. This description of the process may lead one to imagine that each iteration step requires two function evaluations, but that is not so. Just like the bisection algorithm, the appropriate upper or lower bound is replaced by the next guess so that the two bounds continue to bracket a solution. The efficiency of this approach is realized by not discarding the Newton iteration already performed on the other bound. Only one new function evaluation is necessary per step.

The reader is encouraged to walk through the algorithm graphically using Figure 5. It should be apparent that the second iteration step will be a bisection step even though a second Newton iteration should be more expedient. However, the third step arrives remarkably close to the solution. In this way, the cautious approach employed in HYBRID1 carries a cost when employed on well behaved functions.

In practice, HYBRID1 usually converges with one or two extra function evaluations compared with ITER1 on well behaved functions like those found in ideal gas properties. However, in multi-phase systems, ITER1 often fails to converge, and HYRID1 dramatically outperforms traditional bisection.

3 Considerations for Array Support

Algorithm 3 HYBRID1: Hybrid bisection and Newton iteration

```
Require: A function and its derivative: $f(x)$, $f'(x)$
Require: A maximum and minimum boundary on $x$: $x_a < x < x_b$
Require: An acceptable error in $x$, $\epsilon_x$
Require: A target value and acceptable error, $y$, $\epsilon_y$
  Evaluate $f$ at the boundaries:
  $y_a \leftarrow f(x_a)$, $y'_a \leftarrow f'(x_a)$, $y_b \leftarrow f(x_b)$, $y'_b \leftarrow f'(x_b)$
  if $y_a > y_b$ then
    Swap $a$ and $b$:
    $x_a, y_a, y'_a \leftrightarrow x_b, y_b, y'_b$
  end if
  if NOT $y_a < y < y_b$ then
    return ERROR: the solution may not be bracketed.
  end if
  Calculate three candidate guesses:
  $x_1 \leftarrow x_a + (y - y_a)/y'_a$
  $x_2 \leftarrow x_b + (y - y_b)/y'_b$
  $x_3 \leftarrow (x_a + x_b)/2$
  while $|x_b - x_a| > \epsilon_x$ do
    Select a candidate:
    $x_c \leftarrow \mathrm{median}(x_1, x_2, x_3)$
    if NOT $x_a < x_c < x_b$ then
      Force bisection:
      $x_c \leftarrow x_3$
    end if
    Evaluate $f$:
    $y_c \leftarrow f(x_c)$, $y'_c \leftarrow f'(x_c)$
    if $|y_c - y| < \epsilon_y$ then
      return $x_c$
    end if
    if $y_c < y$ then
      Replace the lower bound:
      $x_a \leftarrow x_c$, $y_a \leftarrow y_c$, $y'_a \leftarrow y'_c$
      Calculate a new lower bound guess:
      $x_1 \leftarrow x_a + (y - y_a)/y'_a$
    else
      Replace the upper bound:
      $x_b \leftarrow x_c$, $y_b \leftarrow y_c$, $y'_b \leftarrow y'_c$
      Calculate a new upper bound guess:
      $x_2 \leftarrow x_b + (y - y_b)/y'_b$
    end if
    Calculate a new bisection guess:
    $x_3 = (x_a + x_b)/2$
    return $x_3$
  end while
```