1 Evaluation of functional groups

In the evaluation of empirical fits for thermodynamic properties, it is necessary to calculate complicated functions of many terms on one or two variables. In simple cases like ideal gases, there are standard forms of these equations that can be applied across many species simply by adjusting coefficients (e.g. a polynomial). However, especially when formulating precise relationships for multi-phase substances, whole new terms can appear to address the eccentricities of the specific substances.

There are two pragmatic approaches to address the problem. Classically, special algorithms have been encoded to evaluate each substance or sub-group of substances with similar irregularities. Alternately, we can find a way to encode the functional form in the species data, and use a standard algorithm to evaluate all species. The challenge of the latter is finding an encoding that can still be efficiently evaluated. After all, flexibility in software often comes at the cost of performance.

1.1 Groups on one variable

PYroMat groups are divided in to functions of one or two variables. Beginning with a function of one variable,

$$g_1(x) = \sum_k \prod_m a_4 \, f_{k,m}{}^{a_5}(a_1 x^{a_2} + a_3) + a_6, \quad (1)$$

when f is one of a set of known functions: `exp()`, `log()`, `p()` (a polynomial), or `x` (the identity function). How many terms are included, which functions are used, and the a coefficient values are specific to the property and the species.

At first glance, this looks quite contrived, so it bears some discussion. Let us start with the case that $a_1 = a_2 = a_4 = a_5 = 1$ and $a_3 = a_6 = 0$. Then,

$$g_1(x) = \sum_k \prod_m f_{k,m}(x), \quad (2)$$

which is just a collection of functions of x. It might appear $g(x) = f_1(x)f_2(x) + f_3(x)\ldots$

The group is comprised of a sum of *terms*, each of which is a multiplication of individual functions. The functions could be any of the four available functions. A single $f()$ could even be a polynomial, for which we develop an efficient evaluation algorithm below.

However, Equation 2 is not sufficiently tunable to allow us to construct the wide varieties of terms that appear in many of the property formulations. For that, we resort to pre- and post-modifiers, the a coefficients. Pre-modifiers a_1, a_2, a_3 affect x *before* it is passed as an argument to f. The post-modifiers a_4, a_5, a_6 are used to modify the function's result. These represent an exponent, multiplier, and offset.

The following is a group that can be constructed this way

$$\begin{aligned} g(x) = \exp\left(-x^2\right)\left(0.5x^2 - .012x + .1\right)\ldots \\ + \frac{x^4 - 2}{1.5 + 2\sqrt{x^3 - 2}}\ldots \\ + x^{-0.5}(1 + 0.1x + 0.225x^2) \end{aligned} \quad (3)$$

These may look strange, but all of these are terms that can be found in property models.

1.2 Groups on two variables

An equivalent is needed for properties of two variables.

$$X = a_1 x^{a_2} + a_3 \tag{4a}$$
$$Y = a_4 y^{a_5} + a_6 \tag{4b}$$
$$g_2(x,y) = \sum_k \prod_m a_7 f_{m,k}{}^{a_8}(X,Y) + a_9 \tag{4c}$$

2 Polynomials of two variables

In the evaluaiton of empirical thermodynamic properties, there is often the problem of evaluating polynomial expansions on two variables. These expansions are typically of the form

$$p(x,y) = \sum_{i,j} c_{i,j} x^i y^j \tag{5}$$

where a and b are real coefficients such that i and j are integer indices.

2.1 Modifying polynomials for non-integer and negative powers

Fractional and negative exponents are also possible within this framework if we were to accept input values X, and Y, and adjust them according to pre-exponentials a and b,

$$x = X^a \tag{6}$$
$$y = Y^b. \tag{7}$$

The new polynomial formed is

$$p(X,Y) = \sum_{i,j} c_{i,j} X^{ai} Y^{bj}. \tag{8}$$

For example, were $a = 0.25$, then the expansion on X could proceed in fractions of 4 even though the evaluation algorithm we use is purely expressed in integer exponents. It is relatively computationally inexpensive to apply these exponentials prior to the code's execution. It would be far more costly to apply them to each term.

Similarly, negative exponentials can be achieved by using post-exponential terms

$$P(X,Y) = X^\alpha Y^\beta p(X,Y). \tag{9}$$

When α or β are non-zero, the effective exponents are all shifted positively or negatively by a single multiplication operation.

2.2 Efficient evaluation of the polynomial

The PYroMat polynomial evaluation algorithm is an expansion with purely integer exponents.

$$p(x,y) = \sum_{i,j} c_{i,j} x^i y^j \tag{10}$$

However, evaluating each term individually requires two expensive calls to a `pow` function and two floating point multiplications.

The widely accepted method for evaluating a polynomial of one variable is to construct a recursive expansion

$$q(y) = c_0 + y(c_1 + y(c_2 + y(\dots \tag{11}$$

If there are n coefficients, then this amounts to only m multiplications with no `pow` calls. In order to extend this algorithm to two variables, more elegant notation will be helpful. If we name the intermediate value calculated in the process of these recursions q, then a polynomial with n terms implies the series

$$q_n = c_n \tag{12}$$
$$q_j(y) = c_j + y\, q_{j+1}(y) \tag{13}$$
$$q_0(y) = q(y). \tag{14}$$

This is a series beginning with q_n, and proceeding backwards through the values of j to q_0, which is the final value for $q(y)$. In practice, there is no need to keep the old values of q, so a single register may be used to hold the latest value.

How can this be extended to a polynomial of two variables? We may consider the polynomials to be nested; the evaluation of a polynomial on Y determines the individual coefficients for a polynomial on X.

$$p(x,y) = \sum_i q_i(y) x^i \tag{15}$$

$$q_i(y) = \sum_j c_{i,j} y^j \tag{16}$$

We only need a minor modification to the intermediate values for the x polynomial since there will be a separate expansion for each value of i. If there are n j terms,

$$q_{i,n}(y) = c_{n,j} \tag{17a}$$
$$q_{i,j}(y) = c_{i,j} + y\, q_{i+1,j}(y) \tag{17b}$$
$$q_{i,0}(y) = q_i(y). \tag{17c}$$

If there are m x terms,

$$p_m(x) = q_m(x) \tag{18a}$$
$$p_i(x,y) = q_i(x) + y\, p_{i+1}(x,y) \tag{18b}$$
$$p_0(x,y) = p(x,y). \tag{18c}$$

2.3 Efficient evaluation of derivatives

The partial derivatives of the polynomial can be efficiently evaluated along with the polynomial itself. To relax the already cumbersome notation, the functional dependencies (y) and (x,y) will be dropped. For the purpose of thermodynamic property evaluation, the first two derivatives will suffice.

Let us begin with the simpler task of calculating the derivatives of q_j.

$$q_{i,n|y} = 0 \tag{19a}$$
$$q_{i,j|y} = q_{i+1,j} + y\, q_{i+1,j|y} \tag{19b}$$
$$q_{i,0|y} = q_{i|y} \tag{19c}$$

$$q_{i,n|yy} = 0 \tag{20a}$$
$$q_{i,j|yy} = 2q_{i,j+1|y} + y\, q_{i,j+1|yy} \tag{20b}$$
$$q_{i,0|yy} = q_{i|yy} \tag{20c}$$

The derivatives on p are constructed somewhat differently because they can be in both x and y. Beginning with y,

$$p_{n|y} = 0 \tag{21a}$$
$$p_{j|y} = q_{i|y} + x\, p_{j+1|y} \tag{21b}$$
$$p_{0|y} = p_y \tag{21c}$$

$$p_{m|yy} = 0 \tag{22a}$$
$$p_{i|yy} = q_{i|yy} + x\, q_{i+1|yy} \tag{22b}$$
$$p_{0|yy} = p_{yy} \tag{22c}$$

The derivatives on x appear

$$p_{m|x} = 0 \tag{23a}$$
$$p_{i|x} = p_{i+1} + x\, p_{i+1|x} \tag{23b}$$
$$p_{0|x} = p_x \tag{23c}$$

$$p_{n|xx} = 0 \tag{24a}$$
$$p_{i|xx} = 2p_{i+1|x} + x\, p_{i+1|xx} \tag{24b}$$
$$p_{0|xx} = p_{xx} \tag{24c}$$

Finally, the cross-term (both x and y) appears

$$p_{n|xy} = 0 \tag{25a}$$
$$p_{i|xy} = p_{i+1|y} + y\, p_{i+1|xy} \tag{25b}$$
$$p_{0|xy} = p_{xy} \tag{25c}$$

2.4 Implementation of the algorithm

In practice, this cumbersome notation can be drastically simplified in code because it is not necessary to distinguish between p and q in their various incarnations; provided care is taken not to overwrite a value before it is needed.

In most practical polynomials of two variables of given order, very few of the possible coefficients may be non-zero, so storing and looping over all $m \times n$ coefficients may not be sensible. Instead, it is common to take an approach closer to spare matrix storage.

If we have one-dimensional arrays of polynomial coefficients, c_k, and exponents, a_k and b_k, the polynomial will be constructed as

$$p(X, Y) = \sum_k = 0^{N-1} c_k X^{a_k} Y^{b_k}. \tag{26}$$

In this way, the polynomial

$$p(X, Y) = -0.1X^2 + XY + 0.5Y^2 - Y - 0.2 \tag{27}$$

may be represented by

$$a = [2,\ 1,\ 0,\ 0,\ 0] \tag{28}$$
$$b = [0,\ 1,\ 2,\ 1,\ 0] \tag{29}$$
$$c = [-0.1,\ 1,\ 0.5,\ -1,\ 0.2] \tag{30}$$

For the algorithm to function efficiently, it is reasonable to impose some prior sorting of the exponent values. Since the series above requires that we interact with higher-order terms first, let us assert that the polynomial should be expressed in order of descending exponents on X and then Y.

In Algorithm 1, we employ an outer loop on the values of i and an inner loop on the values of j. If coefficients are absent from the arrays (if the i, j pair is not found), then the coefficient is presumed to be zero. Starting with the maximum value for each exponent, the indices are reduced incrementally until the i, j combination corresponding to the next row (k) is found.

This is extremely practical for polynomials where most of the possible combinations are not represented, and does not cost much for cases where they are.

Algorithm 1 Efficient evaluation of a polynomial of two variables

Require: $a, b, c \in \mathbb{R}^N$

Require: The coefficient order is sorted in descending order of a and then b so that
$a_k \geq a_{k+1} \; \forall \; k \in [0, N-1)$
$b_k > b_{k+1} \; \forall \; k : a_k = a_{k+1}$

Ensure: p, p_x, p_y, p_{xx}, p_{xy}, p_{yy} are equal to the polynomial and its derivatives.

$p, p_x, p_y, p_{xx}, p_{xy}, p_{yy} \leftarrow 0$
$i_{max} \leftarrow a[0]$
$k \leftarrow 0$
for $i = i_{max}$ to 0 **do**
  if $k < N$ **and** a_k equals i **then**
    $j_{max} \leftarrow b_k$
    $q, q_y, q_{yy} \leftarrow 0$
    for $j = j_{max}$ to 0 **do**
      $q_{yy} \leftarrow 2q_y + yq_{yy}$
      $q_y \leftarrow q + yq_y$
      if $k < N$ **and** a_k is i **and** b_k is j **then**
        $q \leftarrow c_k + yq$
        $k \leftarrow k + 1$
      else
        $q \leftarrow yq$
      end if
    end for
    $p_{yy} \leftarrow q_{yy} + xp_{yy}$
    $p_{xx} \leftarrow 2p_x + xp_{xx}$
    $p_{xy} \leftarrow p_y + xp_{xy}$
    $p_x \leftarrow p + xp_x$
    $p_y \leftarrow q_y + xp_y$
    $p \leftarrow q + xp$
  else
    $p_{yy} \leftarrow xp_{yy}$
    $p_{xx} \leftarrow 2p_x + xp_{xx}$
    $p_{xy} \leftarrow p_y + xp_{xy}$
    $p_x \leftarrow p + xp_x$
    $p_y \leftarrow xp_y$
    $p \leftarrow xp$
  end if
end for