Errors in Calculations of Wet-bulb Temperature

Rob Warren and Cass Rogers, Bureau of Meteorology, Melbourne, Australia

19 October 2023

We have identified a set of three bugs in two open-source codes for calculating adiabatic wet-bulb temperature, T_{w}, using the method of Davies-Jones (2008) (hereafter DJ08). The first code is a MATLAB script written by Bob Kopp. This was ported from the implementation of DJ08 in Jonathan Buzan's HumanIndexMod Fortran 90 code back in 2016. The second code is a Python script by Xian-Xiang Li, which was itself ported from the MATLAB code in 2019. Other implementations of the DJ08 method containing the bugs may exist of which we are unaware.

The first bug was likely introduced when the code was ported to MATLAB and vectorized. In defining the initial estimate for T_{w}, the "cold" index array is set in the MATLAB code (on L157) as:

```
cold = ((X>=1).*(X<=D));
```

Here $X = (C/T_{\mathrm{e}})^{\lambda}$, where T_{e} is the equivalent temperature, $C = 273.15$ K, and $\lambda = 3.504$. The same definition is used in the Python code (L307). With this definition, the adjustment of –1.21 to the coefficients k_1 and k_2 is applied where $1 \leq X \leq D$, when it should be applied for $X < 1$ (see Eq. 4.8–4.11 from DJ08). In HumanIndexMod, the condition is correctly specified (on L486–490) as:

```
if ((X >= 1.) .AND. (X <= D)) then
   cold = 0.
else
   cold = 1.
endif
```

The second bug is in the "Q_sat2" function, where $\partial f/\partial T_{\mathrm{w}}$ is calculated. In the MATLAB code, the derivative is specified (L326) as:

```
fdT = -lambd_a.*(1./T_k + vkp.*de_mbdT./pminuse + gdT);
```

The same equation is used in the Python code (L188). However, this is the equation for $\partial(\ln f)/\partial T$ (Eq. A2 from DJ08). To get $\partial f/\partial T_{\mathrm{w}}$, $\partial(\ln f)/\partial T_{\mathrm{w}}$ must be multiplied by f (Eq. A1 from DJ08). This bug was present in the original HumanIndexMod code (and appears in Eq. A20 in Buzan et al. 2015) but was identified by Qinqin Kong and fixed by Jonathan Buzan in late 2020 (see edits on L937–944).

The third "bug" is arguably more of an approximation than a bug per se; however, it turns out to be the most significant in terms of impact on solution accuracy (see below). The MATLAB code defines the initial humidity variables as follows (L83–93):

```
if HumidityMode==0
    qin=Humidity; % specific humidity
    relhum = 100*qin./rs; % relative humidity (%)
    vapemb = es_mb .* relhum * 0.01; % vapor pressure (mb)
elseif HumidityMode==1
    relhum=Humidity;  % relative humidity (%)
    qin = rs .* relhum * 0.01; % specific humidity
    vapemb = es_mb .* relhum * 0.01; % vapor pressure (mb)
end
mixr = qin * grms; % change specific humidity to mixing ratio (g/kg)
```

where `HumidityMode=0` if the input moisture variable is specific humidity, q, in kg/kg, and `HumidityMode=1` if the input moisture variable is relative humidity, RH, in %. The same definitions are used in the Python code (L231–243). There are two errors in these equations:

1. Specific humidity and mixing ratio, r, are assumed to be equivalent (but with the former in kg/kg and the latter in g/kg) when, in fact, $r = q/(1-q)$.

2. Relative humidity is assumed to be related to the mixing ratio and saturation mixing ratio, r_s, by $\mathrm{RH} = r/r_s$. In fact, $\mathrm{RH} = e/e_s$, where e is the vapour pressure and e_s is the saturation vapour pressure. Vapour pressure and mixing ratio are related by $r = \varepsilon e/(p - e)$, where $\varepsilon = R_d/R_v$ is the ratio of the specific gas constants for dry air and water vapour, while vapour pressure and specific humidity are related by $q = r/(1 + r) = \varepsilon e/[p - (1 - \varepsilon)e]$.

The second approximation is also present in HumanIndexMod (L452); however, in this case it represents a deliberate design choice (J. Buzan, personal communication). That code was built to be used with the Community Land Model (Buzan et al. 2015), which defines the relative humidity at 2 m using mixing ratios rather than vapour pressures. In this context, it is accurate to calculate the mixing ratio as $r = r_s \times \mathrm{RH}$. However, for applications where relative humidity is defined in the standard way (i.e., $\mathrm{RH} = e/e_s$), mixing ratio should instead be calculated as $r = \varepsilon e/(p - e)$, where $e = e_s \times \mathrm{RH}$.

The figures below show the impact of the bugs individually and combined on errors in T_w relative to a correct implementation of DJ08. Values were calculated at a pressure of 1000 hPa and are plotted as a function of temperature and relative humidity. Errors for Bugs 1 and 2 are insensitive to the choice of input humidity variable so are only shown for `HumidityMode=0` (Figure 1). On the other hand, errors for Bug 3, and hence the combination of all three bugs, differ substantially between the two `HumidityMode` settings so are plotted for both (Figure 2).

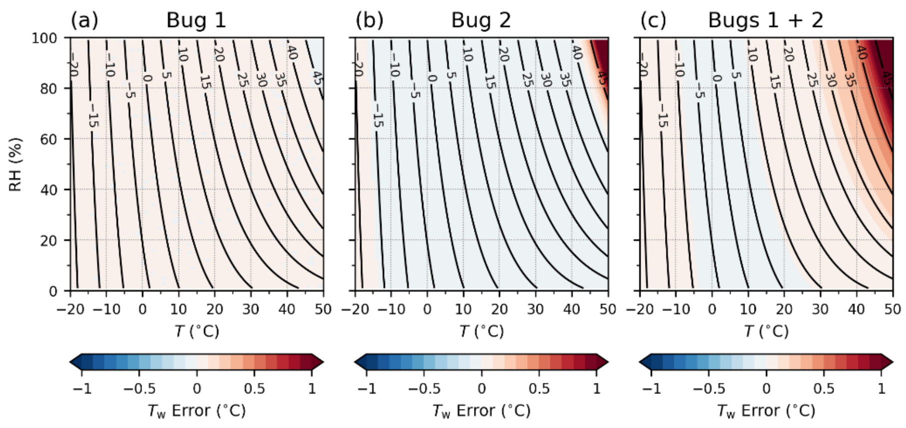

Figure 1. Errors in T_w caused by (a) Bug 1, (b) Bug 2, and (c) Bugs 1 and 2 combined, plotted as a function of temperature, T, and relative humidity, RH. Values are calculated at a pressure of 1000 hPa. Contours show T_w (in °C) from a corrected implementation of DJ08.

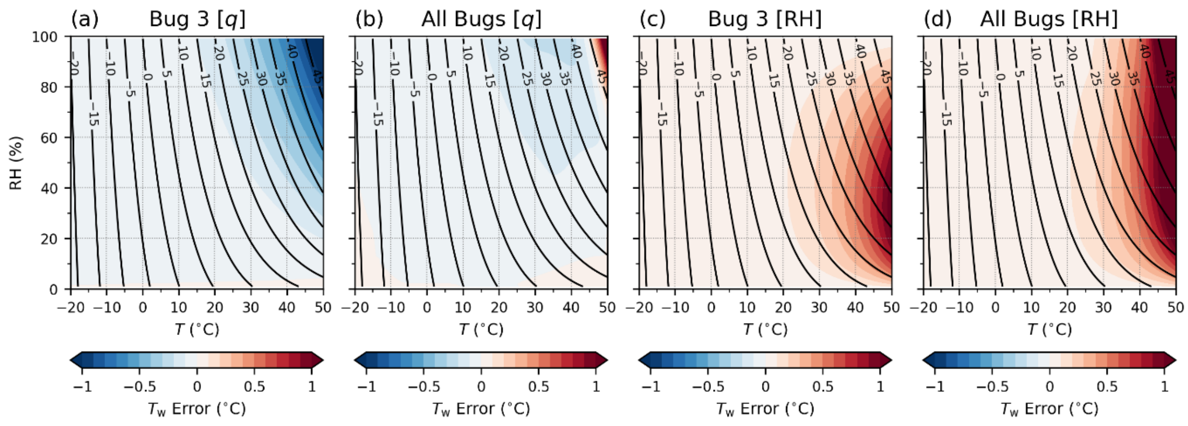

Figure 2. As in Figure 1 but showing errors caused by (a, c) Bug 3 and (b, d) all three bugs combined with (a, b) specific humidity, q, as the input moisture variable (`HumidityMode=0`) and (b, d) relative humidity, RH, as the input moisture variable (`HumidityMode=1`).

In isolation, Bug 1 has little impact on solution accuracy, while Bug 2 only has a substantial impact at extreme values of T_w (> 45°C). When combined, however, the two bugs result in non-negligible errors for T_w > 27°C, which increase nonlinearly with increasing T_w. These errors can be mitigated via additional iterations (not shown), indicating that the bugs slow down convergence of the solution. This motivated the addition of the `ConvergenceMode` option in HumanIndexMod (L502–529), which is replicated in the MATLAB (L190–219) and Python (L342–375) codes. With `ConvergenceMode` on, the errors associated with Bugs 1 and 2 are eliminated. However, in the absence of these bugs, the solution converges to within 0.001°C after just four iterations, even at very high T_w, rendering the `ConvergenceMode` option obsolete. Turning to Bug 3, for `HumdityMode=0` we see an underestimation of T_w at the highest T and RH values. Conversely, for `HumidityMode=1` errors are positive, with maximum magnitudes at high T and moderate RH. Interestingly, for `HumidityMode=0` the combination of all three bugs leads to a cancellation of errors at all but the most extreme T_w values. Conversely, for `HumidityMode=1`, the errors compound, resulting in a large positive bias at high T across almost the full range of RH.

It is worth noting that, in the absence of any bugs, the DJ08 method is very accurate. Figure 3 compares errors in T_w (again at a pressure of 1000 hPa) for the Stull (2011) method, the wet_bulb_temperature function from MetPy (May et al. 2022), a correct implementation of DJ08, and a new method that we have developed called NEWT (Noniterative Evaluation of Wet-bulb Temperature). Each method is compared against an "exact" iterative solution[1] that follows the diagrammatic approach to calculating adiabatic wet-bulb temperature (i.e., lifting a parcel dry adiabatically to saturation and then descending back to the original pressure along a pseudoadiabat). The Stull and MetPy methods show large positive and negative errors, respectively, at high T and low to moderate RH. The Stull method in particular suffers from very large errors in this region, with magnitudes up to 1.3°C. Maximum error magnitudes for the MetPy function are smaller but not insignificant (around 0.4°C). Conversely, maximum error magnitudes for DJ08 and NEWT are only around 0.05°C and 0.01°C, respectively. Note that at this level of precision, choices such as the equation for saturation vapour pressure (SVP) and the values of the heat capacities become significant. The DJ08 method uses the Magnus equation for SVP from Bolton (1980), whereas NEWT uses the analytical equation from Ambaum (2020). The latter is more accurate, particularly for temperatures above 40°C (see Fig. 2 from Ambaum 2020). Ultimately, however, both the DJ08 and NEWT methods are sufficiently accurate for use in studies of humid heat extremes. On the other hand, large biases in the Stull method make it unsuitable for studying these extremes.

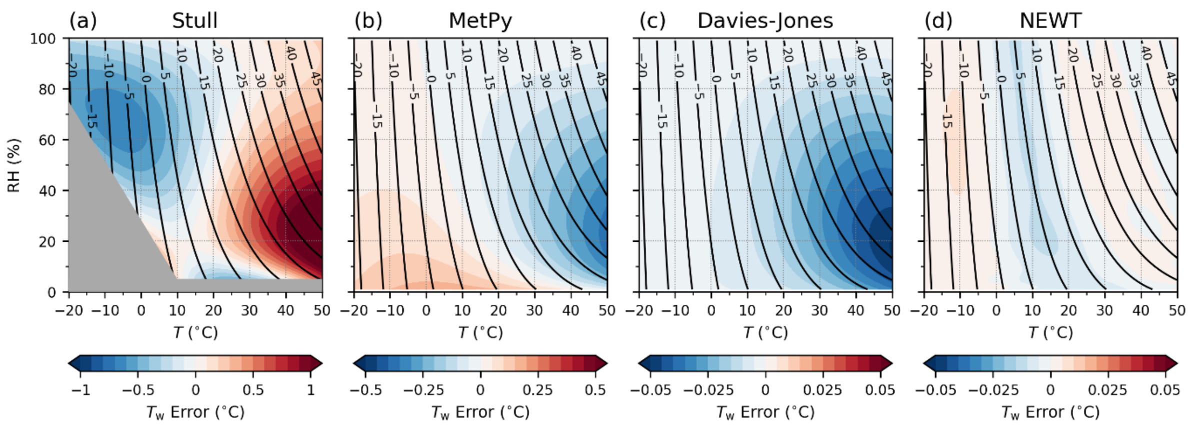

Figure 3. Errors in T_w, relative to an "exact" diagrammatic solution, for calculations using the (a) Stull, (b) MetPy, (c) DJ08, and (d) NEWT methods. Values are calculated at a pressure of 1000 hPa and plotted as a function of temperature, T, and relative humidity, RH. Contours show T_w (in °C) from the diagrammatic solution. Results for the Stull method are masked where the method is considered invalid. Note the different colour scale for each panel.

[1] We say "exact" because even this approach uses the so-called Rankine–Kirchhoff approximations of ideal gases, constant specific heats, and zero specific volume of condensates (Romps 2021).

We note that Bug 3 has also made its way into Dawei Li's Python code for calculating the so-called isobaric wet-bulb temperature. In this code, specific humidity, q, saturation specific humidity, q_s, and relative humidity, RH, are assumed to be related by $\mathrm{RH} = q/q_s$ and q_s is assumed to be equal to the saturation mixing ratio, r_s. Instead, as previously noted, relative humidity is given by $\mathrm{RH} = e/e_s$ and saturation specific humidity by $q_s = r_s/(1 + r_s) = \varepsilon e_s/[p - (1 - \varepsilon)e_s]$. The code also contains a minor error in Bolton's formula for e_s, with 243.15 instead of 243.5 on the denominator.

The errors in isobaric T_w due to Bug 3, shown below in Figure 4, are very similar to those for the DJ08 method; namely, an overestimation of T_w at high T and moderate RH when using relative humidity as the input moisture variable (`h_type='r'`) and an underestimation of T_w at high T and high RH when using specific humidity (`h_type='s'`). On the other hand, when using dewpoint temperature as the input moisture variable (`h_type='d'`), there appears to be some cancellation of errors, resulting in only a small underestimation of T_w at high T and low to moderate RH.

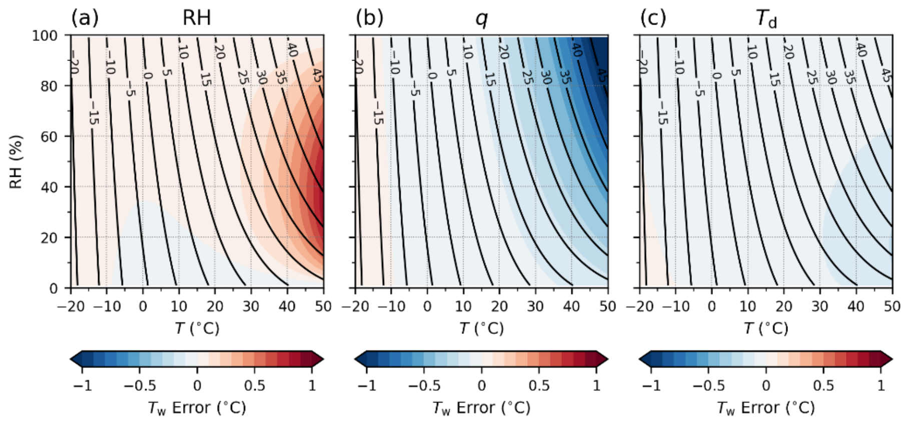

Figure 4. Errors in isobaric T_w caused by Bug 3 for calculations using (a) relative humidity (`h_type='r'`), (b) specific humidity (`h_type='s'`), and (c) dewpoint temperature (`h_type='d'`) as the input moisture variable. Values are calculated at a pressure of 1000 hPa and plotted as a function of temperature, T, and relative humidity, RH. Contours show isobaric T_w (in °C) calculated without Bug 3.

It is important to note that the isobaric wet-bulb temperature is different from the adiabatic wet-bulb temperature. The former is the temperature of a parcel of air cooled isobarically to saturation via evaporation of water into it, while the latter is the temperature of a parcel of air lifted adiabatically to saturation and then brought adiabatically at saturation back to its starting pressure. According to the AMS Glossary the adiabatic wet-bulb temperature is "always less than the isobaric wet-bulb temperature, usually by a fraction of a degree centigrade". To avoid confusion, in the following discussion and in Figure 5 we use T_wi for isobaric wet-bulb temperature and T_wa for adiabatic wet-bulb temperature. It should be emphasized that the Stull, DJ08, MetPy, and NEWT methods, together with our "exact" diagrammatic method, all provide estimates of T_wa rather than T_wi.

Figure 5a compares T_wi, computed using a corrected version of Dawei Li's code, against T_wa, computed using the "exact" diagrammatic method. While T_wi exceeds T_wa across most of the phase space, the sign of the difference is reversed at higher temperatures and humidities, which is inconsistent with the AMS Glossary definition above. This results from inaccuracies in the method used to calculate T_wi. In particular, the method (as described in the supplementary material to Li et al. 2020) neglects the temperature dependence of the latent heat of vaporisation, L_v. It also neglects the contribution of moisture to the isobaric specific heat capacity, c_p, although this is a less severe approximation than constant L_v. A more accurate equation for T_wi can be derived without these approximations and solved using Newton's method (see Appendix). Figure 5b and c compare the resulting T_wi values against T_wa and the approximate T_wi, respectively. We see that the assumptions of constant L_v and c_p lead to an underestimation of T_wi for temperatures above freezing (Figure 5c).

The magnitude of this error increases with increasing T and decreasing RH, reaching a maximum of ~0.4°C. With our more accurate equation, T_{wi} exceeds T_{wa} across the entire phase space, consistent with the definition in the AMS Glossary (Figure 5b). Differences exceed 1°C for high T and very low RH but are smaller at higher humidities where the most extreme wet-bulb temperatures occur. This indicates that the choice of adiabatic versus isobaric wet-bulb temperature is not overly relevant in studies of humid heat extremes.

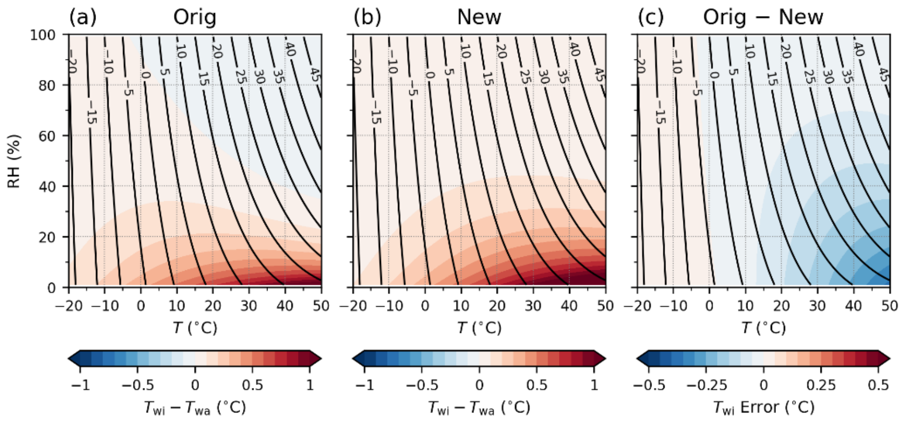

Figure 5. (a) Difference between isobaric wet-bulb temperature, T_{wi}, computed using the original approximate method (without Bug 3), and adiabatic wet-bulb temperature, T_{wa}, from an "exact" diagrammatic solution. (b) As in (a) but for T_{wi} calculated using our new more accurate method. (c) Difference in T_{wi} calculated using the original approximate method and the new more accurate method. Values are calculated at a pressure of 1000 hPa and plotted as a function of temperature, T, and relative humidity, RH. Contours show T_{wi} (in °C) from our more accurate method. Note the different colour scale for (c).

Summary and Recommendations

We have identified a set of three bugs in two open-source implementations of the DJ08 method for calculating adiabatic wet-bulb temperature. Combined, the first two bugs lead to an overestimation of high T_{w} values, which can be mitigated through additional iterations (using `ConvergenceMode` on). Without these bugs, the solution converges to within 0.001°C after just four iterations (rendering the `ConvergenceMode` option obsolete). The impact of the third bug depends on the choice of input moisture variable, with negative errors when using specific humidity (`HumidityMode=0`) and positive errors when using relative humidity (`HumidityMode=1`). This bug has also found its way into an open-source code for calculating the isobaric wet-bulb temperature. Additional errors in the latter code result from the assumption of constant L_{v} and constant c_p. A more accurate equation for isobaric wet-bulb temperature has been derived without these assumptions (see Appendix). In addition, we have developed a new, fast method for calculating adiabatic wet-bulb temperature called NEWT (Noniterative Evaluation of Wet-bulb Temperature), which is even more accurate than DJ08. This method will be presented in a future publication.

Based on these results we offer the following recommendations:

- For calculating adiabatic wet-bulb temperature, we recommend using either a corrected implementation of DJ08 or NEWT. We strongly advise against the use of the Stull method due to its severe overestimation of T_{w} at high temperatures and low to moderate humidities.
- For calculating isobaric wet-bulb temperature, we recommend using the iterative method detailed in the Appendix.

Note that while the adiabatic and isobaric wet-bulb temperature differ substantially at very low relative humidities, the two are within a fraction of a degree Celsius at higher humidities where the most extreme wet-bulb temperatures occur.

Colin Raymond has put together a corrected and Numba-accelerated Python implementation of DJ08. He has also written a blog post in which he discusses the bugs and analyses their impact on the results of his 2020 paper (Raymond et al. 2020). A Python library that includes NEWT and our more accurate method for calculating isobaric wet-bulb temperature will be made available later this year. Readers who have been using the DJ08 method in their work are encouraged to check their codes and correct the bugs if present. We would also recommend revisiting published results that were based on one of the erroneous codes to assess the impact of the bugs on key conclusions.

If you have any questions or comments regarding our analysis please reach out to us via email at rob.warren1@bom.gov.au and cassandra.rogers@bom.gov.au.

Acknowledgements

Thanks to Jonathan Buzan for providing information on the design of HumanIndexMod and Colin Raymond for useful discussions regarding the various implementations of the DJ08 method.

References

Ambaum, M. H. P., 2020: Accurate, simple equation for saturated vapour pressure over water and ice. *Q. J. Roy. Meteor. Soc.*, **146**, 4252–4258, https://doi.org/10.1002/qj.3899.

Bolton, D., 1980: The computation of equivalent potential temperature. *Mon. Wea. Rev.*, **108**, 1046–1053, https://doi.org/10.1175/1520-0493(1980)108<1046:TCOEPT>2.0.CO;2.

Buzan, J. R., K. Oleson, and M. Huber, 2015: Implementation and comparison of a suite of heat stress metrics within the Community Land Model version 4.5. *Geosci. Model Dev.*, **8**, 151–170, https://doi.org/10.5194/gmd-8-151-2015.

Davies-Jones, R., 2008: An efficient and accurate method for computing the wet-bulb temperature along pseudoadiabats, *Mon. Wea. Rev.*, **136**, 2764–2785, https://doi.org/10.1175/2007MWR2224.1.

Li, D., J. Yuan, and R. E. Kopp, 2020: Escalating global exposure to compound heat-humidity extremes with warming. *Environ. Res. Lett.*, **15**, 064003. https://doi.org/10.1088/1748-9326/ab7d04.

May, R. M., K. H. Goebbert, J. E. Thielen, J. R. Leeman, M. D. Camron, Z. Bruick, E. C. Bruning, R. P. Manser, S. C. Arms, and P. T. Marsh, 2022: MetPy: A meteorological Python library for data analysis and visualization. *Bull. Amer. Meteor. Soc.*, **103**, E2273–E2284, https://doi.org/10.1175/BAMS-D-21-0125.1.

Raymond, C., T. Matthews, and R. M. Horton, 2020: The emergence of heat and humidity too severe for human tolerance. *Science Advances*, **6**, eaaw1838, https://doi.org/10.1126/sciadv.aaw1838.

Romps, D. M., 2021: The Rankine–Kirchhoff approximations for moist thermodynamics. *Q. J. Roy. Meteor. Soc.*, **147**, 3493–3497, https://doi.org/10.1002/qj.4154.

Stull, R., 2011: Wet-bulb temperature from relative humidity and air temperature. *J. Appl. Meteor. Climatol.*, **50**, 2267–2269, https://doi.org/10.1175/JAMC-D-11-0143.1.

Appendix: A more accurate equation for isobaric wet-bulb temperature

We start our derivation from the isobaric moist enthalpy equation:

$$c_p \mathrm{d}T = -L_\mathrm{v} \mathrm{d}q$$

Here, c_p is the isobaric specific heat, T is the temperature, L_v is the latent heat of vaporisation, and q is the specific humidity (water vapour mass fraction). To derive an equation for the isobaric wet-bulb temperature (which, for simplicity, we will denote here as T_w), we integrate this equation from the true temperature and specific humidity (T, q) to the wet-bulb temperature and corresponding saturation specific humidity $(T_\mathrm{w}, q_\mathrm{s}(T_\mathrm{w}))$. A common approximation is to assume that c_p and L_v are constants. In this case, we can integrate trivially to obtain

$$c_p(T_\mathrm{w} - T) = -L_\mathrm{v}(q_\mathrm{s}(T_\mathrm{w}) - q)$$

However, c_p and L_v are not constants but functions of specific humidity and temperature, respectively:

$$c_p(q) = (1-q)c_{p\mathrm{d}} + q c_{p\mathrm{v}} = c_{p\mathrm{d}}(1 + q/\gamma - q)$$

$$L_\mathrm{v}(T) = L_{\mathrm{v}0} + (c_{p\mathrm{v}} - c_{p\mathrm{l}})(T - T_0)$$

Here, $c_{p\mathrm{d}}$, $c_{p\mathrm{v}}$, and $c_{p\mathrm{l}}$ are the specific heat capacities for dry air, water vapour, and liquid water, respectively (which are assumed constant), $\gamma = c_{p\mathrm{d}}/c_{p\mathrm{v}}$, $T_0 = 273.16$ K is the triple-point temperature, and $L_{\mathrm{v}0} = 2.501 \times 10^6$ J kg^{-1} K^{-1} is the latent heat of vaporisation at the triple point. Substituting in these definitions and rearranging we obtain

$$\frac{1}{L_{\mathrm{v}0} + (c_{p\mathrm{v}} - c_{p\mathrm{l}})(T - T_0)} \mathrm{d}T = -\frac{1}{c_{p\mathrm{d}}(1 + q/\gamma - q)} \mathrm{d}q$$

Integrating from (T, q) to $(T_\mathrm{w}, q_\mathrm{s}(T_\mathrm{w}))$ gives

$$\frac{1}{c_{p\mathrm{v}} - c_{p\mathrm{l}}} \ln\left[\frac{L_{\mathrm{v}0} + (c_{p\mathrm{v}} - c_{p\mathrm{l}})(T_\mathrm{w} - T_0)}{L_{\mathrm{v}0} + (c_{p\mathrm{v}} - c_{p\mathrm{l}})(T - T_0)}\right] = -\frac{1}{c_{p\mathrm{v}} - c_{p\mathrm{d}}} \ln\left[\frac{c_{p\mathrm{d}}\big(1 + q_\mathrm{s}(T_\mathrm{w})/\gamma - q_\mathrm{s}(T_\mathrm{w})\big)}{c_{p\mathrm{d}}(1 + q/\gamma - q)}\right]$$

or, written more concisely,

$$\frac{1}{c_{p\mathrm{v}} - c_{p\mathrm{l}}} \ln\left(\frac{L_\mathrm{v}(T_\mathrm{w})}{L_\mathrm{v}(T)}\right) = -\frac{1}{c_{p\mathrm{v}} - c_{p\mathrm{d}}} \ln\left(\frac{c_p(q_\mathrm{s}(T_\mathrm{w}))}{c_p(q)}\right)$$

This equation can be solved iteratively using Newton's method. Rearranging we obtain

$$f(T_\mathrm{w}) = \frac{1}{c_{p\mathrm{v}} - c_{p\mathrm{l}}} \ln\left(\frac{L_\mathrm{v}(T_\mathrm{w})}{L_\mathrm{v}(T)}\right) + \frac{1}{c_{p\mathrm{v}} - c_{p\mathrm{d}}} \ln\left(\frac{c_p(q_\mathrm{s}(T_\mathrm{w}))}{c_p(q)}\right) = 0$$

Taking the derivative with respect to T_w

$$f'(T_\mathrm{w}) = \frac{1}{L_\mathrm{v}(T_\mathrm{w})} + \frac{1}{c_p(q_\mathrm{s}(T_\mathrm{w}))} \frac{\mathrm{d}q_\mathrm{s}}{\mathrm{d}T_\mathrm{w}}$$

To obtain an expression $\mathrm{d}q_\mathrm{s}/\mathrm{d}T_\mathrm{w}$ we use the definition of saturation specific humidity:

$$q_\mathrm{s}(T) = \frac{\varepsilon e_\mathrm{s}(T)}{p - (1-\varepsilon)e_\mathrm{s}(T)}$$

where $e_\mathrm{s}(T)$ is the saturation vapour pressure at temperature T, p is the total pressure, and $\varepsilon = R_\mathrm{d}/R_\mathrm{v}$ is the ratio of the specific gas constants for dry air and water vapour. Taking the derivative of this equation with respect to temperature using the quotient rule

$$\frac{\mathrm{d}q_\mathrm{s}}{\mathrm{d}T} = \frac{[p - (1-\varepsilon)e_\mathrm{s}(T)]\varepsilon \frac{\mathrm{d}e_\mathrm{s}}{\mathrm{d}T} + \varepsilon e_\mathrm{s}(T)(1-\varepsilon)\frac{\mathrm{d}e_\mathrm{s}}{\mathrm{d}T}}{[p - (1-\varepsilon)e_\mathrm{s}(T)]^2} = \frac{\varepsilon \frac{\mathrm{d}e_\mathrm{s}}{\mathrm{d}T} + q_\mathrm{s}(T)(1-\varepsilon)\frac{\mathrm{d}e_\mathrm{s}}{\mathrm{d}T}}{p - (1-\varepsilon)e_\mathrm{s}(T)}$$

Using the definition of q_s, we note that

$$\frac{\varepsilon}{p-(1-\varepsilon)e_\mathrm{s}(T)}=\frac{q_\mathrm{s}(T)}{e_\mathrm{s}(T)}$$

and

$$\frac{1}{p-(1-\varepsilon)e_\mathrm{s}(T)}=\frac{q_\mathrm{s}(T)}{\varepsilon e_\mathrm{s}(T)}$$

We also make use of the Clausius–Clapeyron equation:

$$\frac{1}{e_\mathrm{s}(T)}\frac{\mathrm{d}e_\mathrm{s}}{\mathrm{d}T}=\frac{L_\mathrm{v}(T)}{R_\mathrm{v}T^2}$$

Substituting these in and simplifying, we obtain

$$\frac{\mathrm{d}q_\mathrm{s}}{\mathrm{d}T}=q_\mathrm{s}(T)(1+q_\mathrm{s}(T)/\varepsilon-q_\mathrm{s}(T))\frac{L_\mathrm{v}(T)}{R_\mathrm{v}T^2}$$

Setting $T=T_\mathrm{w}$ and substituting back into the equation for $f'(T_\mathrm{w})$, we obtain

$$f'(T_\mathrm{w})=\frac{1}{L_\mathrm{v}(T_\mathrm{w})}+\frac{q_\mathrm{s}(T_\mathrm{w})(1+q_\mathrm{s}(T_\mathrm{w})/\varepsilon-q_\mathrm{s}(T_\mathrm{w}))}{c_p(q_\mathrm{s}(T_\mathrm{w}))}\frac{L_\mathrm{v}(T_\mathrm{w})}{R_\mathrm{v}{T_\mathrm{w}}^2}$$

Defining

$$b(T_\mathrm{w})=\frac{1+q_\mathrm{s}(T_\mathrm{w})/\varepsilon-q_\mathrm{s}(T_\mathrm{w})}{1+q_\mathrm{s}(T_\mathrm{w})/\gamma-q_\mathrm{s}(T_\mathrm{w})}$$

this can be written more succinctly as

$$f'(T_\mathrm{w})=\frac{1}{L_\mathrm{v}(T_\mathrm{w})}+b(T_\mathrm{w})\frac{L_\mathrm{v}(T_\mathrm{w})q_\mathrm{s}(T_\mathrm{w})}{c_{p\mathrm{d}}R_\mathrm{v}{T_\mathrm{w}}^2}$$

Starting from an initial guess for ${T_\mathrm{w}}^{(0)}$, the iterative solution is given by

$${T_\mathrm{w}}^{(n)}={T_\mathrm{w}}^{(n-1)}-\frac{f(T_\mathrm{w})}{f'(T_\mathrm{w})}$$

We can use T for the initial guess; however, slightly faster convergence is achieved by using the mean of the temperature and dewpoint temperature, T_d:

$${T_\mathrm{w}}^{(0)}=\frac{T+T_\mathrm{d}}{2}$$

Testing shows that the solution convergences to within 0.001°C after a maximum of five iterations, for temperatures in the range –20 to 50°C and relative humidities in the range 1 to 99 %. This is considerably faster than Dawei Li's implementation of the approximate equation, which used the bisection method to solve for T_w (see supplementary material to Li et al. 2020).