11 January 2015

TO: Ranadu documentation
FROM: Al Cooper
SUBJECT: Skew-T diagram

1 Introduction and goal

The skew-T diagram (formally, the Skew-T log-P diagram) and related thermodynamic diagrams (tephigram, emagram, pastagram, aerogram, St?ve diagram) are standard tools for analysis of stability in the atmosphere. There are many versions of this diagram available. A fairly recent and very well documented source is this URL, provided by Bret Whissel. Blank diagrams and also source code are available here. The Storm Prediction Center, National Centers for Environmental Prediction, Norman, OK, uses a sounding analysis system called NSHARP, but it has been updated frequently since its introduction and I haven't yet learned the details of how it currently is generated. NCL (see this link) provides a skew-T diagram based on the USAF diagram (form dod-wpc 9-16-1) but this dates back to the 1970s and so probably uses the Rossby form for equivalent potential temperature.

There have been two recent developments that have potential influences on the pseudo-adiabats in the skew-T diagram. First, Davies-Jones (Davies-Jones, R., 2009: On formulas for equivalent potential temperature. Mon. Wea. Review, 137, 3137– 3148) developed an improved representation for the pseudoadiabatic equivalent potential temperature, taking into account factors like the variation of the latent heat of vaporization with temperature and making other adjustments. Second, Murphy and Koop (Q. J. R. Meteorol. Soc. (2005), 131, pp. 1539–1565) developed an improved representation of the equilibrium water vapor pressure as a function of temperature. So far as I have been able to find, these advances have not been incorporated into thermodynamic diagrams that are readily available. Furthermore, any definition of pseudo-adiabatic equivalent potential temperature will likely involve approximations because the specific heat and latent heat of vaporization of water vary with temperature, and the former has not been included even in the Davies-Jones formula. For that reason, the goal here is to construct new diagrams based on those two advances. For the pseudo-adiabatics, the calculation will be based on direct integration of the differential equation representing constant entropy, as discussed in a 2011 memo referenced from the document ProcessingAlgorithms.pdf.

2 The basic equations and general approach

2.1 The coordinate transformation

The basic coordinates in a skew-T log-p diagram are an ordinate that is based on the base-10 logarithm of the pressure and isotherms that are geometrically at 45° slope with respect to both the abscissa and ordinate. This leads to an abscissa coordinate that is a function of both temperature and pressure and is expressed as a value in the range 0–1 (the plot limits):

$$x = \frac{T - T_l}{T_h - T_l} - \frac{\log_{10}(p/p_l)}{\log_{10}(p_l/p_h)} \tag{1}$$

where T and p are the respective temperature [°C] and pressure [hPa] and $\{T_l, T_h\}$ and $\{p_l, p_h\}$ are the respective lower and upper limits for temperature along the abscissa and ordinate. For generating the diagram, this is coded into a function that provides the abscissa for the plot, as follows:

```
####
##  given temperature (deg.C) and pressure (hPa), generates the appropriate abscissa coordinate
##  and returns that along with the original pressure is a vector suitable for plotting.
####
# references tBot, tTop, pBot, pTop in the global environment!! caution!!
XYplot <- function (.T, .p) {
  return (data.frame (X=(.T-tBot) / (tTop-tBot) - log10(.p/pBot) / log10(pBot/pTop), Y=.p))
}
```

2.2 The structure for the plot data and background

A convenient way to represent the values used to construct the plots is in terms of data.frames, with the following structure:

1. Each data.frame has pressure as the first variable and specific values, perhaps 1 or 2 hPa apart, that represent levels for the other variables.

2. Other variables in the data.frame contain the temperature corresponding to a particular value of the variable at the corresponding pressure level in the data.frame. For example, in the data.frame containing mixing ratio, a variable MR10 may appear that contains, at each pressure level, the temperature at which the equilibrium mixing ratio at that pressure would be 10.

3. The line can then be plotted on the skew-T diagram using the coordinates XYplot (DF$MR10, DF$P) where DF is the data.frame containing the results of prior calculation.

4. The structure for the data.frame is as follows:

variable	**columns**	**number of variables**	**first value**	**last value**
Pressure	P (length 1100 to 50 by 5)	1		
Theta	ThetaM100 to ThetaP200	61 (by 5)	T_0-100	T_0+200
Mixing Ratio	MR0.01 to MR30	15 specified levels	0.01 / 1000	30 / 1000
Rossby ThetaE	ThetaEM60 to ThetaE50	23 (by 5)	-60+T_0	50+T_0
DJ ThetaE	ThetaPM60 to ThetaP50	"	"	"
Direct-integration	ThetaIM60 to ThetaI50	"	"	"

The pressure variable determines the number of rows for each of the other variables, so for example each of multiple columns for Theta have the same length as the pressure variable. However, there are 300/5+1=61 Theta rows representing individual values of Theta vs pressure, so for example there is a row of values for Theta=$T_0 - 100$ called ThetaM100 representing the temperature corresponding to that value of Theta at each pressure level. Once this table is constructed, plots can be generated from data in the table without need to repeat the calculations, and the table can be saved as a reference for generating sub-plots that might span only parts of the range covered by the table.

The plot background itself is then constructed from ggplot calls to make it possible to save the generated background for overplotting with data.

2.3 Basic lines: isotherms, isobars, dry adiabats

Plotting the isobars and isotherms is then straightforward. For dry adiabats, solving the equation for potential temperature gives the following equation for the temperature T corresponding to pressure p:

$$T = (T_R)\left(\frac{p}{p_l}\right)^{R_d/c_{pd}} - T_0 \tag{2}$$

where T_R is the reference temperature for the potential-temperature line (in kelvin) at the lower limit for p (or, conventionally, 1000 hPa, where T_R is also the potential temperature) and T_0 is 273.15 K. In this equation, dry-air values are conventionally used for the gas constant (R_d) and the specific heat at constant pressure (c_{pd}), although this can introduce errors of 1 K or more for moist air. For this reason, it may be helpful to plot a range about the dry adiabats to represent this uncertainty, using as an indication of error the value of T the difference between the above value and that obtained for 100% relative humidity, using the relationship

$$\frac{R}{c_p} = \frac{R_d}{c_{pd}(1+\frac{1}{5}\frac{e}{p})}$$

where quantities with subscript d refer to dry air (cf. ProcessingAlgorithms.pdf, p. 38).

```
pBot <- 1100
pRef <- 1000
pTop <- 50
tTop <- 50
tBot <- -50
tMin <- -140
CP <- SpecificHeats ()
RbyCP <- CP[3] / CP[1]
pLevels <- seq (pBot, pTop, by=-5)
tLevels <- seq (tMin, tTop, by=5)
plot (c(0., 1.), c(pTop, pBot), log='y', ylim=c(1000, 100),pch=NA)
```

```
for (p in pLevels) {
  lines (c(0., 1.), c(p, p))
}

for (t in tLevels) {
  x1 <- XYplot (t, pBot)[1]
  y1 <- pBot
  x2 <- x1 + 1
  y2 <- pTop
  lines (c(x1, x2), c(y1, y2), col='blue')
}

## start building data.frame to contain arrays holding lines to be plotted
SkewTData <- data.frame (P=pLevels)
for (theta in seq (TZERO - 100, TZERO + 200, by=5)) {
  lines (XYplot (theta / ((1000/pLevels) ^ RbyCP) - TZERO, pLevels), col='orange')
  SkewTData[sprintf("Theta%.2f", theta)] <- theta / ((1000/pLevels) ^ RbyCP) - TZERO
}
```

2.4 Mixing ratio

Lines representing constant mixing ratio on a thermodynamic diagram represent the equilibrium value at the specified temperature and pressure, and so are often considered in relationship to measurements of dew point in sounding plots. Expressed in units of grams per kilogram of dry air, the mixing ratio MR for moist air in equilibrium with a plane water surface at temperature T and pressure p is

$$\mathrm{MR} = 1000\,\varepsilon \frac{\mathrm{e_s(T)}}{\mathrm{(p - e_s(T))}} \tag{3}$$

where $e_s(T)$ is the equilibrium water vapor pressure. Given a specified value of MR, the value of T giving that mixing ratio at a specified pressure can be found numerically by solving

$$\mathrm{MR} - 1000\,\varepsilon \frac{\mathrm{e_s(T)}}{\mathrm{(p - e_s(T))}} = 0$$

numerically, varying T with p amd MR fixed. The R routine 'nleqslv' is used here to find the temperature, which can be used with (1) to find the plotted abscissa coordinate. This is implemented using this function in the call to 'nleqslv':

```
# note: this omits the enhancement factor, as is conventional def. of eq. vapor pressure
TfromRmix <- function (.T, .rMix, .P) {
  return (.rMix - MixingRatio (MurphyKoop (.T) / .P))
}
```

Using this function with calls to 'nleqslv' for a sequence of pressures gives a sequence of temperatures and hence a sequence of plot points for constructing the diagram.

```
rMix <- c(0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 1:3, 5, seq (10, 30, by=5)) * 0.001
tt <- pLevels  # overwritten below, just a quick allocation
for (rmix in rMix) {
  for (i in 1:length (pLevels)) {
    tt[i] <- nleqslv (10., TfromRmix, jac=NULL, rmix, pLevels[i])$x
  }
  lines (XYplot (tt, pLevels), col='darkgreen', lty=2)
  SkewTData[sprintf("MR%.02f", rmix*1000)] <- tt
}
```

2.5 Pseudo-adiabatic equivalent potential temperature[1]

The representation of pseudo-adiabats can be done in two ways, either via the formula of Davies-Jones (2009), as presented in ProcessingAlgorithms.pdf, p. 50, or by direct integration of the differential equation obtained by setting the total derivative of entropy with pressure to zero. Previous integrations, discussed in the memo referenced above, demonstrate good agreement between these two approaches, but the direct integration is more basic and is the test against which the Davies-Jones formula was developed so that will be used here.

For integrations that include representations of the temperature dependence of the specific heats and latent heat as well as the Murphy and Koop (2006) representation of equilibrium water vapor pressure, the appropriate differential form of the entropy can be used to find the pseudo-adiabatic (or adiabatic) derivative of temperature with respect to pressure. For an adiabatic process where all changes occur in equilibrium, the molar entropy s' is related to temperature T and pressure p (for a perfect gas) via

$$Tds' = c'_p dT - v'dp \tag{4}$$

where primed quantities refer to molar quantities, such that c'_p and $v'$are the molar heat capacity at constant pressure and the molar volume, respectively. There are three contributions to the entropy to consider for a moist air parcel: the entropy of the (i) dry air; (ii) water vapor; and (iii) liquid water. If the respective mole numbers of these three components are n'_d, n'_v, and n'_w, then

$$Tds' = (n'_d c'_{pd} + n'_v c'_{pv} + n'_w c'_w)dT - n'_d v'_d dp_d - n'_v v'_v de + L'_v dn'_v \tag{5}$$

where p_d is the pressure of dry air, e is the water vapor pressure, and L'_v is the molar latent heat of vaporization of liquid water. The last term in (5) arises because there is an entropy change

[1]Sometimes called, incorrectly, pseudo-equivalent potential temperature (it is pseudo-adiabatic in the sense that all water condensate is removed as it appears) or equivalent potential temperature (a term better understood to refer to the true adiabatic process).

associated with the phase change from liquid water to water vapor, and this last term is the heat released by that phase change.[2] The other terms arise from summing (4) for the three individual components.

The mixing ratios r andr_w are, respectively, the masses of water vapor and liquid water per unit mass of dry air:

$$\begin{aligned} r &= \frac{n'_v M_w}{n'_d M_d} \\ r_w &= \frac{n'_w M_w}{n'_d M_d} \end{aligned} \tag{6}$$

where M_w is the molecular weight of water (mass of water per mole) and M_d that of dry air. Dividing (5) by $n'_d M_d T$ and setting ds' to zero for an isentropic process leads to

$$(c_{pd}+rc_{pv}+r_w c_w)\frac{dT}{T}-\frac{v_d}{T}dp_d - r\frac{v_v}{T}de+\frac{L_v}{T}dr=0 \tag{7}$$

where unprimed quantities c_{pd}, c_{pv}, c_w, v_d, v_v, and L_v are specific quantities (i.e., per unit mass of dry air for c_{pd} and v_d and per unit mass of water for c_{pv}, c_w, v_v and L_v; e.g., $c_{pd}=c'_{pd}/M_d$ and $c_{pv}=c'_{pv}/M_w$). For perfect gases, $v_d/T=R_d/p_d$ and $v_v/T=R_w/e$ where R_d and R_w are the gas constants for dry air and water vapor, respectively. Also, the ideal-gas form of the Clausius-Clapeyron equation is

$$\frac{de_s}{e_s}=\frac{L_v dT}{R_w T^2} \tag{8}$$

and Kirchhoff's equation (cf., e.g., Emanuel 1994, Eq. 4,4,3) is

$$dL_V=(c_{pv}-c_w)\,dT\ . \tag{9}$$

With $e=e_s(T)$ and $r=r_s(T)=\frac{M_w}{M_d}e_s(T)/p_d$ as corresponds to a saturated parcel, and with some additional transformations as follow, all terms in (7) can be transformed into differential relationships that only involve derivatives of T and p:

$$\frac{L_v dr}{T}=\frac{d(L_v r)}{T}-r\frac{dL_v}{dT}\frac{dT}{T}=\frac{d(L_v r)}{T}-r(c_{pv}-c_w)\frac{dT}{T}$$

$$rR_w\frac{de_s}{e_s}=rR_w\frac{L_{vdT}}{R_w T^2}=\frac{L_v r}{T}\frac{dT}{T}$$

[2] An additional contribution arises from the entropy increase associated with mixing of the water vapor and the dry air, but this is insignificant and will be neglected.

$$\frac{d(L_v r)}{T} = d\left(\frac{L_v r}{T}\right) + \frac{L_v r}{T}\frac{dT}{T}$$

$$d\left(\frac{L_v r}{T}\right) = \left(\frac{\partial\left(\frac{L_v r}{T}\right)}{\partial T}\right)_{p_d} dT + \left(\frac{\partial\left(\frac{L_v r}{T}\right)}{\partial p_d}\right)_T dp_d = \frac{\varepsilon T\, d\left(\frac{L_v e_s(T)}{T}\right)}{p_d dT}\frac{dT}{T} - \frac{L_v r}{T}\frac{dp_d}{p_d}$$

where the transformation to dependence on T is a consequence of assuming that the parcel remains saturated, and where the temperature dependence of L_v and the specific heats is implicit. Gathering terms in (7) after these transformations leads to

$$\left[(c_{pd} + r_t c_w) + \frac{T\varepsilon}{p_d}\left(\frac{\partial\left(\frac{L_v e_s(T)}{T}\right)}{\partial T}\right)_{p_d}\right]\frac{dT}{T} = \left[R_d + \frac{L_v r}{T}\right]\frac{dp_d}{p_d}$$

where $r_t = r + r_w$ is the total water mixing ratio and where $\varepsilon = M_w/M_d$. The result then gives a relationship between T and p_d:

$$\frac{dT}{dp_d} = \frac{TR_d + L_v r}{p_d}\left[(c_{pd} + r_t c_w) + \frac{T\varepsilon}{p_d}\left(\frac{\partial\left(\frac{L_v e_s(T)}{T}\right)}{\partial T}\right)_{p_d}\right]^{-1} \tag{10}$$

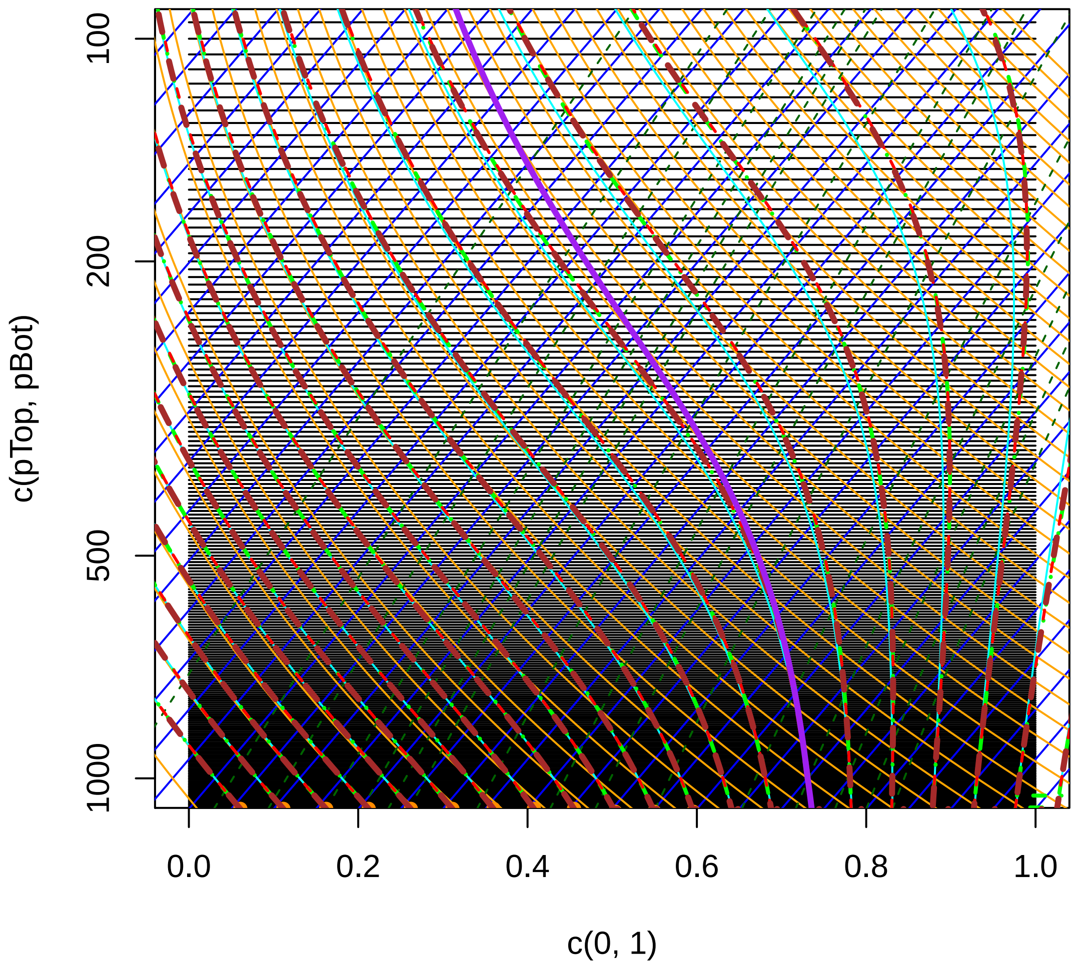
The result is a derivative that can be used for numerical integrations that take into account the temperature dependence of the specific heats and the latent heat of vaporization and improved representation of the equilibrium vapor pressure $e_s(T)$, as in Murphy and Koop (2006). This is also used below to evaluate the accuracy of representations of the equivalent potential temperatures.

Equation (10) is appropriate for the adiabatic process and so can lead to the wet-equivalent potential temperature Θ_q. The similar formula for the pseudo-adiabatic equivalent potential temperature Θ_p can be obtained by neglecting the heat capacity of the liquid water, and so would be the same as (10) but with r_t replaced by r.

The pseudo-adiabats are then constructed by integration of (10) either upward from a fixed starting point at 1000 hPa (which would represent the wet-bulb pseudo-adiabatic potential temperature) or starting from a pressure where humidity is negligible (which would result in pseudo-adiabats labeled by their value equivalent to potential temperature after all water is condensed). Here the former is chosen. The integration is performed using the following R code:

```
# this chunk includes functions including one to integrate between two levels
# if the specific-heat-of-liquid-water data have not been read, read and store them
if (file.exists ("./SpecificHeatWater.Rdata")) {
  load ("./SpecificHeatWater.Rdata") # reloads cw.table
} else {
  cw.table <- read.table ("./MurphyKoopFig6.txt", sep=',', col.names=c("ID", "T", "cw"))
  cw.table$cw[cw.table$ID == 'n'] <- cw.table$cw[cw.table$ID == 'n'] / StandardConstant("MWW")
  save (cw.table, file="./SpecificHeatWater.Rdata")
}
CWData <- cw.table [ , 2:3]
CWData <- CWData[order(CWData), ]
CWData <- CWData[!is.na(CWData[ ,1]), ]
load ("./CPV.Rdata")

LatentHeatApprox <- function (.T) {  # input in deg.C
  return (2.501e6 - 2370 * .T)
}
CPWaterVapor <- function (.T) {
  tk <- .T + TZERO
  # note that there was an error in this formula in the earlier ThetaE memo, 3 Jan 2011
  return (1000 * LagrangeInterpolate (tk, 4, CPV))
}
SpecificHeatLiquidWater <- function (.T) {
  tk <- .T + TZERO
  return (1000. * LagrangeInterpolate (tk, 4, CWData))
}
# not used; here for reference
LCLfn <- function (.p, RbyCP, thetam, mr) {  # used by LCL function call to nleqslv
  tt <- thetam / (1000/.p)^RbyCP
  ee <- MurphyKoop (tt-TZERO)
  return (ee - mr * .p / (mr + StandardConstant("MWW")/StandardConstant("MWD")))
}

# not used; here for reference; for pseudoadiabat, always at 100% RH
LCL <- function (.RH, .T, .p) {  # .RH in a fraction (not %), .T in deg.C, .p in hPa
  et <- .RH * MurphyKoop (.T)
  mr <- MixingRatio (et/.p)
  CPM <- SpecificHeats (et/.p)
  RbyCP <- CPM[3] / CPM[1]
  thetam <- .T+TZERO * (1000 / .p)^RbyCP
  pLCL <- nleqslv (.p, LCLfn, jac=NULL, RbyCP, thetam, mr)
  tLCL <- thetam / (1000/pLCL)^RbyCP
  return (data.frame ("pLCL"=pLCL, "tLCL"=tLCL))
}

deriv1 <- function (.T, .LV) { # evaluate last partial deriv. in (10)
  delt <- 0.1
  tc <- .T + delt
  LVp <- .LV + (CPWaterVapor (tc) - SpecificHeatLiquidWater (tc)) * delt
  esp <- MurphyKoop (tc)
  delp <- LVp * esp / (.T + delt + TZERO)
  tc <- .T - delt
  LVm <- .LV - (CPWaterVapor (tc) - SpecificHeatLiquidWater (tc)) * delt
  esm <- MurphyKoop (tc)
  delm <- LVm * esm / (.T - delt + TZERO)
  return ((delp-delm) / (2*delt))
}

dTdpdF <- function (.T, .pd, .LV, .rtot=NA, .aflag=FALSE) {
  e <- MurphyKoop (.T)
  p <- .pd + e
  if (.aflag) {  # adiabatic
    rx <- .rtot
  } else {
    rx <- MixingRatio (e/p)
  }
  tk <- .T + TZERO
  cp_x <- CP[1] + rx * SpecificHeatLiquidWater (.T)
  A <- ((tk * SpecificHeats(0)[3] + .LV * MixingRatio (e/p)) / .pd) /
      (cp_x + (EPS * tk / .pd) * deriv1 (.T, .LV))
  return (A)
}

# integration step: call repeatedly to construct plottable profile
# tk Initial temperature, kelvin
# p1 Initial pressure, hPa
# p2  Final pressure, hPa
# value returned, temperature at p2
# aflag  FALSE for pseudoadiabatic, adiabatic otherwise
IntegrationStep <- function (tc, pd1, pd2, rtot, aflag) {
  nsteps <- 5
  delpd <- (pd1 - pd2) / nsteps
  pd <- pd1
  tk <- tc + TZERO
  e <- MurphyKoop (tc)
  p <- pd + e
  r <- MixingRatio (e/p)

  # ready to calculate the derivative dT/dp_d, adiabatic and pseudo-adiabatic cases
  while (pd > pd2) {
    if ((pd-delpd) < pd2) {delpd <- pd-pd2}
    dTdpd <- dTdpdF (tc, pd, LV, rtot, aflag) # wet-adiabatic version if aflag TRUE
    tch <- tc - dTdpd * delpd/2
    ex <- MurphyKoop (tch)
    LVh <- LV - (CPWaterVapor (tch) - SpecificHeatLiquidWater (tch)) * dTdpd * delpd / 2
    dTdpd <- dTdpdF (tch, pd-delpd/2, LVh, rtot, aflag)
    tc <- tc - dTdpd * delpd  # take full step with derivative evaluated half-step
    pd <- pd - delpd
    LV <- LV + (CPWaterVapor (tch) - SpecificHeatLiquidWater (tch)) * dTdpd * delpd
  }
  return (tc)
}
```

```
pBot <- 1100
pTop <- 50
tTop <- 50
tBot <- -50
tMin <- -140
CP <- SpecificHeats ()
RbyCP <- CP[3] / CP[1]
pLevels <- seq (pBot, pTop, by=-5)
tLevels <- seq (tMin, tTop, by=5)
plot (c(0., 1.), c(pTop, pBot), log='y', ylim=c(1000, 100),pch=NA)
for (p in pLevels) {
  lines (c(0., 1.), c(p, p))
}

for (t in tLevels) {
  x1 <- XYplot (t, pBot)[1]
  y1 <- pBot
  x2 <- x1 + 1
  y2 <- pTop
  lines (c(x1, x2), c(y1, y2), col='blue')
}

## start building data.frame to contain arrays holding lines to be plotted
SkewTData <- data.frame (P=pLevels)
for (theta in seq (TZERO - 100, TZERO + 200, by=5)) {
  lines (XYplot (theta / ((1000/pLevels) ^ RbyCP) - TZERO, pLevels), col='orange')
  SkewTData[sprintf("Theta%.2f", theta)] <- theta / ((1000/pLevels) ^ RbyCP) - TZERO
}
rMix <- c(0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 1:3, 5, seq (10, 30, by=5)) * 0.001
tt <- pLevels  # overwritten below, just a quick allocation
for (rmix in rMix) {
  for (i in 1:length (pLevels)) {
    tt[i] <- nleqslv (10., TfromRmix, jac=NULL, rmix, pLevels[i])$x
  }
  lines (XYplot (tt, pLevels), col='darkgreen', lty=2)
  SkewTData[sprintf("MR%.02f", rmix*1000)] <- tt
}

## first show Rossby solution
TWB <- seq (tBot, tTop, by=5)
TfromEPT <- function (.T, .thetaE, .P) {
#  e <- MurphyKoop (.T)
#  r <- MixingRatio (e / .P)
#  lhv <- 2.501e6 - 2370. * .T    # latent heat of vaporization, temp-dependent
#  expn=lhv * r / (CP[1] * (TZERO + .T))
#  return (.thetaE - (TZERO+.T)*(1000/(.P-e))^RbyCP * exp(expn))
  return (.thetaE - RossbyEquivalentPotentialTemperature (.P, .T))
}

# Bolton form
TBfromEPT <- function (.T, .ThetaB, .P) {
  return (.ThetaB - BoltonEquivalentPotentialTemperature (.P, .T))
}

# this is solution of Davies-Jones formula; do integration later
DJTfromEPT <- function (.T, .thetaE, .P) {
  return (.thetaE - EquivalentPotentialTemperature (.P, .T))
}
tt <- pLevels  # shortut to define new vector
for (twb in TWB) {
  thetaR <- RossbyEquivalentPotentialTemperature (1000, twb)
  for (i in 1:length(pLevels)) {
    tt[i] <- nleqslv (10., TfromEPT, jac=NULL, thetaR, pLevels[i])$x
  }
  lines (XYplot (tt, pLevels), col='cyan')
  SkewTData[sprintf("ThetaR%.02f", twb+TZERO)] <- tt
}

for (twb in TWB) {
  thetaB <- BoltonEquivalentPotentialTemperature (1000, twb)
  for (i in 1:length(pLevels)) {
    tt[i] <- nleqslv (10., TBfromEPT, jac=NULL, thetaB, pLevels[i])$x
  }
  lines (XYplot (tt, pLevels), col='green', lwd=2, lty=4)
  SkewTData[sprintf("ThetaB%.02f", twb+TZERO)] <- tt
}


for (twb in TWB) {  # Davies-Jones form
  thetaP <- EquivalentPotentialTemperature (1000, twb, MurphyKoop (twb))
  for (i in 1:length(pLevels)) {
    tt[i] <- nleqslv (10., DJTfromEPT, jac=NULL, thetaP, pLevels[i])$x
  }
  lines (XYplot (tt, pLevels), col='red', lty=2, lwd=1.5)
  SkewTData[sprintf("ThetaP%.02f", twb+TZERO)] <- tt
}


# now add lines obtained by integration:
CP <- SpecificHeats(0)
for (twb in TWB) {
  thetaP <- EquivalentPotentialTemperature (1000, twb)
  ttt <- nleqslv (10., DJTfromEPT, jac=NULL, thetaP, pBot)$x
  # get appropriate starting point in dry-air pressure
  pB <- pBot
  while ((p1 <- pB - MurphyKoop (ttt)) < pLevels[2]) {
    pB <- pB + 10
    ttt <- nleqslv (10., DJTfromEPT, jac=NULL, thetaP, pB)$x
  }
  ptLevels <- pLevels
  ptLevels[1] <- pB
  rtot <- MixingRatio (MurphyKoop (ttt) / pB)
  LV <- LatentHeatApprox (ttt)
  tt[1] <- ttt
  points (XYplot (ttt, pB), pch=16, col='darkorange')
  for (i in 1:(length(pLevels)-1)) {
    tt[i+1] <- IntegrationStep (tt[i], p1, pLevels[i+1], rtot, aflag=FALSE)
    p1 <- pLevels[i+1]
    tch <- (tt[i+1]+tt[i])/2
    LV <- LV + (CPWaterVapor (tch) - SpecificHeatLiquidWater (tch)) * (tt[i+1]-tt[i])
  }
  ptLevels <- pLevels + MurphyKoop(tt)
  ptLevels[1] <- pBot
  ## now interpolate to get tt interpolated to pLevels, for data.frame:
  DI <- data.frame (ptLevels, tt)
  tti <- tt
  for (i in 1:(length(pLevels))) {
    tti[i] <- LagrangeInterpolate (pLevels[i], 4, DI)
  }
  lines (XYplot (tt, ptLevels), col='brown', lty=2, lwd=3)
## this needs correction for diff pdry vs p
  SkewTData[sprintf("ThetaI%.02f", twb+TZERO)] <- tti
}

# add a wet-adiabat

TQfromWBPT <- function (.T, .ThetaQ, .P, .rtot) {
  e <- MurphyKoop (.T)
  r <- MixingRatio (e / .P)
 lwc <- ifelse ((.rtot > r), 1000 * (.rtot - r) * (100 * (.P-e) /
                            (SpecificHeats(0)[3] * (.T+TZERO))), 0.)
  return (.ThetaQ - WetEquivalentPotentialTemperature (.P, .T, 0, lwc))
}

ThetaQ <- WetEquivalentPotentialTemperature (1000, TWB[15])
rTot <- MixingRatio (MurphyKoop (TWB[15]) / 1000)
for (i in 1:length (pLevels)) {
  tt[i] <- nleqslv (10., TQfromWBPT, jac=NULL, ThetaQ, pLevels[i], rTot)$x
}
lines (XYplot (tt, pLevels), col='purple', lty=1, lwd=3)
```



```r
save(SkewTData, file="SkewTData.Rdata")
```

3 Constructing the Skew-T background via ggplot

3.1 Reasons for using ggplot

For the plot generated in the preceding sections, the "base" graphics of R have been used. The plot package 'ggplot2' is better suited to generation of the background for a skew-T plot because it enables generation via layers and preserving the results in a reusable form. Layers can represent the isobars, isotherms, pseudo-adiabats, etc, and for subsequent use as the background for a sounding the result can be generated by the preserved plot definition, with the actual sounding over-plotted as an additional layer. That is the approach taken in the remainder of this document.

3.2 The basic background

Here are some variables that determine the range covered by the plot. These can be changed to give plots covering different areas, although some work is still needed to get that to work for other than the standard limits here:

```
####
##   Here are characteristics for the plot, which can be changed as desired.
##   Some other characteristics, like definition of which mixing-ratio lines
##   to plot, have already been determined when the data file was
##   generated above; see, for example, MR[] there.

##   background color for plot area
bColor <- "gray95"
bColor <- "lightyellow"
bColor <- "lemonchiffon1"
bColor <- "cornsilk"
bColor <- "aliceblue"
bColor <- "ivory"
##   color for pressure lines and plot border, and thickness of lines
pColor <- 'darkblue'          # pressure lines
plwd <- 0.4                   # line width for pressure lines
## color for isotherms
tColor <- 'darkblue'          # temperature lines
tlwd=0.4                      # line width for temperature lines

pBot <- 1000                  # range in pressure and temperature for the plot
pTop <- 100                   # (for GV)
tBot <- -40
tMin <- -140                  # needed to cover isotherms in top-left of plot
pTop <- 300                   # (for C-130)
tBot <- -20
tMin <- -120
tTop <- 40                    # both
pLevels <- seq (pBot, pTop, by=-50)     # lines to plot for pressure
tLevels <- seq (tMin, tTop, by=5)       # lines to plot for temperature
####
##   end of changeable plot characteristics
####
```

```
require (ggplot2)
require (ggthemes)
require (grid)
library (scales)
```

```
## the following is already in memory if this program is run from the beginning, but
## some of the items here including the following "load" of the generated data are
## models for how the generated information can be used. Note that the following datafile
## contains some variables that are not used below but are present because they were
## generated for comparison to the variables actually used.
load ("./SkewTData.Rdata")
## read a file for access to sample data
```

For convenience in determining the correspondence between {T, P} coordinates and plot coordinates, the basic transformation function used above is redefined here, with the same name as before. The reason is that there were problems defining a plot with ggplot that had both logarithmic and reverse-order coordinates, as needed for pressure, so instead of using that scale definition here a linear scale is used but values are converted to logarithms before plotting. For that purpose, the XYplot function is redefined to return the logarithm of the pressure rather than the pressure as the second coordinate.

```
XYplot <- function (.T, .p) {
  return (data.frame(X=(.T-tBot) / (tTop-tBot) - log10(.p/pBot) / log10(pBot/pTop),
}
```

Here is code to construct the ggplot definition, with reference to the data already generated for the isopleths:

```
NPL <- length (pLevels)
XP <- vector()
YP <- vector()
for (p in pLevels) {  # in general, this is poor, a very slow method in R. But ...
  lp <- log10(p)
  XP <- c(XP, c(0, 1, NA))
  YP <- c(YP, lp, lp, 0)
}
XYP <- data.frame (X=XP, Y=YP)
XYP2 <- XYP           # 2nd data.frame to hold line definition skipping even levels,
                      # for plotting solid pressure lines instead of dashed
sq <- 4:5             # assemble a vector of points in XYP2 to skip
j <- 1
while (4+6*j+1 <= length (XYP2$X)) {
  sq <- c(sq, (4+6*j):(4+6*j+1))
  j <- j + 1
}
XYP2$X[sq] <- NA      # this forces the flagged points to be skipped

# Define g to hold the plot definition and the pressure lines
g <- ggplot (data=XYP, aes(x=X, y=Y)) + ylim(log10(pBot), log10(pTop))
         # set a background color
g <- g + theme(panel.background = element_rect(fill = bColor))
         # plot the pressure lines, first dashed, then solid
g <- g + geom_path (data=XYP, aes(x=X, y=Y), color=pColor, lty=2)
```

```
g <- g + geom_path (data=XYP2, aes(x=X, y=Y), color=pColor, lty=1)
         # suppress auto grid lines and axis labels and ticks
g <- g + theme(panel.grid.major = element_blank())
g <- g + theme(panel.grid.minor = element_blank())
g <- g + theme (axis.text = element_blank ())
g <- g + theme (axis.ticks = element_blank ())
         # specify characters for axes
g <- g + theme (axis.title=element_text(face="plain",size=12,colour="blue"))
         # add a border to complete the box around the plot
border <- data.frame (X=c(0,0,NA,1,1),
                      Y=c(log10(pBot),log10(pTop),log10(pBot),log10(pBot),log10(pTop)))
g <- g + geom_path (data=border, color=pColor, lwd=plwd)
        # write the axis labels
g <- g + xlab (expression (paste("Temperature or dew point [",degree,"C]")))
g <- g + ylab ("Pressure [hPa]")
         # label the pressure lines, all in one geom_text call with data in DLPLS
pls <- seq (pBot, pTop, by=-100)
LPLS <- length(pls)
labl <- character()
for (i in 1:LPLS) {labl <- c(labl, sprintf("%d", pls[i]))}
DLPLS <- data.frame (X=rep(-0.025, LPLS), Y=log10(pls), LABEL=labl)
g <- g + geom_text (data=DLPLS, aes(x=X, y=Y, label=LABEL), size=4)
         # draw the isotherms
xt <- vector ()
yt <- vector ()
YR <- log10 (pBot/pTop)
for (t in tLevels) {
  XPb <- XYplot (t, pBot)        # (x,y) at bottom axis
  XPt <- XYplot (t, pTop)        # (x,y) at top axis
  if (XPt$X <= 0) {next}         # skip lines out of range
  if (XPb$X >= 1) {next}
  if (XPb$X < 0) {               # truncate lines to stay within plot area
    XPb$Y <- XPb$Y+XPb$X * YR
    XPb$X <- 0
  }
  if (XPt$X > 1) {
    XPt$Y <- XPt$Y + (XPt$X - 1) * YR
    XPt$X <- 1
  }
  xt <- c(xt, c(XPb$X, XPt$X, NA))
  yt <- c(yt, c(XPb$Y, XPt$Y, NA))
}
DT <- data.frame (X = xt, Y = yt)
DT2 <- DT
g <- g + geom_path (data=DT, aes(x=X, y=Y), color=tColor, lwd=tlwd, lty=2)
sq <- 4:5
j <- 1
while (4+6*j+1 <= length(DT2$X)) {
  sq <- c(sq, (4+6*j):(4+6*j+1))
  j <- j + 1
}
```

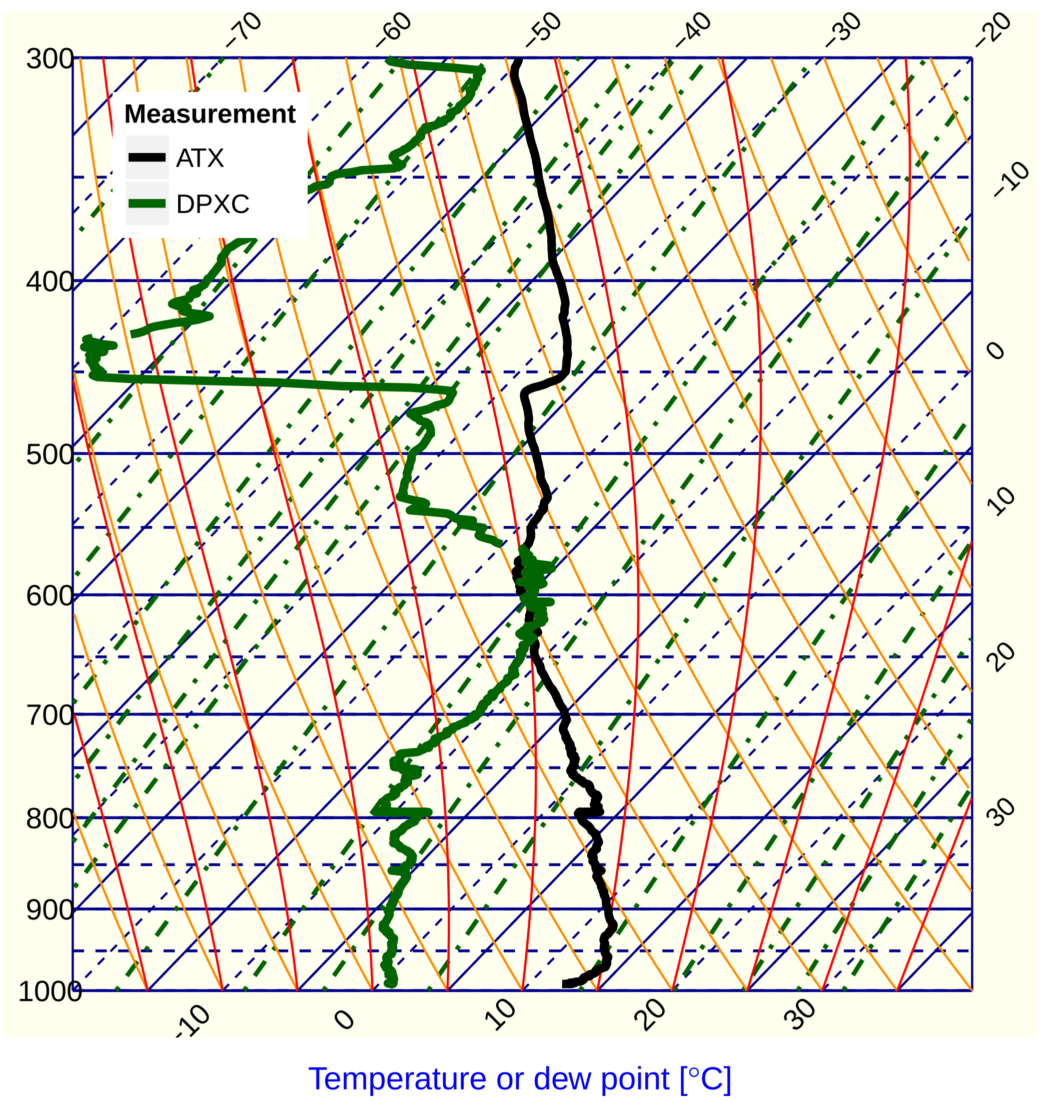
```
DT2$X[sq] <- NA
g <- g + geom_path (data=DT2, aes(x=X, y=Y), color=tColor, lwd=tlwd, lty=1)

## add text for temperature, labeling every other line
pltt <- seq (tLevels[3], tLevels[length(tLevels)-2], by=2*(tLevels[2]-tLevels[1]))
LPLT <- length (pltt)
lablt <- character()
for (i in 1:LPLT) {lablt <- c(lablt, sprintf("%d", pltt[i]))}
lablt[lablt=="0"] <- "0  "          # move the 0 to look better
DLPLT <- data.frame (X=XYplot (pltt, pBot)$X, Y=rep(log10(pBot), LPLT), LABEL=lablt)
DLPLT2 <- DLPLT [(DLPLT$X > 0), ]
g <- g + geom_text (data=DLPLT2, aes(x=X, y=Y, label=LABEL, angle=45, hjust=1.5, vjust=0.5),
                    size=4)
g <- g + geom_text (data=DLPLT2, aes(x=1.02, y=Y-(1-X) * YR, label=LABEL, angle=45, hjust=-0.3, vjust=-(
                    size=3.5)
DLPLT3 <- DLPLT[(DLPLT$X <= 0), ]
DLPLT3 <- DLPLT3[(DLPLT3$X > -1), ]
g <- g + geom_text (data=DLPLT3, aes(x=X+1, y=log10(pTop), label=LABEL, angle=45, hjust=-0.2, vjust=0.2)
                    size=3.5)

## add variables from SkewTData:
         # first set pressure out-of-plot to be missing
SkewTData$P[SkewTData$P > pBot] <- NA
SkewTData$P[SkewTData$P < pTop] <- NA
         # convert everything else to plot coordinates
for (name in names(SkewTData)) {
  if (name == 'P') {next}
  SkewTData[, name] <- XYplot (SkewTData[, name], SkewTData$P)$X
}
SkewTData$P <- log10(SkewTData$P)
         # define a function to use to limit variables to plot area
LimitToPlotArea <- function (name) {          # assumes x range is (0,1)
  Valid <- (SkewTData[ , name] <= 1)
  Valid[SkewTData[ , name] < 0] <- FALSE
  SkewTData[ , name][!Valid] <<- NA
}

         # mixing-ratio lines: accumulate all into xmr,ymr
xmr <- vector ()
ymr <- vector ()
for (name in names(SkewTData)) {
  if (grepl ("MR", name)) {
    #v <- sub ("MR", "", name)
    #val <- as.numeric (v)
         # set points outside plot range to missing
    Valid <- (SkewTData[,name] <= 1)
    Valid[(SkewTData[ , name] < 0)] <- FALSE
    SkewTData[, name][!Valid] <- NA
    xmr <- c(xmr, SkewTData[ , name])
    ymr <- c(ymr, SkewTData$P)
  }
}
MRDF <- data.frame (X=xmr, Y=ymr)
g <- g + geom_path (data=MRDF, aes(x=X, y=Y), color='darkgreen', lty=4, lwd=0.8)

         # potential temperature lines
xmth <- vector ()
ymth <- vector ()
for (name in names(SkewTData)) {
  if (grepl ("Theta", name)) {
    if (grepl ("R", name) || grepl ("B", name) || grepl ("P", name) ||
          grepl("I", name)) {next}
    Valid <- (SkewTData[ , name] <= 1)
    Valid[(SkewTData[ , name] < 0)] <- FALSE
    SkewTData[ , name][!Valid] <- NA
    xmth <- c(xmth, SkewTData[ , name])
    ymth <- c(ymth, SkewTData$P)
  }
}
MRDFTH <- data.frame (X=xmth, Y=ymth)
g <- g + geom_path (data=MRDFTH, aes(x=X, y=Y), color='darkorange', lty=1, lwd=0.4)

         # equivalent potential temperature: have several to choose from:
         #   I  results from integration
         #   R  results from Rossby definition
         #   B  results from Bolton definition
         #   P  results from Davies-Jones definition
ThetaESelection <- "I"
ThetaEVariable <- paste ("Theta", ThetaESelection, sep='')

## equivalent potential temperature
xmI <- vector ()
ymI <- vector ()
for (name in names(SkewTData)) {
  if (grepl (ThetaEVariable, name)) {
    #v <- sub (ThetaEVariable, "", name)
    #val <- as.numeric (v)
    Valid <- (SkewTData[ , name] <= 1)
    Valid[SkewTData[ , name] < 0] <- FALSE
    SkewTData[ , name][!Valid] <- NA
    xmI <- c(xmI, SkewTData[ , name])
    ymI <- c(ymI, SkewTData$P)
  }
}
MRDFI <- data.frame (X=xmI, Y=ymI)
g <- g + geom_path (data=MRDFI, aes(x=X, y=Y), color='red', lty=1, lwd=0.4)

## save the diagram for other uses
skewTDiagram <- g
save(skewTDiagram, tBot, tTop, pBot, pTop, file="./skewTDiagramC130.Rdata")
```

3.3 Adding flight data to the background

Once the diagram is constructed as above, the following code chunk can be used to superimpose a measured sounding. Furthermore, the diagram can be preserved before this addition so that subsequent uses can just retrieve the ggplot definition of the background without the need to construct it again.

This is the basis for the function in Ranadu called SkewTDiagram( ), which returns the plot definition with a supplied sounding added to the background as preserved here.

```
## plot a sample sounding
# load("./skewTDiagram.Rdata")
# g <- skewTDiagram
Directory <- DataDirectory ()
Flight <- "rf16"
Project = "DEEPWAVE"
fname = sprintf("%s%s/%s%s.nc", Directory,Project,Project,Flight)
Data <- getNetCDF (fname, standardVariables(c("THETAP")))
r <- setRange (Data$Time, 123100, 125500)
DS <- Data[r, c("PSXC", "ATX", "DPXC")]
DP <- data.frame (P=XYplot (DS$ATX, DS$PSXC)$Y,
                  ATX=XYplot (DS$ATX, DS$PSXC)$X,
                  DPXC=XYplot(DS$DPXC, DS$PSXC)$X)
g <- g + geom_path (data=DP, aes(x=ATX, y=P, color='ATX'), lwd=1.5)
g <- g + geom_path (data=DP, aes(x=DPXC, y=P, color='DPXC'), lwd=1.5)
g <- g + theme(legend.position=c(0.2,0.85), legend.background=element_rect(fill="white"))
g <- g + labs(color='Measurement')
g <- g + scale_colour_manual(name = "Measurement",values = c('black', 'darkgreen'))

print(g)
```


Measurement
ATX
DPXC
Pressure [hPa]
300
400
500
600
700
800
900
1000
Temperature or dew point [°C]
-10
0
10
20
30
−70
−60
−50
−40
−30
−20
−10
0
10
20
30

– End of Memo –

Reproducibility:

PROJECT:	SkewT
ARCHIVE PACKAGE:	SkewT.zip
CONTAINS:	attachment list below
PROGRAM:	SkewT.Rnw
ORIGINAL DATA:	/scr/raf_data/DEEPWAVE/rf16.nc
GIT:	git@github.com:WilliamCooper/SkewT.git

Attachments:	SkewT.Rnw
	SkewT.pdf
	skewTDiagram.Rdata
	SessionInfo