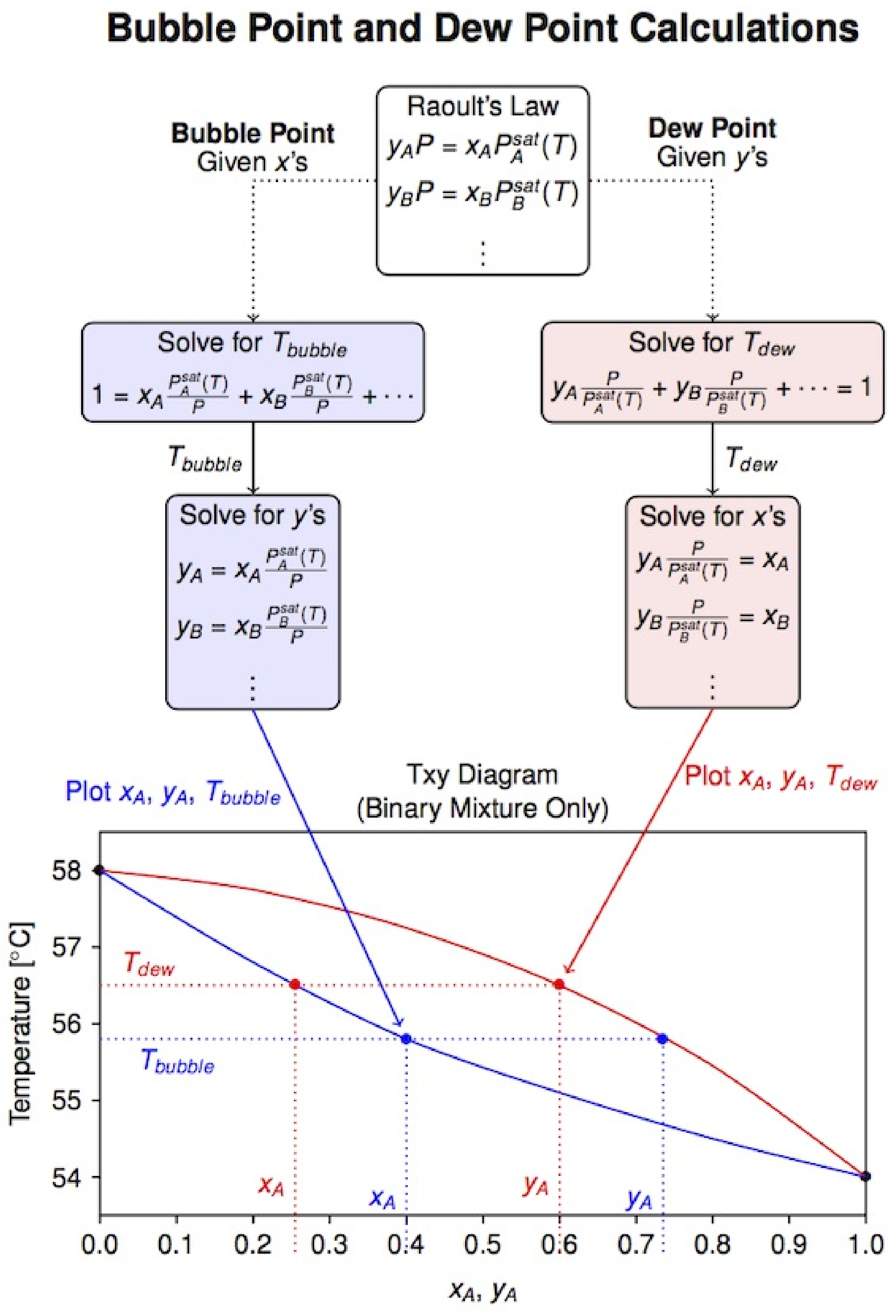
7.7 Bubble and Dew Points for Binary Mixtures

Summary

This notebook illustrates the use of Raoult's Law and Antoine's equation to compute the bubble and dew points for an ideal solution. The video is used with permission from LearnChemE, a project at the University of Colorado funded by the National Science Foundation and the Shell Corporation.

Overview of the Calculations


Bubble Point and Dew Point Calculations
Raoult's Law
$y_A P = x_A P_A^{sat}(T)$
$y_B P = x_B P_B^{sat}(T)$
Bubble Point Given x's
Dew Point Given y's
Solve for T_{bubble}
$1 = x_A \frac{P_A^{sat}(T)}{P} + x_B \frac{P_B^{sat}(T)}{P} + \cdots$
Solve for T_{dew}
$y_A \frac{P}{P_A^{sat}(T)} + y_B \frac{P}{P_B^{sat}(T)} + \cdots = 1$
T_{bubble}
T_{dew}
Solve for y's
$y_A = x_A \frac{P_A^{sat}(T)}{P}$
$y_B = x_B \frac{P_B^{sat}(T)}{P}$
Solve for x's
$y_A \frac{P}{P_A^{sat}(T)} = x_A$
$y_B \frac{P}{P_B^{sat}(T)} = x_B$
Plot x_A, y_A, T_{bubble}
Txy Diagram (Binary Mixture Only)
Plot x_A, y_A, T_{dew}
Temperature [°C]
T_{dew}
T_{bubble}
x_A
y_A
x_A, y_A

Bubble Point Calculations for Binary Mixtures

For an ideal binary mixture of components A and B, applying Raoult's law at the bubble point temperature gives the constraint

$$P = x_A P_A^{sat}(T) + x_B P_B^{sat}(T)$$

where $x_A P_A^{sat}(P)$ and $x_B P_B^{sat}(P)$ are the partial pressures of A and B, respectively. This relationship is the basis for an interative procedure for computing the bubble point temperature.

Step 1: Guess the temperature.

Step 2: Compute the 'K-factors'

$$K_A = \frac{P_A^{sat}(T)}{P} \qquad \text{and} \qquad K_B = \frac{P_B^{sat}(T)}{P}$$

Step 3: Compute the vapor phase mole fractions

$$y_A = K_A x_A \qquad \text{and} \qquad y_B = K_B x_B$$

Step 4: Check if $y_A + y_B = 1$. Adjust the temperature and repeat until the vapor phase mole fractions sum to one.

Solution by Manual Iteration

We're given a binary mixture composed of acetone and ethanol at atmospheric pressure where the liquid phase mole fraction of acetone is 0.40. The problem is to find the equilibrium temperature and the composition of the vapor phase.

Initialize the Python workspace with with default settings for plots.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [5]:

```
from scipy.optimize import brentq

class Species(object):

    def __init__(self, name='no name', Psat=lambda T: null):
        self.name = name
        self.Psat = Psat

    # compute saturation pressure given temperature.
    def Psat(self, T):
        raise Exception('Psat() has not been defined for ' + self.name)

    # compute saturation temperature given pressure
    def Tsat(self, P):
        return brentq(lambda T: self.Psat(T) - P, -50, 200)


acetone  = Species('acetone',  lambda T: 10**(7.02447 - 1161.0/(T + 224)))
benzene  = Species('benzene',  lambda T: 10**(6.89272 - 1203.531/(T + 219.888)))
ethanol  = Species('ethanol',  lambda T: 10**(8.04494 - 1554.3/(T + 222.65)))
hexane   = Species('hexane',   lambda T: 10**(6.88555 - 1175.817/(T + 224.867)))
toluene  = Species('toluene',  lambda T: 10**(6.95808 - 1346.773/(T + 219.693)))
p_xylene = Species('p_xylene', lambda T: 10**(6.98820 - 1451.792/(T + 215.111)))
```

In [6]:

```
A = acetone
B = ethanol

P = 760
xA = 0.4
xB = 1 - xA
```

We will use Antoine's equation to compute the saturation pressures for the pure components. These function are stored as entries in a simple Python dictionary.

The next cell performs the calculations outlined above. Execute this cell with different values of `T` until the vapor phase mole fractions sum to one.

In [7]:

```
T = 65

KA = A.Psat(T)/P
KB = B.Psat(T)/P

yA = KA*xA
yB = KA*xB

print(yA + yB)
```

```
1.337689658645641
```

Solution with a Root-Finding Function

To compute the bubble point for a binary mixture we need to solve the equation

$$P = x_A P_A^{sat}(T_{bubble}) + x_B P_B^{sat}(T_{bubble})$$

where P and x_A (and therefore $x_B = 1 - x_A$) are known. The bubble point composition is given by

$$y_A = \frac{x_A P_A^{sat}(T)}{P} \quad \text{and} \quad y_B = \frac{x_B P_B^{sat}(T)}{P}$$

Matlab and Python functions for solving equations rely on *root-finding* methods, that is, methods that find the *zeros* of a function. In this case we need to write our problem as

$$x_A \frac{P_A^{sat}(T)}{P} + x_B \frac{P_B^{sat}(T)}{P} = 1$$

Here we use the `brentq` function from the scipy.optimize library to return the root of this equation.

In [7]:

```
from scipy.optimize import brentq

brentq(lambda T: xA*A.Psat(T)/P + xB*B.Psat(T)/P - 1.0, 0, 100)
```

Out[7]:

```
68.5195836108029
```

In [9]:

```
def Tbub(X) :
    xA, xB = X
    return brentq(lambda T: xA*A.Psat(T)/P + xB*B.Psat(T)/P - 1.0, 0, 100)

print("Bubble point temperature = {:6.3f} [deg C]".format(Tbub((xA,xB))))

yA = xA*A.Psat(Tbub((xA,xB)))/P
yB = xB*B.Psat(Tbub((xA,xB)))/P

print("Bubble point composition = {:.3f}, {:.3f}".format(yA,yB))
```

```
Bubble point temperature = 68.520 [deg C]
Bubble point composition = 0.598, 0.402
```

Bubble Point Curve for a Txy Diagram

It's a relatively simply matter to encapsulate the bubble point calculation into a function that, given the liquid phase mole fraction for an ideal binary mixture, uses a root-finding procedure to return the bubble point temperature.

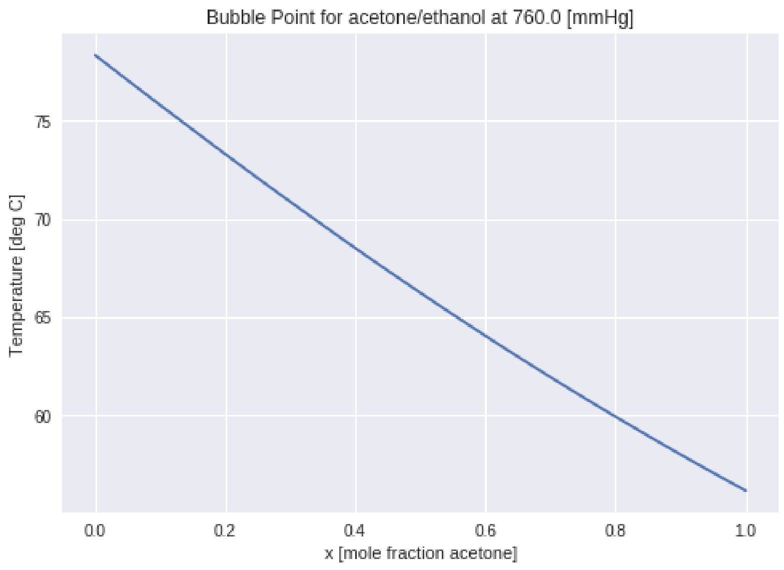
In [13]:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0,1)
plt.plot(x,[Tbub((xA,xB)) for (xA,xB) in zip(x,1-x)])
plt.xlabel('x [mole fraction {:s}]'.format(A.name))
plt.ylabel('Temperature [deg C]')
plt.title('Bubble Point for {:s}/{:s} at {:5.1f} [mmHg]'.format(A.name,B.name,P))
```

Out[13]:

```
Text(0.5,1,'Bubble Point for acetone/ethanol at 760.0 [mmHg]')
```



Dew Point Calculations for a Binary Mixture

To compute the dew point for a binary mixture we need to solve the equation

$$y_A \frac{P}{P_A^{sat}(T)} + y_B \frac{P}{P_B^{sat}(T)} = 1$$

where P and y_A (and therefore $y_B = 1 - y_A$) are known. The dew point composition is given by

$$x_A = y_A \frac{P}{P_A^{sat}(T)} \qquad \text{and} \qquad x_B = y_B \frac{P}{P_B^{sat}(T)}$$

Matlab and Python functions for solving equations rely on *root-finding* methods, that is, methods that find the *zeros* of a function. Here we use the `fsolve` function from the scipy.optimize library to return the root of the dew point equation. Note that `fsolve` returns a list of roots, so the terminal `[0]` on the expression selects the first root (and presumably only) of the bubble point equation.

In [16]:

```python
def Tdew(Y):
    yA,yB = Y
    return brentq(lambda T:yA*P/A.Psat(T) + yB*P/B.Psat(T) - 1.0, 0, 100)

print("Dew point temperature = {:6.3f} [deg C]".format(Tdew((yA,yB))))

xA = yA*P/A.Psat(Tdew((yA,yB)))
xB = yB*P/B.Psat(Tdew((yA,yB)))

print("Dew point composition = {:.3f}, {:.3f}".format(xA,xB));
```

```
Dew point temperature = 68.520 [deg C]
Dew point composition = 0.400, 0.600
```

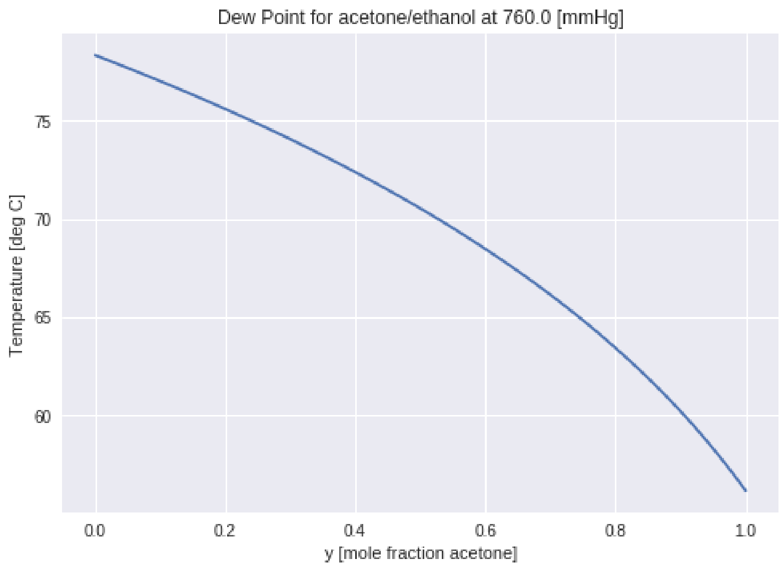
Dew Point Curve on the Txy diagram

As shown above for bubble point calculations, the dew point curve on the Txy diagram can be plotted by mapping the Tdew function onto a grid of mole fractions.

In [17]:

```
import numpy as np
import matplotlib.pyplot as plt

y = np.linspace(0,1)
plt.plot(y,[Tdew((yA,yB)) for (yA,yB) in zip(y,1-y)])
plt.xlabel('y [mole fraction {:s}]'.format(A.name))
plt.ylabel('Temperature [deg C]')
plt.title('Dew Point for {:s}/{:s} at {:5.1f} [mmHg]'.format(A.name,B.name,P));
```



In [ ]: