# Efficiency and Economic Analysis of Steady-State Ethanol Distillation

Lucas Culverhouse
Mar 17, 2023
ECH 145A Section A06

**Abstract**

Fractional distillation is an incredibly common technique in chemical engineering from the purification of solutions. One industry where it is frequently employed is the alcoholic beverage industry. This report has investigated fractional distillation and analysis of ethanol water mixtures using the McCabe-Thiele assumption. We obtain a method and analysis for determining the efficiency of the column, and the economic impacts therein. This analysis suggests the MaCabe-Theile assumption is accurate under steady-state conditions, and that the other assumptions described herein are useful for fractional distillation analysis.

**Introduction**

The following experiments were conducted to evaluate the behavior of a steady-state distillation of ethanol and water. Distillation columns are very important in chemical engineering and industry. The purpose shown here, distillation of ethanol, is widely used in the consumer alcohol industry. The effectiveness of these distillations has implications on business efficiency and legal restrictions.

The processes explored in this report demonstrate use of McCabe-Theile analysis, a powerful set of assumptions about distillation columns that makes analysis of steady state columns quite simple.

To perform this analysis, an understanding of temperature effects on density is required. This is explored in the sections regarding gauging alcohol and TTB tables. Density is explored as a method of measuring the composition of relevant distillation streams to conduct further analysis.

Also, economic analysis of the distillation performed is conducted. This accounts for industrial considerations, relevant to industrial and consumer alcohol production.

**Theory**

*Measuring Density: Pycnometer.* Density is described by a simple relation:

$$\rho = \frac{m}{V} \quad (1)$$

So, if we have a known solution volume and mass we can easily calculate the density. This is the principle behind a pycnometer, which is simply a precisely constructed glass instrument with a known volume. The pycnometer can be measured for its mass to get the mass of the solution once it has been added. Density is also a strong function of temperature, so pycnometers also have a thermometer attached to account for these effects when collecting data.

*Measuring Density: Densitometer.* A densitometer can measure density using much smaller samples by leveraging the vibration of a small u-shaped glass tube. The sample is put into this glass tube which is then made to vibrate at its characteristic frequency electronically [1]. This frequency is changed by the mass of the sample inside, and since the tube has a known volume, its density can then be calculated using Eq. 1.

*TTB Tables.* The US Alcohol and Tobacco Tax and Trade Bureau (TTB) publishes several tables that are useful in gauging alcohol [2, 3]. These tables allow gauging of alcohol using a hydrometer. TTB table #1 accounts for the effects of temperature on the gauging of alcohol. Temperature not only affects the density of the fluid but the density of the hydrometer as well; both are accounted for by table 1. Experimentally, hydronometers are not suitable for our purpose as the volumes of solution they require to be accurate are too large. Instead we must convert the densities measured by our glass devices to the density a pycnometer would have measured at the same temperature. This is done using the following equation [3]:

$$\rho_2 = \rho_1(1 + \alpha(T_1 - 60^\circ F)) \quad (2)$$

Eq. 2 describes what the density measured by a hydrometer, $\rho_2$would be given the measured density

$\rho_1$ and an expansion coefficient, α, at the given temperature. TTB table #1 provides conversions from proof and temperature to true proof, so we first need to convert our density to proof units using TTB table #6. This table provides a conversion from specific gravity, which is obtained easily from density, to proof. This proof can then be used with TTB table #1 to get the true proof of the solution being measured.

*True Proof.* Proof is a function of temperature, so alcohol gauging must be done in the temperature corrected true proof to ensure accurate results. The true proof, $C_3$, can be expressed as the following truncated taylor series expansion [3]:

$$C_3 = f = \left[ f + (C_2 - C_R)\frac{\partial f}{\partial C} + (T_1 - T_R)\frac{\partial f}{\partial T} \right]_{C_R, T_R} \quad (3)$$

The partial derivatives shown in Eq. 3 are estimated using the TTB tables previously discussed. From the TTB tables and the equation for true proof, a density reading from our pycnometer or densitometer can be converted to true proof.

*Mole Fraction.* True proofs can be used to find the vol% water and ethanol of the sample from TTB table #6. Using these vol% and molar masses, we can find the mol% ethanol.

*Antoine's Equation.* Antoine's equation is an empirical relation that relates the logarithm of the saturation vapor pressure to temperature, provided three tabulated constants. Antoine's equation is as follows [4]:

$$\log_{10} P^* = A - \frac{B}{T+C} \quad (4)$$

Where *A, B,* and *C* are tabulated constants.

*Van Laar's Model for Activity Coefficients.* Van Laar's model is a simple two constant model relating the natural logarithm of two activity coefficients to compositions and tabulated constants. The model is as follows [4]:

$$\ln \gamma_1 = A_{12}\left(\frac{A_{21}x_2}{A_{12}x_1 + A_{21}x_2}\right)^2$$

$$\ln \gamma_2 = A_{21}\left(\frac{A_{12}x_1}{A_{12}x_1 + A_{21}x_2}\right)^2 \quad (5)$$

From the van Laar parameters $A_{12}$ and $A_{21}$ the activity coefficients can be calculated.

*VLE Curves.* A vapor-liquid equilibrium (VLE) curve describes changes in the mole fractions of a substance in the vapor and liquid phases at equilibrium. The VLE curves presented in this report make several physical assumptions. These are: a) the vapor is ideal, b) pressures are small and temperature is very large, c) Antoine's equation and van Laar's model for activity coefficients are valid. Combining all of these we get the following two equations [4]:

$$\gamma_L^{etOH} x P^{etOH*} = yP \quad (6a)$$

$$\gamma_L^{H_2O} (1 - x) P^{H_2O*} = (1 - y)P \quad (6b)$$

So, given that we have the saturation vapor pressures from Antoine's equation, and the activity coefficients from van Laar's model, for a given T and P we can find x and y. Where x is the composition of ethanol in the liquid phase, and y is the composition of ethanol in the vapor phase.

*McCabe-Thiele Analysis.* McCabe-Theile analysis and diagrams are based on the so-called McCabe-Thiele assumption which is that the number of moles of vapor ascending a distillation column is constant from plate to plate [5]. Using this assumption and some relations that come from it, we can construct a McCabe-Thiele diagram to analyze the column. First we must plot an equilibrium curve, this report uses the VLE curve method described above, and the line $x = y$. From here we plot the point $(x_P, x_P)$ and draw the rectifying line through it. The rectifying line obeys the following equation [5]:

$$y_i = \frac{O}{O+1} x_{i-1} + \frac{x_P}{O+1} \quad (7a)$$

Where $O$ is the reflux ratio of the column. Then the point $(x_W, x_W)$ and the stripping line is plotted. The stripping line obeys the following equation [5]:

$$y_i = \frac{O+F}{O+1} x_{i-1} + \frac{1-F}{O+1} x_W \quad (7b)$$

Where $F$ is the molar flux of feed per mole product. Once these have been plotted their intersection is the feed composition $x_F$. Then successive horizontal and vertical line segments can be drawn between the VLE curve and these two operating lines. The amount of these steps it takes to reach the waste composition from the product composition is the number of theoretical plates needed for the fractionation. This assumes that each q-line has a value of 1.

**Experimental Methods**

The experiment was carried out in two main phases: the pycnometry and calibration of measurement devices, and performing steady-state distillations at various reflux ratios.

*Pycnometer and Calibration.* First the dry weight of a glass pycnometer was taken. Then, the pycnometer was completely filled with deionized water. The total mass and measured temperature were then recorded for calibrating the volume of the pycnometer. Several samples of known mass fractions of ethanol were then created by first weighing the desired amount of ethanol and filling with water until ~50g of total solution was reached. These solutions spanned the range from pure ethanol to pure water.

Each of these solutions was divided into two portions: one portion of about 10mL to be used in the densitometer, and the remaining portion to be used in the pycnometer. The densitometer portion was measured multiple times with each measured portion being discarded after each trial. The pycnometer was first rinsed two times with about 5mL each of the remaining portion. Then the pycnometer was completely filled with the solution and weighed; the temperature and total mass were then recorded.

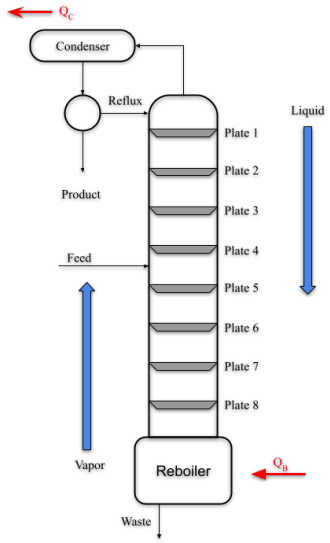
*Steady-State Distillation.* A large distillation column with several thermocouples reporting the temperature at each plate and several other key locations was used. The feed solution was approximately 15 vol% ethanol. First the boiler and chiller were turned on then the inlet valve was opened. The column was then left to operate for approximately 30 minutes until it reached steady state.

Steady state was determined in two ways. Firstly by inspection the temperature readouts reported for changes, and then by measuring the mass flow rates into and out of the column. Once it was determined that steady state had been reached multiple samples were taken of the product and waste streams.

First each was drained so that the measured mass flow rate was not affected by the build-up already present from non-steady-state operation. Then, each was drained and the total volume, mass, and density of each solution was determined. Using a syringe, several samples were taken from four different trays of the distillation column; two were taken from above the feed inlet, and two were taken from below the inlet. These were each measured for density and temperature at the time of both collection and density measurement.

The boiler and chiller powers were collected along with the pressure drop from the column. The column was then shut down and one more density sample was taken from the product and the waste streams.

Below is a diagram of the distillation column used in the experiments described above.




## Results

*Pycnometry and Theoretical Conversions from Density.* From the pycnometer measurements taken and the theory discussed above, theoretical conversions from ethanol water solution density were obtained. Density is converted to true proof and mole fraction; these are reported at several different temperatures, and values are computed assuming $\rho_{water} = 0.99904\ g/cc$ and $\rho_{etOH} = 0.79313\ g/cc$ at 60℉ [6].

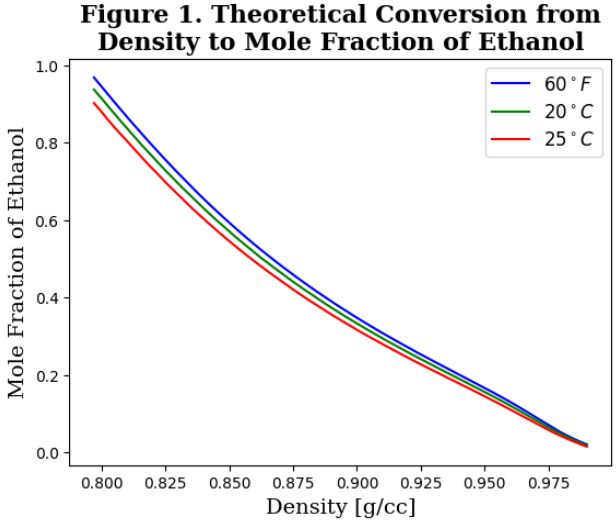



**Figure 1.** Density to mole fraction conversion presented at several different temperatures. Note that 60℉ ~ 15.5℃. Temperature variations produce a maximum ethanol mole fraction variation of about 0.06 /℃.

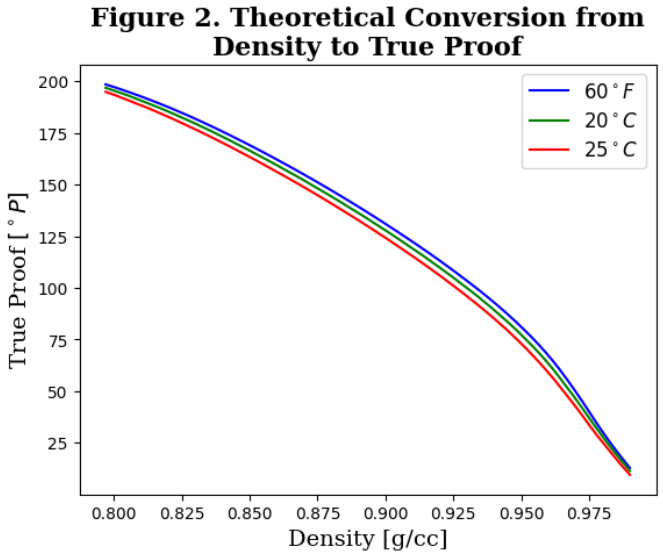



**Figure 2.** Density to true proof conversion presented at several different temperatures. Note that 60℉ ~ 15.5℃. Temperature variations produce a maximum proof variation of about 0.8 °P/℃.

The curves above show the effects of greater amounts of ethanol on the density of the ethanol water mixture. These conversions will be useful in determining the mole fraction of samples obtained from the distillation column, and their true proofs.

*Comparison of Pycnometer and Densitometer Measurements.* Both pycnometer and densitometer measurements were taken for each sample.

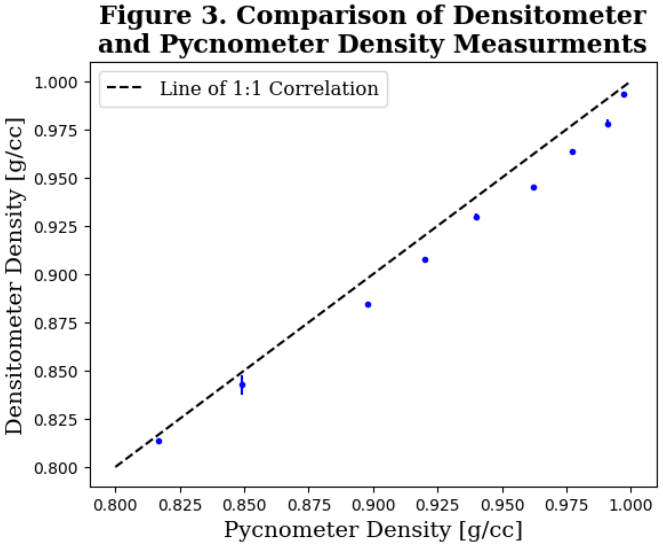



**Figure 3.** Temperature adjusted densitometer measurements compared to pycnometer density measurements at the same ethanol concentration.

Values below the line have a greater pycnometer measured density.

The densitometer consistently reports lower densities than the pycnometer. Densitometer measurements were temperature adjusted as discussed in the theory to account for variation in sampling temperatures.

*McCabe Thiele Diagrams.* As discussed in the experimental methods, two trials of the column distillation were performed at different reflux ratios.

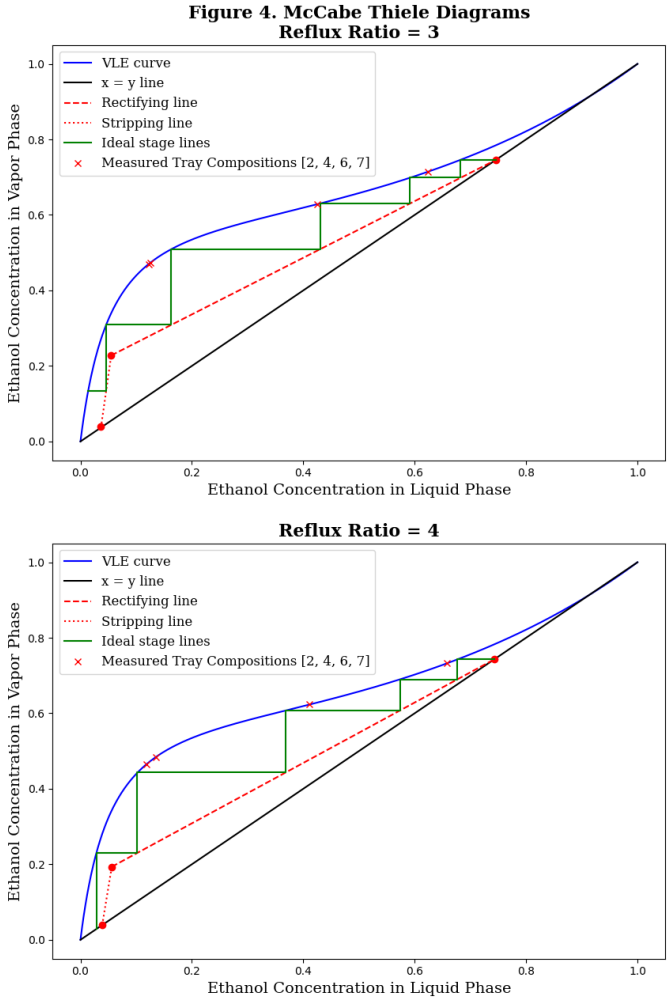



**Figure 4. McCabe Thiele Diagrams**

**Figure 4.** McCabe Thiele diagrams for each column reflux ratio. Ratio 3 VLE curve was calculated with pressure at 752.9 mmHg; ratio 4 VLE curve was calculated with pressure at 759.0 mmHg [7]. All q-values are assumed to be 1 for plotting. Tray concentrations are calculated using theoretical conversions from measured density to mole fraction. Reflux ratios 3 and 4 take ~5.1, and ~4.9 theoretical plates to reach the separation observed

Below are the different plate concentrations for each tray sampled along with the theoretical values from the McCabe Thiele diagram.

**Table 1. Tray Ethanol Concentrations for Reflux Ratio = 3**

| Tray | etOH Concentration | Collected Temp. [℃] | Theory Conc. |
|---|---|---|---|
| 2 | 0.623 ± 0.01 | 79.0 | 0.683 |
| 4 | 0.425 ± 0.06 | 80.2 | 0.430 |
| 6 | 0.125 ± 0.04 | 85.9 | 0.163 |
| 7 | 0.122 ± 0.05 | 86.0 | 0.046 |

**Table 2. Tray Ethanol Concentrations for Reflux Ratio = 4**

| Tray | etOH Concentration | Collected Temp. [℃] | Theory Conc. |
|---|---|---|---|
| 2 | 0.659 ± 0.03 | 78.7 | 0.677 |
| 4 | 0.412 ± 0.04 | 79.5 | 0.368 |
| 6 | 0.135 ± 0.13 | 80.4 | 0.102 |
| 7 | 0.118 ± 0.09 | 86.4 | 0.029 |

**Table 3. Mass Balance and Stream Composition for Reflux Ratio = 3**

| | Mass Flow Rate [g/s] | Composition [mol ethOH/mol] |
|---|---|---|
| Feed | 1.301 | 0.055 |
| Waste | 1.214 | 0.037 |
| Product | 0.261 | 0.746 |

**Table 4. Mass Balance and Stream Composition for Reflux Ratio = 4**

| | Mass Flow Rate [g/s] | Composition [mol ethOH/mol] |
|---|---|---|
| Feed | 1.112 | 0.056 |
| Waste | 1.116 | 0.039 |
| Product | 0.155 | 0.743 |

**Table 3. & Table 4.** Percent closure of the column can be seen to be 88.2% for table 3, and 87.5% for table 4

*Economic Analysis.* The column took on average 1473.5 W to run both the chiller and the boiler. Using the national average price of electricity [8] that means this column would cost $0.248 per hour to run under similar conditions. Using the average US price of ethanol [9] the column could produce valuable ethanol at about $8.86 per hour.

**Discussion**

*Theoretical Conversions from Density.* As seen in figures 2 and 3, the higher the fluid density, the lower the concentration of ethanol. This makes sense because ethanol is less dense, so one would expect the density of the mixture to decrease as more is added. From the different temperatures reported it is apparent that as the temperature of the solution increases, the ethanol concentration associated with that density decreases. This makes physical sense as the same solution at a higher temperature should have a lower density. We can also see the temperature effects are significant but not extreme with the temperature variation producing a maximum variation of about 0.8 °P/°C and about 6 mol% /°C. These values are a maximum and do not reflect the temperature effects across the entire curve. These theoretical curves provide a clear way to get values for stream composition based on the densities measured.

*Comparison on Pycnometer and Densitometer Measurements.* From figure 3 we see that the densitometer consistently predicted a higher density than the pycnometer. This could be due to several factors. The pycnometer and densitometer both need to be completely filled in order for their readings to be accurate. This likely constitutes a systematic error in one of the devices, as with only random error we would not expect one device to consistently over report compared to the other.

*McCabe-Thiele Analysis.* From the McCabe-Theile diagrams we can see the conversion process from our feed stream to the product and the waste streams. As expected from the physical layout of the column, the feed is at approximately 5 plates. Each VLE curve was constructed using the local pressure at the time of the data's recording, so their shapes are not quite identical. It seems from this diagram that our McCabe-Theile analysis is accurate as it provides us with an accurate prediction of the number of plates needed for full conversion.

*Total Mass Flows and Compositions.* The analysis of the total mass flows and their compositions shows that the column is actually distilling the alcohol and making it much more pure. From a mol fraction of about 0.05 up to a much larger mol fraction of about 0.7 turns the feed-stock into a much more useful product. Industrially, the product stream alcohol would be much more valuable than the feedstock. However, when looking at the total flow rates it is clear that this useful product is produced rather slowly. Additionally, although the higher reflux ratio column has a lower mass flow rate, the difference in ethanol purity was not that pronounced. It is likely that a higher reflux ratio could produce higher purity ethanol, but from examining the McCabe-Theile diagrams and the economic analysis, it seems like reflux ratios higher than 3 or 4 aren't practical for producing large quantities of ethanol efficiently, at least not using this configuration.

*Economic Analysis.* Given the sale price and desirability of high concentration ethanol the economic side of the distillation process makes

sense. Alcohol at our feed composition of 0.05 mol% ethanol is not nearly as valuable and lucrative as the 70 mol% ethanol from our product. Also, distillation columns are not incredibly expensive to operate. On top of this transporting higher proof ethanol is more cost effective due to the higher amounts of ethanol transported per unit mass transported.

**Conclusion**

These experiments show the usefulness of alcohol distillation and the process of gauging alcohols for economic purposes.

Methods for measuring the composition of water-ethanol mixtures were developed and can be seen to be quite accurate. So long as there are accurate measurements of the density and temperature, good data can be obtained on the composition of water-ethanol mixtures.

From the economic analysis we can see that modern distillation techniques can efficiently increase the proof of alcohol with relatively low energy costs. Proof can be drastically increased with even a smaller, non industrial distillation column, which is important for the production of distilled spirits.

From the experiments discussed we can also see that the McCabe-Thiele analysis for our experimental conditions is valuable. The analysis accurately describes the physical state of the column and allows us to calculate its efficiency. Construction of the McCabe-Thiele diagrams allows us to better analyze the distillation column system and get insights into its functioning.

**Notations**

ρ: The density of a substance
α: The expansion coefficient of a material
$C$: The proof of a water-ethanol mixture, in $^{\circ}P$
$C_R$: The floor of the proof
$T_R$: The floor of the temperature
$A,\ B,\ C$: Antoine coefficients
$P^*$: The saturation vapor pressure
γ: The activity coefficient
$A_{12}, A_{21}$: The van Laar parameters of a substance
$O$: The reflux ratio of a distillation column
$F$: The molar flux of feed per mole product
$x$: The ethanol concentration in the liquid phase
$y$: The ethanol concentration in the vapor phase
VLE Curve: Vapor-liquid equilibrium curve
TTB: US Alcohol and Tobacco Tax and Trade Bureau
$Q_B$: The heat added at reboiler per mole product
$Q_C$: The heat removed at the condenser per mole product
$x_F$: The mole fraction of ethanol in feed
$x_P$: The mole fraction of ethanol in product
$x_W$: The mole fraction of ethanol in waste

**References**


[1] “Certificate of Conformity 15123R-10 Specific Gravity Pycnometer,” *Kimble Chase,* 17-Oct-2013

[2] “TTB: Distilled spirits: Proofing page interpolation table,” *TTBGov - Proofing Page Interpolation table*, 06-Oct-2022. [Online]. Available: https://www.ttb.gov/distilled-spirits/proofing-page-interpolation-table. [Accessed: 17-Mar-2023].

[3] Wan, J. “*Lab 3.1: Alcohol Gauging”. University of California, Davis, ECH145A.* Gauging Alcohol.pdf

[4] Wan, J. “*Rayleigh Distillation”. University of California, Davis, ECH145A.* Rayleigh.pdf

[5] Wan, J. “*Distillation Columns”. University of California, Davis, ECH145A.* Distillation-columns.pdf

[6] J. R. Rumble, T. J. Bruno, and M. J. Doa, “p. F4,” in *CRC Handbook of Chemistry and Physics: A ready-reference book of Chemical and physical data*, Boca Raton: CRC Press/Taylor & Francis Group, 2021.

[7] N. O. A. A. US Department of Commerce, “Time Series Viewer: University Airport, CA,” *Time Series Viewer*, 13-Mar-2023. [Online]. Available: https://www.weather.gov/wrh/timeseries?site=KEDU. [Accessed: 17-Mar-2023].

[8] “Average energy prices for the United States, regions, census divisions, and selected metropolitan areas,” *U.S. Bureau of Labor Statistics*. [Online]. Available: https://www.bls.gov/regions/midwest/data/averageenergyprices_selectedareas_table.htm. [Accessed: 17-Mar-2023].

[9] “Producer price index by industry: Ethyl Alcohol Manufacturing,” *FRED*, 15-Mar-2023. [Online]. Available: https://fred.stlouisfed.org/series/PCU325193325193. [Accessed: 17-Mar-2023].


**Supplementary Materials**

*Sample Calculations.* The following are a selection of sample calculations to demonstrate the methods used to determine the quantitative values and errors reported. Error calculations will be shown as an implication of the formula used to solve for the desired physical quantity. Many of the calculations: averages, standard deviations, and curve fitting, are shown in the python code attached below this section.

*Error Propagation Example: Density From Pycnometer Measurements*

Density can be calculated as shown in Eq. 1. This requires an error propagation of the division used. The final equation is:

$$\Delta\rho = \rho\sqrt{\left(\frac{\Delta m}{m}\right)^2 + \left(\frac{\Delta V}{V}\right)^2}$$

With example numbers plugged in this is:

$$\Delta\rho = (0.9732\ g/cc)\sqrt{\left(\frac{0.01}{64.81}\right)^2 + \left(\frac{0.7}{66.59}\right)^2}$$

$$\Delta\rho = 0.01$$

In [10]:
```
from dataclasses import dataclass
import numpy as np
from numpy import ndarray, float64
import matplotlib.pyplot as plt
import scipy
```

In [11]:
```
TABLE1 = np.loadtxt("Data/TTB_Table_1_digitized-2.csv", delimiter=",")
TABLE6 = np.loadtxt(
    "./Data/TTB_Table_6_digitized.csv",
    delimiter=",",
    skiprows=10,
    usecols=[0, 1, 2, 3, 4],
)
```

In [12]:
```
@dataclass
class EthanolMeasurement:
    concentration: float64  # %ethanol by mass
    density: ndarray  # g/cm^3
    dtemp: ndarray  # Celsius
    pmass: ndarray  # grams
    ptemp: ndarray  # Celsius


# Data for measurments of various concentrations of ethanol
ethanol00 = EthanolMeasurement(
    concentration=float64(0.0),
    density=np.array([0.9925, 0.9925, 0.9925]),
    dtemp=np.array([23.4, 23.4, 23.4]),
    pmass=np.array([60.505, 60.514, 60.507, 60.508]),
    ptemp=np.array([24.0, 24.0, 24.0]),
)

ethanol10 = EthanolMeasurement(
    concentration=float64(5.003 / 50.2),
    density=np.array([0.9757, 0.9799, 0.9801]),
    dtemp=np.array([25.7, 25.5, 25.5]),
    pmass=np.array([60.346, 60.347, 60.345]),
    ptemp=np.array([25.2, 25.2, 25.4]),
)

ethanol20 = EthanolMeasurement(
    concentration=float64(10.0 / 50.055),
    density=np.array([0.9627, 0.9652, 0.9657]),
    dtemp=np.array([26.9, 26.8, 26.9]),
    pmass=np.array([59.99, 59.987, 59.985]),
    ptemp=np.array([25.8, 25.6, 25.8]),
)

ethanol30 = EthanolMeasurement(
    concentration=float64(15.0 / 50.285),
    density=np.array([0.9462, 0.9480, 0.9481]),
    dtemp=np.array([27.3, 27.4, 27.4]),
    pmass=np.array([59.609, 59.607, 59.604]),
    ptemp=np.array([25.4, 25.4, 25.2]),
```

```
)

ethanol40 = EthanolMeasurement(
    concentration=float64(20.056 / 50.155),
    density=np.array([0.9269, 0.9303, 0.9315]),
    dtemp=np.array([24.8, 24.9, 24.9]),
    pmass=np.array([59.036, 59.031, 59.029]),
    ptemp=np.array([25.0, 25.0, 25.0]),
)

ethanol50 = EthanolMeasurement(
    concentration=float64(25.026 / 50.006),
    density=np.array([0.9085, 0.9078, 0.9081]),
    dtemp=np.array([25.4, 26.0, 26.1]),
    pmass=np.array([58.520, 58.520, 58.519]),
    ptemp=np.array([25.4, 25.6, 25.4]),
)

ethanol60 = EthanolMeasurement(
    concentration=float64(30.028 / 50.175),
    density=np.array([0.8874, 0.8867, 0.8865]),
    dtemp=np.array([25.2, 25.4, 25.5]),
    pmass=np.array([57.958, 57.948, 57.944]),
    ptemp=np.array([24.0, 24.2, 24.3]),
)

ethanol80 = EthanolMeasurement(
    concentration=float64(40.148 / 49.953),
    density=np.array([0.8557, 0.8457, 0.8445]),
    dtemp=np.array([24.0, 24.3, 24.3]),
    pmass=np.array([58.612, 58.613, 58.606]),
    ptemp=np.array([21.5, 22.0, 22.2]),
)

ethanol90 = EthanolMeasurement(
    concentration=float64(45.076 / 50.101),
    density=np.array([0.8156, 0.8153, 0.8152]),
    dtemp=np.array([23.9, 23.9, 24.0]),
    pmass=np.array([57.737, 57.737, 57.739]),
    ptemp=np.array([23.4, 23.4, 23.4]),
)

ethanol_measurments = [
    ethanol00,
    ethanol10,
    ethanol20,
    ethanol30,
    ethanol40,
    ethanol50,
    ethanol60,
    ethanol80,
    ethanol90,
]


@dataclass
```

```
class Pycnometer:
    empty_mass: float64  # in grams
    volume: float64  # in cm^3
    std_volume: float64


def new_pycnometer(
    empty_mass: float64, full_mass: ndarray, water_density: float64
) -> Pycnometer:
    water_mass = full_mass - empty_mass
    water_volume = water_mass / water_density
    volume = water_volume.mean()
    std_volume = water_volume.std()
    return Pycnometer(empty_mass, volume, std_volume)


pycnometer1 = new_pycnometer(
    34.817, np.array([60.505, 60.514, 60.507, 60.508]), 0.997296
)
pycnometer2 = new_pycnometer(35.844, np.array([62.600, 62.586]), 0.997747)


@dataclass
class PycnometerMeasurment:
    full_mass: float64  # in grams
    avg_temp: float64  # in C
    density: float64  # in g/cc
    std_density: float64


def new_pycnometer_measurement(pycnometer: Pycnometer, mass: ndarray, temp: ndarray
    fluid_mass = mass - pycnometer.empty_mass
    avg_mass = fluid_mass.mean()
    std_mass = fluid_mass.std()
    avg_temp = temp.mean()

    avg_density = avg_mass / pycnometer.volume
    std_density = avg_density * np.sqrt(
        (std_mass / avg_mass) ** 2 + (pycnometer.std_volume / pycnometer.volume) **
    )
    return PycnometerMeasurment(avg_mass, avg_temp, avg_density, std_density)


which_pycno = [
    pycnometer1,
    pycnometer1,
    pycnometer1,
    pycnometer1,
    pycnometer1,
    pycnometer1,
    pycnometer1,
    pycnometer2,
    pycnometer2,
]
pycnometer_measurments: list[PycnometerMeasurment] = []
for i, eth in enumerate(ethanol_measurments):
```

```
    measurment = new_pycnometer_measurement(which_pycno[i], eth.pmass, eth.ptemp)
    pycnometer_measurments.append(measurment)
```

```
25.733333333333334
25.733333333333334
```

In [14]:

```
def find_nearest(data, value):
    idx = (np.abs(data - value)).argmin()
    return idx


def find_nearest_value(data, value):
    idx = (np.abs(data - value)).argmin()
    return data[idx]


def true_proof(density, temperature):
    """
    Accepts a density in g/cm^3 and a temperature in C and returns the true proof
    """
    alpha = 25e-6
    t1 = (temperature * 1.8) + 32
    p2 = density * (1 + alpha * (t1 - 60))
    spg2 = p2 / 0.99904

    interp_proof = scipy.interpolate.interp1d(TABLE6[:, 4], TABLE6[:, 0])
    c2 = interp_proof(spg2)

    cr = np.floor(c2)
    tr = np.floor(t1)

    proof_range = np.arange(0, 207)
    temp_range = np.arange(1, 101)
    temp_idx = np.where(temp_range == tr)[0][0] + 1
    proof_idx = np.where(proof_range == cr)[0][0]

    f = TABLE1[proof_idx, temp_idx]
    f_proof_plus_one = TABLE1[proof_idx + 1, temp_idx]
    f_temp_plus_one = TABLE1[proof_idx, temp_idx + 1]

    df_dC = (f_proof_plus_one - f) / (cr + 1 - cr)
    df_dT = (f_temp_plus_one - f) / (tr + 1 - tr)

    true_proof = f + (c2 - cr) * df_dC + (t1 - tr) * df_dT
    return true_proof


def mass_fraction(true_proof):
    p_water_60 = 0.99904
    p_ethanol_60 = 0.79313
    interp_percent_water = scipy.interpolate.interp1d(TABLE6[:, 0], TABLE6[:, 2])
    vol_percent_water = interp_percent_water(true_proof)
    vol_percent_ethanol = true_proof / 2
    mass_frac_ethanol = (vol_percent_ethanol * p_ethanol_60) / (
        vol_percent_ethanol * p_ethanol_60 + vol_percent_water * p_water_60
    )
```

```
    return mass_frac_ethanol


def mole_fraction(true_proof):
    p_water_60 = 0.99904
    p_ethanol_60 = 0.79313
    molar_mass_ethanol = 46.06844
    molar_mass_water = 18.01528
    interp_percent_water = scipy.interpolate.interp1d(TABLE6[:, 0], TABLE6[:, 2])
    vol_percent_water = interp_percent_water(true_proof)
    vol_percent_ethanol = true_proof / 2
    mole_frac_ethanol = (vol_percent_ethanol * p_ethanol_60 / molar_mass_ethanol) /
        (vol_percent_ethanol * p_ethanol_60 / molar_mass_ethanol)
        + (vol_percent_water * p_water_60 / molar_mass_water)
    )
    return mole_frac_ethanol


def temp_correct(density, dTemp, pTemp, true_proof):
    """
    Accepts a densitometer denstiy reading in g/cc a
    densometer and pycnometer temperature reading in C
    and a true_proof to return the temperature adjusted
    density measurement of the densitometer in g/cc
    """

    def FtoC(tempF):
        return (tempF - 32) * (5 / 9)

    alpha = 25e-6
    t1 = (dTemp * 1.8) + 32
    p2 = density * (1 + alpha * (t1 - 60))
    p_water = 0.99904
    spg2 = p2 / 0.99904

    interp_proof = scipy.interpolate.interp1d(TABLE6[:, 4], TABLE6[:, 0])
    c2 = interp_proof(spg2)

    cr = np.floor(c2)
    tr = np.floor(t1)

    proof_range = np.arange(0, 207)
    temp_range = np.arange(1, 101)
    temp_idx = np.where(temp_range == tr)[0][0] + 1
    proof_idx = np.where(proof_range == cr)[0][0]
    spg_idx = find_nearest(TABLE6[:, 1], true_proof / 2)

    tp_at_temp = TABLE1[proof_idx, temp_idx]
    tp_at_temp_plus_one = TABLE1[proof_idx, temp_idx + 1]
    spg = TABLE6[spg_idx, 4]
    spg_plus_one = TABLE6[spg_idx + 1, 4]
    ap_at_x = TABLE6[spg_idx, 0]
    ap_at_x_plus_one = TABLE6[spg_idx + 1, 0]

    dAP_dTP = abs((cr + 1 - cr) / (tp_at_temp_plus_one - tp_at_temp))
    dTP_dT = abs((tp_at_temp_plus_one - tp_at_temp) / (FtoC(tr + 1) - FtoC(tr)))
```

```
    dp_dAP = abs((p_water * (spg_plus_one - spg)) / (ap_at_x_plus_one - ap_at_x))

    dp_dT = abs(
        ((dTP_dT * dAP_dTP * dp_dAP) / (1 + alpha * (dTemp - 15.5556)))
        - ((alpha * density) / (1 + alpha * (dTemp - 15.5556)))
    )

    return density + dp_dT * (pTemp - dTemp)
```

In [40]:

```
# Figure 1. mole fraction vs density for various temperatures
font = dict(family="serif", size=14)
legend = dict(family="serif", size=12)
title = dict(family="serif", size=16, weight="bold")

densities = np.linspace(0.7970, 0.9900, 1000)

true_proof_60F = []
true_proof_20C = []
true_proof_25C = []
for density in densities:
    true_proof_60F.append(true_proof(density, 15.5556))
    true_proof_20C.append(true_proof(density, 20))
    true_proof_25C.append(true_proof(density, 25))
true_proof_60F = np.array(true_proof_60F)
true_proof_20C = np.array(true_proof_20C)
true_proof_25C = np.array(true_proof_25C)

mole_frac_60F = []
mole_frac_20C = []
mole_frac_25C = []
for i in range(len(densities)):
    mole_frac_60F.append(mole_fraction(true_proof_60F[i]))
    mole_frac_20C.append(mole_fraction(true_proof_20C[i]))
    mole_frac_25C.append(mole_fraction(true_proof_25C[i]))
mole_frac_60F = np.array(mole_frac_60F)
mole_frac_20C = np.array(mole_frac_20C)
mole_frac_25C = np.array(mole_frac_25C)

plt.plot(densities, mole_frac_60F, "-", label="$60^\circ F$", color="blue")
plt.plot(densities, mole_frac_20C, "-", label="$20^\circ C$", color="green")
plt.plot(densities, mole_frac_25C, "-", label="$25^\circ C$", color="red")
plt.title(
    "Figure 1. Theoretical Conversion from\nDensity to Mole Fraction of Ethanol",
    **title
)
plt.xlabel("Density [g/cc]", **font)
plt.ylabel("Mole Fraction of Ethanol", **font)
plt.legend(prop=legend)
plt.show()

# Figure 2. true proof vs density for various temperatures
plt.plot(densities, true_proof_60F, "-", label="$60^\circ F$", color="blue")
plt.plot(densities, true_proof_20C, "-", label="$20^\circ C$", color="green")
plt.plot(densities, true_proof_25C, "-", label="$25^\circ C$", color="red")
plt.title("Figure 2. Theoretical Conversion from\nDensity to True Proof", **title)
plt.xlabel("Density [g/cc]", **font)
```

```
plt.ylabel("True Proof $[^\circ P]$", **font)
plt.legend(prop=legend)
plt.show()


# Figure 3. temperature corrected densinometer density vs pycnometer density
pycno_densities = []
pycno_temps = []
for pycno in pycnometer_measurments:
    pycno_densities.append(pycno.density)
    pycno_temps.append(pycno)
pycno_densities = np.array(pycno_densities)

dens_corrected_densities = []
for i in range(len(pycnometer_measurments)):
    dens_corrected_densities.append(
        temp_correct(
            ethanol_measurments[i].density.mean(),
            ethanol_measurments[i].dtemp.mean(),
            pycnometer_measurments[i].avg_temp,
            true_proof(
                pycnometer_measurments[i].density, pycnometer_measurments[i].avg_te
            ),
        )
    )
dens_corrected_densities = np.array(dens_corrected_densities)

x = np.linspace(0.8, 1.0, 100)
plt.plot(pycno_densities, dens_corrected_densities, "b.")
plt.plot(x, x, "k--", label="Line of 1:1 Correlation")
plt.xlabel("Pycnometer Density [g/cc]", **font)
plt.ylabel("Densitometer Density [g/cc]", **font)
plt.title(
    "Figure 3. Comparison of Densitometer\nand Pycnometer Density Measurments", **t
)
plt.legend(prop=legend)
plt.show()
```

**Figure 1. Theoretical Conversion from Density to Mole Fraction of Ethanol**

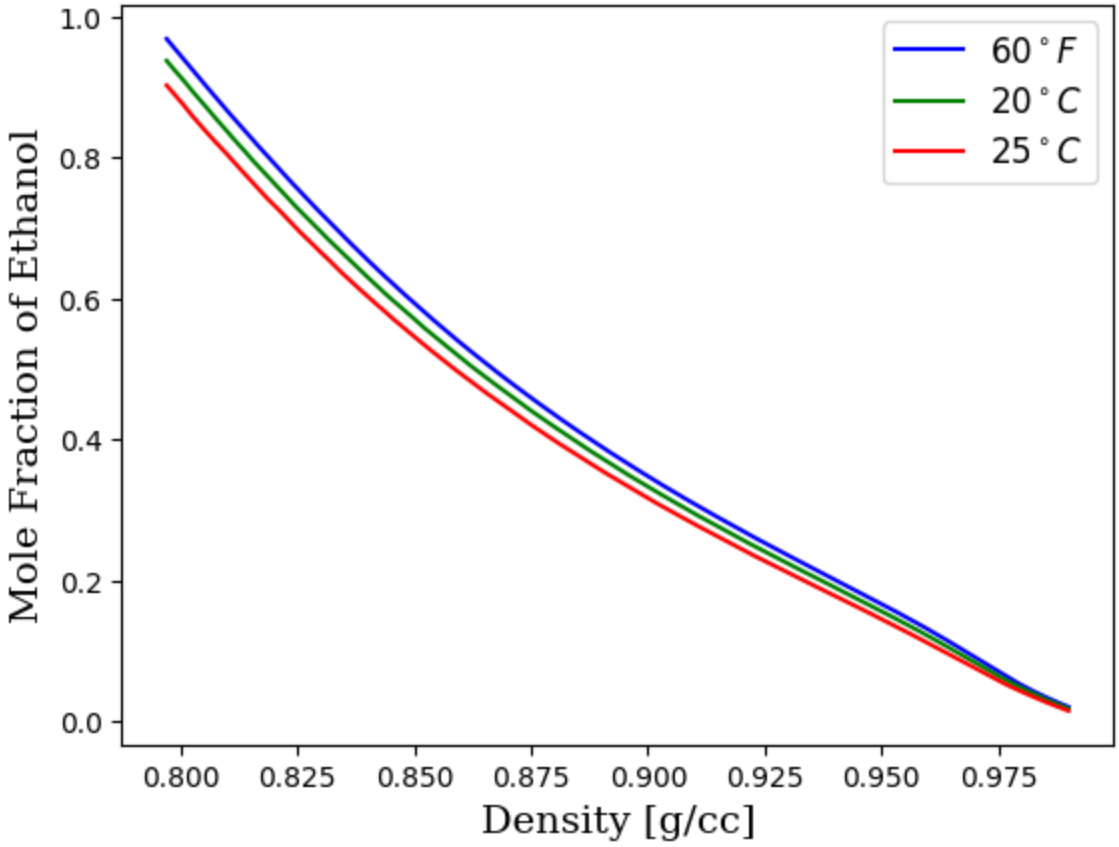
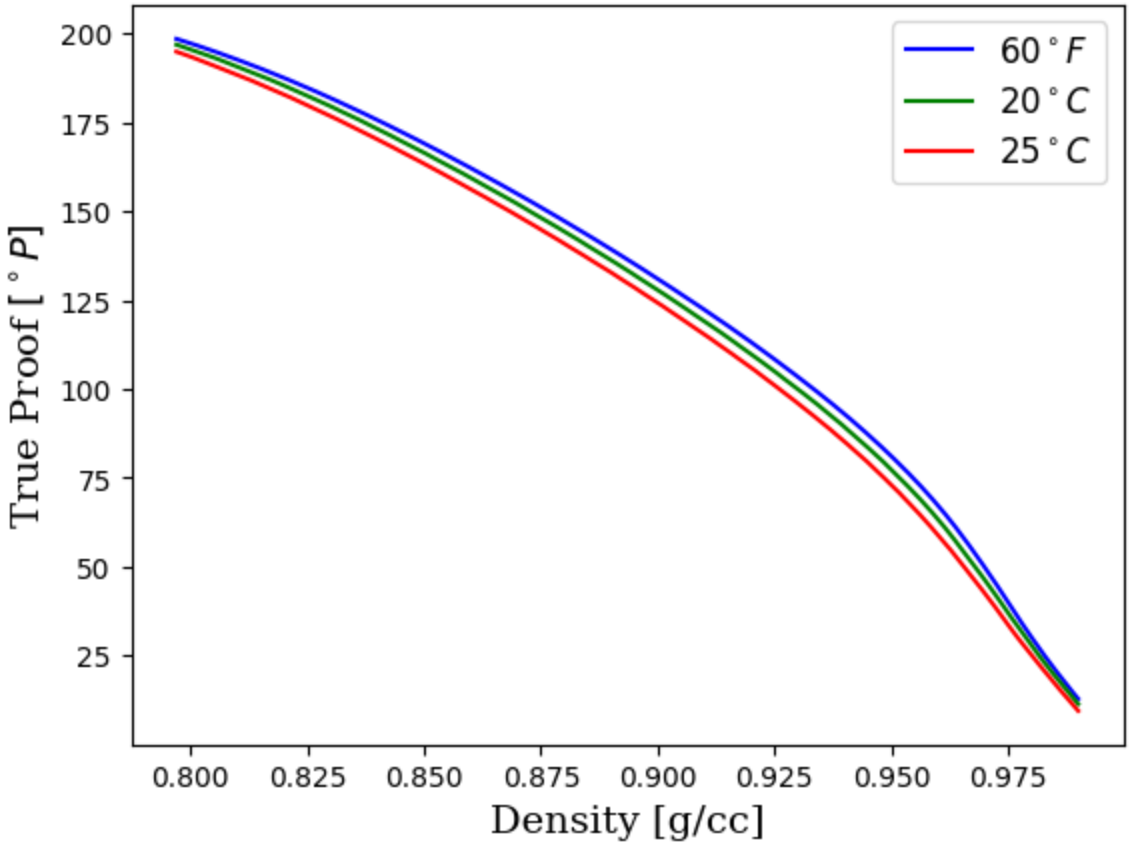
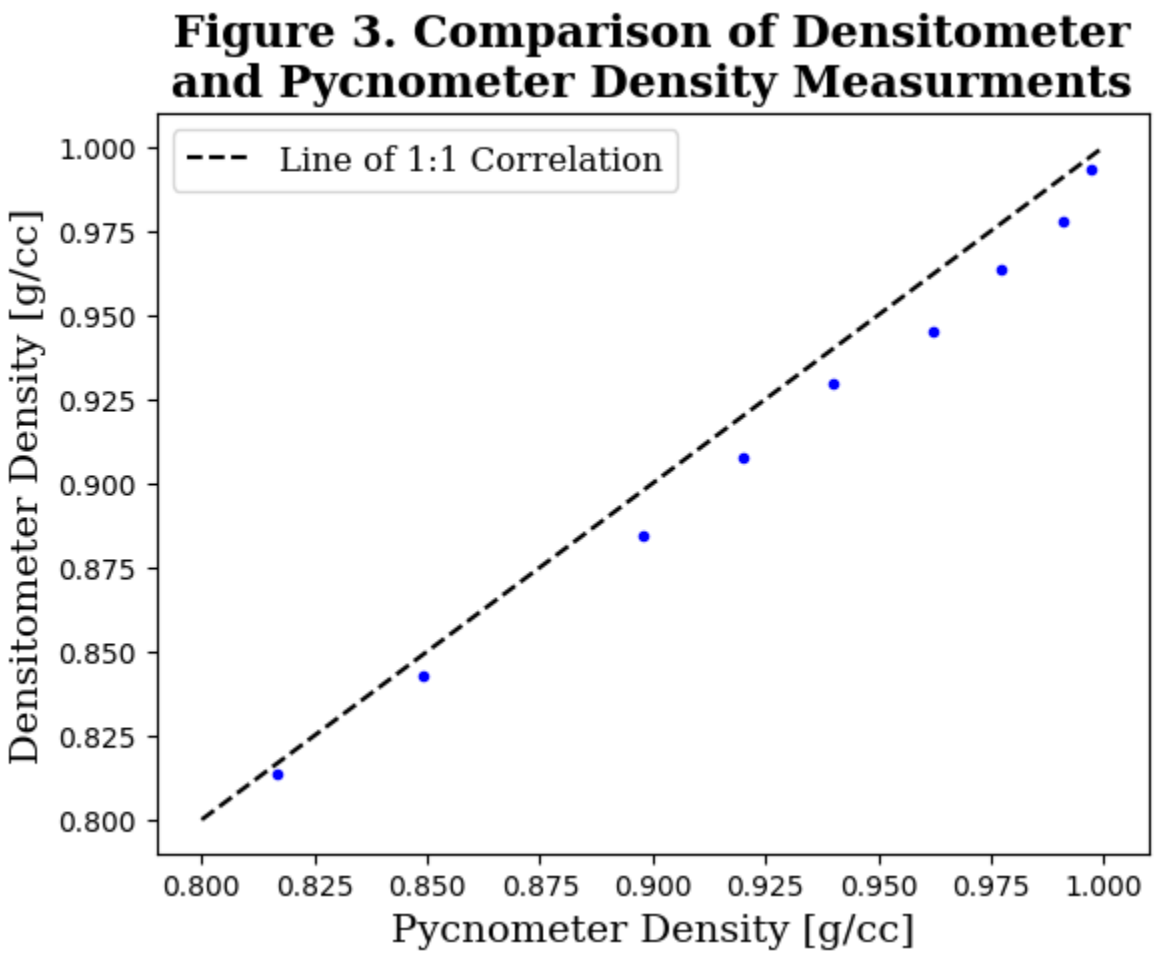


**Figure 2. Theoretical Conversion from Density to True Proof**






In [16]:

```python
# Antoine coefficients in mmHG and degrees Celcius [A, B, C]
antoine_water = [7.96681, 1668.21, 228.00]
antoine_eth = [8.04494, 1554.30, 222.650]

# Van Laar coefficients for ethanol (1) and water (2)
A_12 = 1.6798
A_21 = 0.9227


def antoine(T, A, B, C):
    """
    Returns vapor pressure in mmHG from a temperature T in degrees Celcius
    and a substance's antoine coefficients
    """

    log_P = A - B / (T + C)
    P = 10**log_P
    return P


def van_laar(x_1):
    x_2 = 1 - x_1

    log_gamma_1 = A_12 * ((A_21 * x_2) / (A_12 * x_1 + A_21 * x_2)) ** 2
    log_gamma_2 = A_21 * ((A_12 * x_1) / (A_12 * x_1 + A_21 * x_2)) ** 2
```

```
    gamma_1 = np.exp(log_gamma_1)
    gamma_2 = np.exp(log_gamma_2)

    return gamma_1, gamma_2


def vle(T, x_1, P):
    P1_sat = antoine(T, *antoine_eth)
    P2_sat = antoine(T, *antoine_water)

    gamma_eth, gamma_water = van_laar(x_1)

    # From equation 2a
    y_a = gamma_eth * x_1 * P1_sat / P

    # From equation 2b
    y_b = 1 - (gamma_water * (1 - x_1) * P2_sat) / P

    return y_a - y_b


def calc_y(T, x_1, P):
    P1_sat = antoine(T, *antoine_eth)
    gamma_eth, gamma_water = van_laar(x_1)

    y_a = gamma_eth * x_1 * P1_sat / P

    return y_a
```

In [17]:

```
@dataclass
class CollumnMeasurment:
    reflux: float64
    x_prod: float64
    x_waste: float64
    x_feed: float64
    x_P2: float64
    x_P4: float64
    x_P6: float64
    x_P7: float64
    P_drop: float64  # Pa
    boil_pow: float64  # W
    chill_pow: float64  # W
    mass_feed: ndarray[float64]  # g/s
    mass_waste: ndarray[float64]  # g/s
    mass_prod: ndarray[float64]  # g/s


def new_collumn_measurement(
    reflux: float64,
    feed_density: ndarray[float64],
    feed_temp: ndarray[float64],
    waste_density: ndarray[float64],
    waste_temp: ndarray[float64],
    prod_density: ndarray[float64],
    prod_temp: ndarray[float64],
    feed_flow: ndarray[float64],
```

```
    feed_time: ndarray[float64],
    waste_flow: ndarray[float64],
    waste_time: ndarray[float64],
    prod_flow: ndarray[float64],
    prod_time: ndarray[float64],
    P2_density: ndarray[float64],
    P2_temp: ndarray[float64],
    P4_density: ndarray[float64],
    P4_temp: ndarray[float64],
    P6_density: ndarray[float64],
    P6_temp: ndarray[float64],
    P7_density: ndarray[float64],
    P7_temp: ndarray[float64],
    P_drop: float64,
    boil_pow: float64,
    chill_pow: float64,
):
    x_prod = mole_fraction(true_proof(prod_density.mean(), prod_temp.mean()))
    x_waste = mole_fraction(true_proof(waste_density.mean(), waste_temp.mean()))
    x_feed = mole_fraction(true_proof(feed_density.mean(), feed_temp.mean()))
    x_P2 = mole_fraction(true_proof(P2_density.mean(), P2_temp.mean()))
    x_P4 = mole_fraction(true_proof(P4_density.mean(), P4_temp.mean()))
    x_P6 = mole_fraction(true_proof(P6_density.mean(), P6_temp.mean()))
    x_P7 = mole_fraction(true_proof(P7_density.mean(), P7_temp.mean()))

    mass_feed = feed_flow / feed_time
    mass_prod = prod_flow / prod_time
    mass_waste = waste_flow / waste_time

    P_drop = P_drop * 9.80665  # Convert mmH2O to Pa
    boil_pow = boil_pow * 1000  # Convert kW to W

    return ColumnMeasurment(
        reflux,
        x_prod,
        x_waste,
        x_feed,
        x_P2,
        x_P4,
        x_P6,
        x_P7,
        P_drop,
        boil_pow,
        chill_pow,
        mass_feed,
        mass_waste,
        mass_prod,
    )


collumn4 = new_collumn_measurement(
    4.0,
    np.array([0.9769, 0.9761, 0.9753]),
    np.array([22.9, 23.5, 24.5]),
    np.array(
        [
```

```
            0.9805,
            0.9820,
            0.9813,
            0.9795,
            0.9809,
            0.9818,
            0.9812,
            0.9793,
            0.9806,
            0.9787,
            0.9913,
            0.9826,
            0.9820,
        ]
    ),
    np.array(
        [27.1, 26.0, 26.0, 23.8, 23.7, 23.7, 25.2, 25.4, 25.3, 17.2, 17.9, 19.5, 20
    ),
    np.array(
        [
            0.8311,
            0.8286,
            0.8262,
            0.8242,
            0.8197,
            0.8180,
            0.8199,
            0.8188,
            0.8186,
            0.8236,
            0.8175,
            0.8160,
            0.8159,
        ]
    ),
    np.array(
        [18.2, 15.7, 15.2, 23.4, 23.5, 23.5, 22.6, 22.6, 22.7, 22.1, 23.8, 24.8, 25
    ),
    np.array([66.765]),
    np.array([60.0]),
    np.array([15.293, 19.22, 21.607]),
    np.array([14.11, 15.63, 20.91]),
    np.array([4.95, 6.25, 5.479]),
    np.array([64.82, 38.18, 24.5]),
    np.array([0.8392, 0.8389, 0.8364]),
    np.array([16.5, 17.1, 17.4]),
    np.array([0.8957, 0.8921, 0.8890]),
    np.array([5.8, 6.3, 8.8]),
    np.array([0.9636, 0.9635, 0.9634]),
    np.array([7.3, 7.1, 7.4]),
    np.array([0.9641, 0.9643, 0.9672]),
    np.array([11.2, 13.7, 10.2]),
    90.0,
    0.74,
    734,
)
```

```
collumn3 = new_collumn_measurement(
    3.0,
    np.array([0.9775, 0.9775]),
    np.array([20.4, 20.4]),
    np.array([0.9824, 0.9824, 0.9820, 0.9824, 0.9812, 0.9832, 0.9824]),
    np.array([23.7, 24.1, 25.0, 23.0, 24.5, 21.6, 23.7]),
    np.array([0.8266, 0.8223, 0.8195, 0.8192, 0.8190, 0.8189]),
    np.array([21.2, 21.5, 21.6, 22.1, 22.2, 22.1]),
    np.array([0.0]),
    np.array([1.0]),
    np.array([34.155]),
    np.array([1.0]),
    np.array([4.885]),
    np.array([1.0]),
    np.array([0.8507, 0.8467, 0.8392, 0.8386, 0.8375]),
    np.array([16.1, 17.5, 19.2, 19.5, 20.0]),
    np.array([0.8781, 0.8798, 0.8813, 0.8819]),
    np.array([14.4, 13.8, 21.2, 21.8]),
    np.array([0.9606, 0.9618, 0.9618, 0.9533, 0.9564, 0.9563]),
    np.array([21.0, 18.2, 18.1, 22.6, 21.6, 22.2]),
    np.array([0.9535, 0.9574, 0.9595, 0.9576, 0.9575, 0.9579]),
    np.array([25.1, 24.1, 20.7, 24.8, 25.1, 25.1]),
    96.0,
    0.74,
    733,
)
```

In [29]:

```
def operating_line(x_vals, reflux, x_prod):
    return (reflux / (reflux + 1)) * x_vals + (x_prod / (reflux + 1))


def stripping_line(x_vals, waste_x, waste_y, feed_x, feed_y):
    return ((waste_y - feed_y) / (waste_x - feed_x)) * (x_vals - feed_x) + feed_y


x_vals = np.linspace(0, 1, 1000)
guess_T = 50  # degC
# Data for pressure taken from:
# https://www.weather.gov/wrh/timeseries?site=KEDU
# Pressures taken are reported sation pressures
# reflux 4 on Feb 22, 2023 at 2:55pm
# reflux 3 on Mar 01, 2023 at 2:55pm
reflux4_P = 29.88 * 25.4  # mmHg
reflux3_P = 29.64 * 25.4  # mmHg

sol4_T = []
sol4_y = []
sol3_T = []
sol3_y = []
for x in x_vals:
    sol4 = scipy.optimize.root(vle, guess_T, args=(x, reflux4_P))
    sol4_T.append(sol4.x)
    sol4_y.append(calc_y(sol4.x, x, reflux4_P))

    sol3 = scipy.optimize.root(vle, guess_T, args=(x, reflux3_P))
```

```
    sol3_T.append(sol3.x)
    sol3_y.append(calc_y(sol3.x, x, reflux3_P))
sol4_T = np.array(sol4_T)
sol4_y = np.array(sol4_y)
sol3_T = np.array(sol3_T)
sol3_y = np.array(sol3_y)

op_x = np.linspace(collumn3.x_feed, collumn3.x_prod, 1000)
op_line3 = operating_line(op_x, collumn3.reflux, collumn3.x_prod)
op_line3_full = operating_line(x_vals, collumn3.reflux, collumn3.x_prod)
y_feed3 = operating_line(collumn3.x_feed, collumn3.reflux, collumn3.x_prod)
strip_x = np.linspace(collumn3.x_waste, collumn3.x_feed, 1000)
strip_line3 = stripping_line(
    strip_x, collumn3.x_waste, collumn3.x_waste, collumn3.x_feed, y_feed3
)
strip_line3_full = stripping_line(
    x_vals, collumn3.x_waste, collumn3.x_waste, collumn3.x_feed, y_feed3
)


plate3_y = [
    y := collumn3.x_prod,
    y,
    y := op_line3_full[yy := find_nearest(sol4_y, y)],
    y,
    y := op_line3_full[yy := find_nearest(sol4_y, op_line3_full[yy])],
    y,
    y := op_line3_full[yy := find_nearest(sol4_y, op_line3_full[yy])],
    y,
    y := op_line3_full[yy := find_nearest(sol4_y, op_line3_full[yy])],
    y,
    y := strip_line3_full[yy := find_nearest(sol4_y, op_line3_full[yy])],
    y,
]
plate3_x = [
    x := collumn3.x_prod,
    x := x_vals[xx := find_nearest(sol4_y, x)],
    x,
    x := x_vals[xx := find_nearest(sol4_y, op_line3_full[xx])],
    x,
    x := x_vals[xx := find_nearest(sol4_y, op_line3_full[xx])],
    x,
    x := x_vals[xx := find_nearest(sol4_y, op_line3_full[xx])],
    x,
    x := x_vals[xx := find_nearest(sol4_y, op_line3_full[xx])],
    x,
    x_vals[find_nearest(sol4_y, y)],
]

font = dict(family="serif", size=14)
legend = dict(family="serif", size=12)
title = dict(family="serif", size=16, weight="bold")

plt.figure(figsize=(10, 15), dpi=200)

plt.subplot(2, 1, 1)
```

```
plt.plot(x_vals, sol3_y, "b", label="VLE curve")
plt.plot(x_vals, x_vals, "k", label="x = y line")
plt.plot(column3.x_prod, column3.x_prod, "ro")
plt.plot(column3.x_waste, column4.x_waste, "ro")
plt.plot(op_x, op_line3, "r--", label="Rectifying line")
plt.plot(column3.x_feed, y_feed3, "ro")
plt.plot(strip_x, strip_line3, "r:", label="Stripping line")
plt.plot(plate3_x, plate3_y, "g-", label="Ideal stage lines")
plt.plot(
    column3.x_P2,
    sol4_y[find_nearest(x_vals, column3.x_P2)],
    "rx",
    label="Measured Tray Compositions [2, 4, 6, 7]",
)
plt.plot(column3.x_P4, sol4_y[find_nearest(x_vals, column3.x_P4)], "rx")
plt.plot(column3.x_P6, sol4_y[find_nearest(x_vals, column3.x_P6)], "rx")
plt.plot(column3.x_P7, sol4_y[find_nearest(x_vals, column3.x_P7)], "rx")
plt.title("Figure 4. McCabe Thiele Diagrams\nReflux Ratio = 3", **title)
plt.xlabel("Ethanol Concentration in Liquid Phase", **font)
plt.ylabel("Ethanol Concentration in Vapor Phase", **font)
plt.legend(prop=legend)


op_x = np.linspace(column4.x_feed, column4.x_prod, 1000)
op_line4 = operating_line(op_x, column4.reflux, column4.x_prod)
op_line4_full = operating_line(x_vals, column4.reflux, column4.x_prod)
y_feed4 = operating_line(column4.x_feed, column4.reflux, column4.x_prod)
strip_x = np.linspace(column4.x_waste, column4.x_feed, 1000)
strip_line4 = stripping_line(
    strip_x, column4.x_waste, column4.x_waste, column4.x_feed, y_feed4
)
strip_line4_full = stripping_line(
    x_vals, column4.x_waste, column4.x_waste, column4.x_feed, y_feed4
)

plate4_x = [
    x := column4.x_prod,
    x := x_vals[xx := find_nearest(sol4_y, x)],
    x,
    x := x_vals[xx := find_nearest(sol4_y, op_line4_full[xx])],
    x,
    x := x_vals[xx := find_nearest(sol4_y, op_line4_full[xx])],
    x,
    x := x_vals[xx := find_nearest(sol4_y, op_line4_full[xx])],
    x,
    x := x_vals[xx := find_nearest(sol4_y, op_line4_full[xx])],
    x,
]
plate4_y = [
    y := column4.x_prod,
    y,
    y := op_line4_full[yy := find_nearest(sol4_y, y)],
    y,
    y := op_line4_full[yy := find_nearest(sol4_y, op_line4_full[yy])],
    y,
    y := op_line4_full[yy := find_nearest(sol4_y, op_line4_full[yy])],
```

```
    y,
    y := op_line4_full[yy := find_nearest(sol4_y, op_line4_full[yy])],
    y,
    x,
]

plate4_lconcs = [collumn4.x_P2, collumn4.x_P4, collumn4.x_P6, collumn4.x_P7]

plt.subplot(2, 1, 2)
plt.plot(x_vals, sol4_y, "b", label="VLE curve")
plt.plot(x_vals, x_vals, "k", label="x = y line")
plt.plot(collumn4.x_prod, collumn4.x_prod, "ro")
plt.plot(collumn4.x_waste, collumn4.x_waste, "ro")
plt.plot(op_x, op_line4, "r--", label="Rectifying line")
plt.plot(collumn4.x_feed, y_feed4, "ro", label="")
plt.plot(strip_x, strip_line4, "r:", label="Stripping line")
plt.plot(plate4_x, plate4_y, "g-", label="Ideal stage lines")
plt.plot(
    collumn4.x_P2,
    sol4_y[find_nearest(x_vals, collumn4.x_P2)],
    "rx",
    label="Measured Tray Compositions [2, 4, 6, 7]",
)
# plt.plot(plate4_lconcs, plate_concs4, "gX")
plt.plot(collumn4.x_P4, sol4_y[find_nearest(x_vals, collumn4.x_P4)], "rx")
plt.plot(collumn4.x_P6, sol4_y[find_nearest(x_vals, collumn4.x_P6)], "rx")
plt.plot(collumn4.x_P7, sol4_y[find_nearest(x_vals, collumn4.x_P7)], "rx")
plt.xlabel("Ethanol Concentration in Liquid Phase", **font)
plt.ylabel("Ethanol Concentration in Vapor Phase", **font)
plt.title("Reflux Ratio = 4", **title)
plt.legend(prop=legend)
# plt.tight_layout()
plt.show()
```

**Figure 4. McCabe Thiele Diagrams**