Getting Started

Jupyter Notebooks, Python, and Google Colaboratory

Summary

The purpose of this notebook is to introduce Jupyter Notebooks, Python, and Google Colaboratory for routine engineering calculations. This introduction assumes this is your first exposure to these topics.

Jupyter Notebooks are interactive documents, like this one, that include text and code. The documents are ordinary files with a suffix `.ipynb`. They can be uploaded, downloaded, and shared like any other digital document. The notebooks are composed of individual cells containing either text, code, or the output of a calculation. The code cells be written in different programming languages such as Python, R, and Julia.

Python is a high-level, object-oriented programming language well suited to the rapid development of web and scientific applications. It is widely used for interactive science and engineering computations where productivity is a more important than achieving the highest levels of numerical performance. Python refers to the language itself plus a large eco-system of development tools, documentation, and code libraries for numerical computations, data management, and graphics.

Google Colaboratory is Google's cloud environment for viewing and executing Jupyter/Python notebooks. It operates like Google docs, providing for the collaborative creation and editing of notebooks, and allowing notebooks to be saved to your Google Drive account. Because Colaboratory is based entirely in the cloud, there is no need to install any local software. Colaboratory requires only a browser and internet access to view and execute notebooks.

There other software development environments well suited to viewing and executing the notebooks in this repository. Among them are the excellent distributions

- Anaconda available from Continuum Analytics.
- Enthought Canopy available from Enthought, Inc.

which include tools to view and edit Jupyter notebooks, curated versions of all the major Python code libraries, and additional software development tools. You will want to install one of these distributions if you need off-line access to these notebooks, or if you intend to do more extensive development.

Setting up Google Colaboratory and Google Drive

The easiest way to get started with Google Colaboratory is to open a notebook. If you are not currently reading this notebook on Colaboratory, open it here from the github repository.

Google Colaboratory is tightly integrated with Google Drive. Notebooks can be opened directly from Google Drive and automatically saved back to Google Drive. Notebooks can be shared, and support mutiple users collaborating simultaenously the same notebook. These features will be familiar if you've previously used Google Docs. You can obtain a Google Drive or a Google One account here if you don't already have one. The free tier offers 15GB of storage which is enough for routine use of these notebooks.

When opened from a third party resource such as github, the notebook is initially in `playground mode`. This is fine for viewing notebooks and executing code cells. To save your work you will need to save a copy of the notebook. You can save a copy to:

- your Google Drive
- a repository in your own github account
- download the notebook to your personal device

Python Basics

Python is an elegant and modern language for programming and problem solving that has found increasing use by engineers and scientists. In the next few cells we'll demonstrate some basic Python functionality.

Arithmetic Operations

Basic arithmetic functions

In [6]:

```
a = 12
b = 2
print("a + b = ", a + b)
print("a**b = ", a**b)
print("a/b = ", a/b)
```

```
a + b =  14
a**b =  144
a/b =  6.0
```

Most math functions are in the `numpy` library. This next cell shows how to import `numpy` with the prefix `np`, then use it to call a common function

In [7]:

```
import numpy as np
np.sin(2*np.pi)
```

Out[7]:

```
-2.4492935982947064e-16
```

Lists

Lists are a versatile way of organizing your data in Python. Here are some examples, more can be found on this Khan Academy video.

In [8]:

```
xList = [1, 2, 3, 4]
print(xList)
```

```
[1, 2, 3, 4]
```

Concatentation is the operation of joining one list to another.

In [9]:

```
# Concatenation
x = [1, 2, 3, 4];
y = [5, 6, 7, 8];

x + y
```

Out[9]:

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

A common operation is to apply a binary operation to elements of a single list. For an example such as arithmetic addition, this can be done using the `sum` function from `numpy`.

In [10]:

```
# Two ways to sum a list of numbers
print(np.sum(x))
```

```
10
```

A for loop is a means for iterating over the elements of a list. The colon marks the start of code that will be executed for each element of a list. Indenting has meaning in Python. In this case, everything in the indented block will be executed on each iteration of the for loop.

In [11]:

```
for x in xList:
    print("x =", x, "    sin(x) = ", np.sin(x))
```

```
x = 1     sin(x) =  0.8414709848078965
x = 2     sin(x) =  0.9092974268256817
x = 3     sin(x) =  0.1411200080598672
x = 4     sin(x) =  -0.7568024953079282
```

A **list comprehension** is a very useful tool for creating a new list using data from an existing. For example, suppose you have a list consisting random numbers

In [12]:

```
from random import random

[i + random() for i in range(0,10)]
```

Out[12]:

```
[0.1952325947356377,
 1.5664950727528784,
 2.8645620994056857,
 3.610813783399313,
 4.745590841711996,
 5.929632460883009,
 6.0790789599780926,
 7.643940863853748,
 8.721811826350383,
 9.506420667869708]
```

Dictionaries

Dictionaries are useful for storing and retrieving data as key-value pairs. For example, here is a short dictionary of molar masses. The keys are molecular formulas, and the values are the corresponding molar masses.

In [13]:

```
mw = {'CH4': 16.04, 'H2O': 18.02, 'O2':32.00, 'CO2': 44.01}
print(mw)
```

```
{'CH4': 16.04, 'H2O': 18.02, 'O2': 32.0, 'CO2': 44.01}
```

We can a value to an existing dictionary.

In [14]:

```
mw['C8H18'] = 114.23
print(mw)
```

```
{'CH4': 16.04, 'H2O': 18.02, 'O2': 32.0, 'CO2': 44.01, 'C8H18': 114.23}
```

We can retrieve a value from a dictionary.

In [15]:

```
mw['CH4']
```

Out[15]:

```
16.04
```

A for loop is a useful means of interating over all key-value pairs of a dictionary.

In [16]:

```
for species in mw.keys():
    print("The molar mass of {:<s} is {:<7.2f}".format(species, mw[species]))
```

```
The molar mass of CH4 is 16.04
The molar mass of H2O is 18.02
The molar mass of O2 is 32.00
The molar mass of CO2 is 44.01
The molar mass of C8H18 is 114.23
```

Dictionaries can be sorted by key or by value

In [17]:

```
for species in sorted(mw):
    print(" {:<8s}  {:>7.2f}".format(species, mw[species]))
```

```
 C8H18      114.23
 CH4         16.04
 CO2         44.01
 H2O         18.02
 O2          32.00
```

In [18]:

```
for species in sorted(mw, key = mw.get):
    print(" {:<8s}  {:>7.2f}".format(species, mw[species]))
```

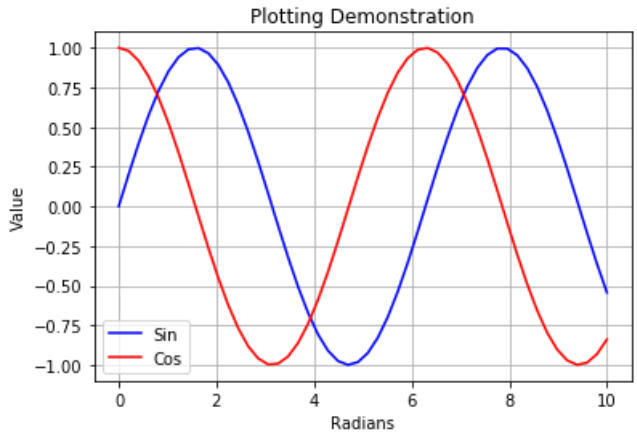
```
 CH4          16.04
 H2O          18.02
 O2           32.00
 CO2          44.01
 C8H18       114.23
```

Plotting

Importing the `matplotlib.pyplot` library gives IPython notebooks plotting functionality very similar to Matlab's. Here are some examples using functions from the

In [21]:

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

x = np.linspace(0,10)
y = np.sin(x)
z = np.cos(x)

plt.plot(x,y,'b',x,z,'r')
plt.xlabel('Radians');
plt.ylabel('Value');
plt.title('Plotting Demonstration')
plt.legend(['Sin','Cos'])
plt.grid(True)
```

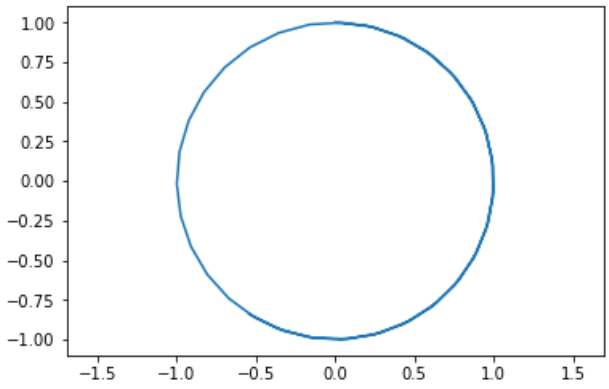


In [22]:

```
plt.plot(y,z)
plt.axis('equal')
```

Out[22]:

```
(-1.09972447591003,
 1.0979832896606587,
 -1.0992804688576738,
 1.0999657366122702)
```

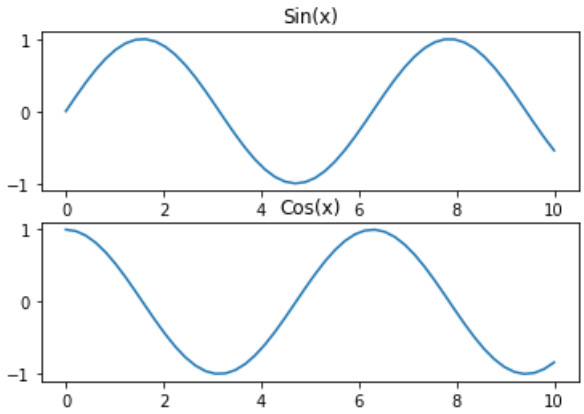


In [23]:

```
plt.subplot(2,1,1)
plt.plot(x,y)
plt.title('Sin(x)')

plt.subplot(2,1,2)
plt.plot(x,z)
plt.title('Cos(x)')
```

Out[23]:

```
Text(0.5, 1.0, 'Cos(x)')
```



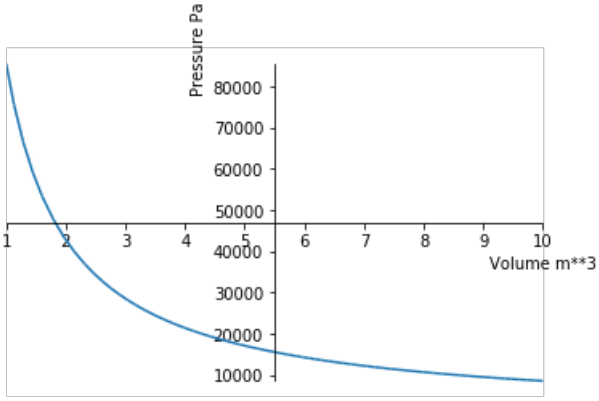
Solving Equations using Sympy

One of the best features of Python is the ability to extend it's functionality by importing special purpose libraries of functions. Here we demonstrate the use of a symbolic algebra package `Sympy` for routine problem solving.

In [24]:

```
import sympy as sym

sym.var('P V n R T');

# Gas constant
R = 8.314          # J/K/gmol
R = R * 1000       # J/K/kgmol

# Moles of air
mAir = 1           # kg
mwAir = 28.97      # kg/kg-mol
n = mAir/mwAir     # kg-mol

# Temperature
T = 298

# Equation
eqn = sym.Eq(P*V,n*R*T)

# Solve for P
f = sym.solve(eqn,P)
print(f[0])

# Use the sympy plot function to plot
sym.plot(f[0],(V,1,10),xlabel='Volume m**3',ylabel='Pressure Pa')
```

```
85521.9882637211/V
```



Out[24]:

```
<sympy.plotting.plot.Plot at 0x12027dac8>
```

Defining your own Functions

Many of the physical properties of chemical species are given as functions of temperature or pressure.

Learn More

Python offers a full range of programming language features, and there is a seemingly endless range of packages for scientific and engineering computations. Here are some suggestions on places you can go for more information on programming for engineering applications in Python.

Tutorial Introductions to Python for Science and Engineering

This excellent introduction to python is aimed at undergraduates in science with no programming experience. It is free and available at the following link.

- Introduction to Python for Science

The following text is licensed by the Hesburgh Library for use by Notre Dame students and faculty only. Please refer to the library's acceptable use policy. Others can find it at Springer or Amazon. Resources for this book are available on github.

- A Primer on Scientific Programming with Python (Fourth Edition) by Hans Petter Langtangen. Resources for this book are available on github.

pycse is a package of python functions, examples, and document prepared by John Kitchin at Carnegie Mellon University. It is a recommended for its coverage of topics relevant to chemical engineers, including a chapter on typical chemical engineering computations.

- pycse - Python Computations in Science and Engineering by John Kitchin at Carnegie Mellon. This is a link into the the github repository for pycse, click on the `Raw` button to download the `.pdf` file.

Interactive Learning and On-Line Tutorials

- Code Academy on Python
- Khan Academy Videos on Python Programming
- Python Tutorial
- Think Python: How to Think Like a Computer Scientist
- Engineering with Python

Official documentation, examples, and galleries

- Notebook Examples
- Notebook Gallery
- Official Notebook Documentation
- Matplotlib

Solving Linear Equations with Simpy

Summary

This notebook shows how to solve linear equations corresponding to material balances on chemical processes using the {ython symbolic algebra library Sympy. The example is adapted with permission from learnCheme.com, a project at the University of Colorado funded by the National Science Foundation and the Shell Corporation.

Problem Statement

In [1]:

```
from IPython.display import YouTubeVideo
YouTubeVideo("KrrZB5LvXF4",560,315,start=0,end=144,rel=0)
```

Out[1]:

Before going further, be sure you can solve a system of two (or three) linear equations in two (or three) unknowns with paper and pencil. Most of you will have seen these problems before in your math classes.

Solving linear equations using Sympy

Assuming you have mastered the solution of linear equations with paper and pencil, let's see how to find solutions using the python symbolic algebra library Sympy.

Sympy is an example of a Python 'library'. The first step in using the library is to import it into the current workspace. It is customary to import into the workspace with the namespace `sym` to avoid name clashes with variables and functions.

In [ ]:

```
import sympy as sym
```

The system of equations to be solved is given by

$$\begin{aligned} n_1 + n_1 &= 100 \\ 0.7n_1 + 0.2n_2 &= 30 \end{aligned}$$

The next step is to introduces names for the unknown variables appearing in our problem. The Sympy function `sym.var()` constructs symbolic variables given a list of variable names.

In [3]:

```
sym.var(['n1','n2'])
```

Out[3]:

```
[n1, n2]
```

The newly constructed symbolic variable are used to create symbolic equations. The sympy function `sym.Eq()` accepts two arguments, each a symbolic expression expressing the left and right hand sides of an equation. For this problem there are two equations to be solved simultaneously, so we construct both and store them in a python list.

In [4]:

```
eqns = [
    sym.Eq(n1 + n2, 100),
    sym.Eq(0.7*n1 + 0.2*n2, 30)
]
print(eqns)
```

```
[Eq(n1 + n2, 100), Eq(0.7*n1 + 0.2*n2, 30)]
```

The last step is solve the equations using `sym.solve()`.

In [5]:

```
soln = sym.solve(eqns)
print(soln)
```

```
{n1: 20.0000000000000, n2: 80.0000000000000}
```

Putting these steps together, we have a three-step procedure for solving systems of linear equations using Sympy.

In [6]:

```
# import sympy
import sympy as sym

# Step 1. Create symbolic variables.
sym.var(['n1','n2'])

# Step 2. Create a list of equations using the symbolic variables
eqns = [
    sym.Eq(n1 + n2, 100),
    sym.Eq(0.7*n1 + 0.2*n2, 30)
]

# Step 3. Solve and display solution
soln = sym.solve(eqns)
print(soln)
```

```
{n1: 20.0000000000000, n2: 80.0000000000000}
```

Exercise

In the cell below, prepare an IPython solution to the second problem described in the screencast involving three linear equations. The problem description starts at the 2:22 mark in the screencast. You can use the example above as a template for your solution.

In [7]:

```
from IPython.display import YouTubeVideo
YouTubeVideo("KrrZB5LvXF4",560,315,start=144,end=166,rel=0)
```

Out[7]:

In [ ]:

```

```

Chapter 1. Units, Quantities, and Engineering Calculations

1.1 Units and Engineering Calculations

Summary

The purpose of this Jupyter notebook is to get you started with units and engineering calculations.

Things to master:

- Unit systems
- Unit conversions
- Difference between mass and weight, and the difference between kg-m, kg-f, and lbm and lbf
- Concept of mole, and the difference between g-mol, kg-mol, and any other kind of mole
- Absolute versus Gauge Pressure
- Typical values of temperature and pressure in real-world applications
- Working with units in python using the `pint` library.

Units of Engineering

The Fundamental Units of Measurement

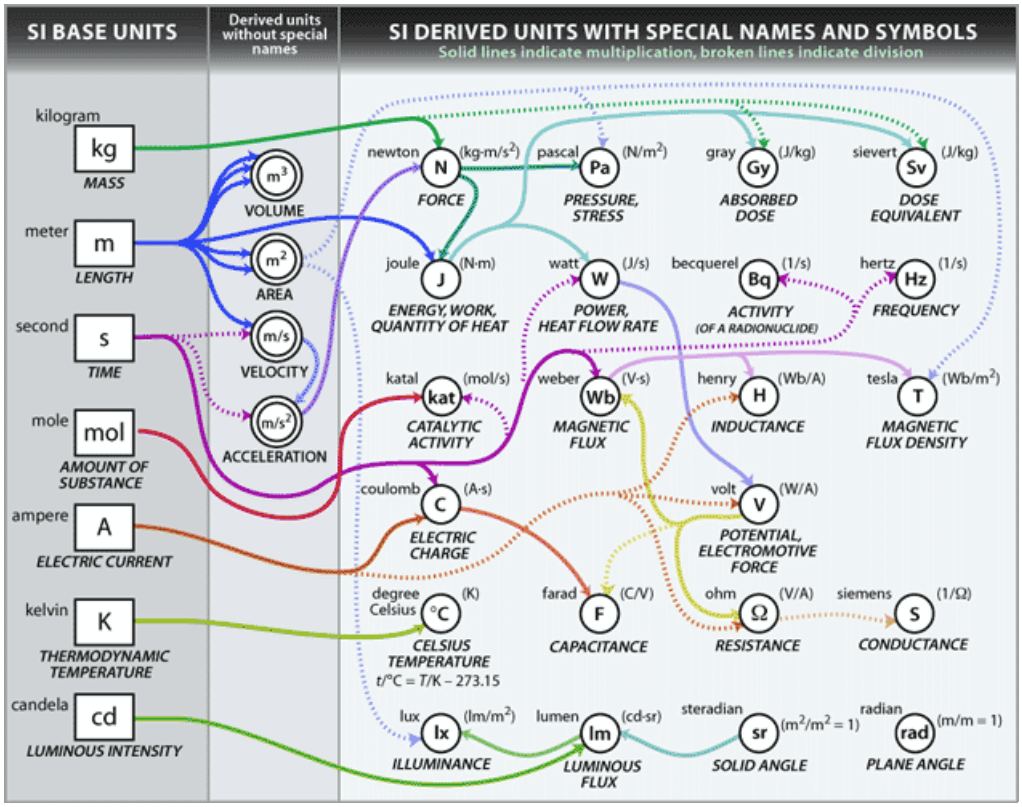
There are seven fundamental units in the Système International d'Unités (SI) system, all other unit can be generated from these.

Dimension	SI Units	CGS	English
Mass	kilogram (kg)	gram (g)	lb-mass, slug
Length	meter (m)	centimeter (cm)	inch (in); foot (ft)
Time	second (s)	second (s)	second (sec)
Temperature	degree Kelvin (K)	degree Celsius (°C)	degree Rankine (°R)
Quantity	mole (gmol)	mole (gmol)	mole (gmol)
Current	Ampere (A)	abampere	
Luminous Intensity	candela (cd)	stlib [luminance]	candlepower, etc.

> **Exercise.** Give an everyday example of something with about 1 SI unit of the corresonding dimension.

Coherent Derived Units

The SI system describes 22 'derived units' that are expressed in terms of the fundamental units. The following diagram from NIST illustrates the relationship among the SI units.



Calculations with coherent units is straightforward since they involve just ratios or products of base units. The calculations break down into three steps:

1. Recast all of the problem quantities in base or coherent derived units.
2. Perform the numerical calculations.
3. Using algebraic rules for the cancelation of terms, simplify and reduce units to units relevant to the context of the problem.

> **Example:** The kinetic energy of a object with mass $m = 3\ \text{kg}$ traveling at velocity $v = 10\ \text{m/s}$ is given by
>
> $$\begin{aligned} K.E. &= \frac{1}{2}mv^2 \\ &= \frac{1}{2}(3\ \text{kg})\left(10\frac{\text{m}}{\text{s}}\right)^2 \\ &= 150\ \frac{\text{kg}\cdot\text{m}^2}{\text{s}^2} = 150\ \text{N}\cdot\text{m} = \boxed{150\ \text{J}} \end{aligned}$$

Example: What is the mass of air in a spherical balloon with radius of 10 cm inflated to 1 atm at 25 degrees Celsius? Assume the molecular weight of air is 28.966 grams/mole, and that atmospheric pressure is 101,325 Pa.

Solution: Start by converting all of the quantities in the problem to base or derived units.

$$MW_{air} = 28.966\ \frac{\text{g}}{\text{mol}} = 0.028966\ \frac{\text{kg}}{\text{mol}}$$
$$P = 101,325\ \text{Pa} = 101,325\ \frac{\text{N}}{\text{m}^2} = 101,325\ \frac{\text{kg}}{\text{m}\cdot\text{s}^2}$$
$$V = \frac{4}{3}\pi r^3 = \frac{4}{3}\pi(0.10\ \text{m})^3 = 0.004189\ \text{m}^3$$
$$R = 8.314\frac{\text{J}}{\text{K}\cdot\text{mol}} = 8.314\frac{\text{kg}\cdot\text{m}^2}{\text{s}^2\cdot\text{K}\cdot\text{mol}}$$
$$T = 25 + 273.15 = 298.15K$$

Using molecular weight and the ideal gas law

$$m_{air} = MW_{air}n_{air} = MW_{air}\frac{PV}{RT}$$

Substituting quantities into this expression for m_{air}

$$m_{air} = \left(0.028966\ \frac{\text{kg}}{\text{mol}}\right)\frac{\left(101,325\ \frac{\text{kg}}{\text{m}\cdot\text{s}^2}\right)\left(0.004189\ \text{m}^3\right)}{\left(8.314\frac{\text{kg}\cdot\text{m}^2}{\text{s}^2\cdot\text{K}\cdot\text{mol}}\right)(298.15\ \text{K})}$$
$$= \frac{0.028966\cdot 101,325\cdot 0.004189}{8.314\cdot 298.15} \times \frac{\frac{\text{kg}}{\text{mol}}\cdot\frac{\text{kg}}{\text{m}\cdot\text{s}^2}\cdot\text{m}^3}{\frac{\text{kg}\cdot\text{m}^2}{\text{s}^2\cdot\text{K}\cdot\text{mol}}\cdot\text{K}}$$
$$= \boxed{0.00496\ \text{kg}}$$

Non-Coherent Derived Units

Coherent units are easy to work with because no numerical factors are required to express them in terms of the base units. On the other hand, working with such a limited set of units makes it difficult to express commonly quantities in easy-to-remember numbers or do back-of-the-envelop calculations. Having to write a multiple of 101,325 Pa to express commonly encountered pressures is just asking for computational errors.

Non-coherent derived units are quantities that can be expressed in terms of a numerical factor and product or ratio of base units. Here are some common examples:

Absolute versus Offset Scales

For the purpose of applying the ideal gas law, what is the air pressure in this tire?

The quantities used in most scientific calculations are measured on absolute scales, such as mass and length. On a absolute scale, a value of zero implies the absence of the underlying quantity. Something with zero mass has no mass, something with zero length means it has no length.

Some common engineering measurements, however, are reported 'relative to' or 'offset from' a reference value. For example, degrees Celsius measures temperature relative to the melting point of ice; degrees Fahrenheit measures temperature relative to a brisk winter day. In the figure above, tire pressure is being measured relative to pressure of the surrounding air. This is called the 'gauge pressure' because that is what is being measured by the pressure gauge.

The offsets result in numerical values convenient for common everyday use, but introduce an additional consideration (and source of errors!) for engineering calculations.

Examples of offset scales include:

- Degrees Celsius
- Degrees Fahrenheit
- Gauge Pressure (pressure relative to normal atmospheric pressure).
- Enthalpy and Free Energy of Formation (energy required to form a substance relative to standard laboratory conditions.)
- Calendar dates (days relative to start of the Common Era)

Engineers sometimes distinguish absolute and gauge pressures by adding an explicit suffix. In particular, psia refers to 'pounds per square inch absolute', while psig refers to 'pounds per square inch gauge'. Non-engineers almost never make these distinction, so you need to be very careful when interpreting information given to you! It's always your responsibility to resolve any ambiguity in the data given to you for analysis.

The misuse of pressure and temperature scales is a common source of errors in engineering calculations.

Example: A football is inflated to 12.5 psig (pounds per square in gauge) at 70 degrees F. The football is taken to play outdoors at 40 degrees F. Using the gas law,
$P_2 \qquad / \qquad T_2 =$

what is the pressure of the football at playing conditions?

Solution: The ideal gas law assumes quantities are given in absolute units. The problem statement, however, gives conditions in offset units. In this case, 0 degrees Farhenheit is 459.67 in absolute (Rankine) units, and 0 psig is 14.696 psia in absolute units. Incorporating the offsets the gas law reads

$$\frac{P_2 + 14.696}{40 + 459.67} = \frac{12.5 + 14.696}{70 + 459.67}$$

Solving for P_2

$$P_2 = \frac{40 + 459.67}{70 + 459.67}(12.5 + 14.696) - 14.696 = 10.96 \text{ psig}$$

Extensive versus Intensive Quantities

An important distinction among measured quantities arises when we consider what happens when combining systems under consideration. For example, mixing two identical one-liter containers of an acidic solution yields two liters of the solution, twice the mass of solution, and twice the number of molecules of all types. If the solutions were at a high temperature, the result would have twice the thermal energy content. Quantities that *scale* in direct proportion to the size of sytem are *extensive* quantities.

Other quantities, such as the temperature of the mixture, concentration of acid in the mixture, or vapor pressure are independent of the size of a system. Quantities that do not scale with the size are *intensive* quantities.

This distinction is fundamental to process modeling and thermodynamics.

Exercise. Catagorize the following quantities as either extensive or intensive variables: Concentration, density, electrical charge, electrical resistivity, length, material hardness, mass, moles, pH, pressure, specific volume, temperature, viscosity, volume.

Ideal Gas Law Calculations

The ideal gas law is a work horse of engineering calculations.

$$PV = nRT$$

Despite its simplicity and familiarity, misuse of units and scale for pressure P and temperature T, and identifying correct values of the gas constant R, are common sources of errors in homework and exams. Taking time now to really master ideal gas law calculations will pay dividends down the road.

Absolute Temperature

Reference	F	C	R	K
Absolute Zero	-459.67	-273.15	0	0
Normal Boiling point of N_2				
Dry Ice	-109.3	-78.5		
Boiling Point of Propane	-43.6	-42		
Freezing Point of Water	32	0	491.67	273.15
STP				
Typical Room Temperature	68	20		
Pleasant Summer Day	77	25		
Human Body		37		
Normal Boiling Point of Water	212	100		
10 psig steam				
70 psig steam				
400 psig steam				

Absolute versus Gauge Pressure

Every engineer needs to know atmospheric pressure in common engineering and scientific units. If need to look up these numbers then you're doing it wrong.

Value	Unit
1	atm
14.696	pounds per square inch (absolute), psia
0	pounds per square inch (gauge), psig
101,325	Newtons per square meter (Pascals)
101.325	kiloPascals
1.01325	bars
1013.25	millibars
760	mm Hg
760	torr
29.92	in Hg
33.9	feet of water

Units of Pressure (to be completed)

PV has units of Energy (to be completed)

Table of Values for R (to be completed)

Working with Units in Python using the pint library

There are a number of python libraries that incorporate units into python calculations. Among these, pint is a relatively new library that builds on experience with earlier attempts.

In [ ]:

```
!pip install pint
```

```
Requirement already satisfied: pint in /Users/jeff/anaconda/lib/python3.5/site-packages
```

The core concept in pint is to work with a `unit registry`, which is created as follows

In [ ]:

```
from pint import UnitRegistry
ur = UnitRegistry()
```

Assigning Multiplicative Units

The unit registry provides a simple means to assign units using the multiplication operator. For example, here's how to compute the molarity of a sodium chloride solution in 58.44 grams of $NaCl$ (mw = 58.44) has been dissolved in water to form 3 liters of solution.

In [ ]:

```
# problem data
V = 3.0 * ur.liters
m = 58.44 * ur.grams
mw = 58.44 * ur.grams/ur.mol

# compute molarity
C = m/(mw*V)

print(C)
```

```
0.3333333333333333 mole / liter
```

Unit Conversion

Each variable with units has `to()` and `ito()` methods for converting the quantity to a desired set of units. The `to()` method is used to create a new variable by converting an existing variable to the indicated units.

In [ ]:

```
x = 0.5 * ur.kilograms/ur.gallon
y = x.to(ur.grams/ur.liter)

print(x)
print(y)
```

```
0.5 kilogram / gallon
132.08602617907425 gram / liter
```

The `ito()` method converts an existing variable 'in-place'.

In [ ]:

```
x = 0.5 * ur.kilograms/ur.gallon
x.ito(ur.grams/ur.liter)

print(x)
```

```
132.08602617907425 gram / liter
```

For example, here's how to compute the molarity of a sodium chloride solution in which 0.5 pounds of NaCl (mw = 58.44) has been dissolved in water to form 2 gallons of solution.

In [ ]:

```
# problem data
V = 3.0 * ur.gallons
m = 0.5 * ur.lbs
mw = 58.44 * ur.grams/ur.mol

# compute concentration
C = m/(mw*V)
print(C)

# convert to desired units and print
C = C.to(ur.mol/ur.liter)
print(C)

# convert to moles per gallon
C.ito(ur.mol/ur.gallon)
print(C)
```

```
0.0028519279032626055 mole * pound / gallon / gram
0.3417363316133261 mole / liter
1.2936127367100163 mole / gallon
```

Muratic Acid is concentrated hydrochloric acid (31.5% by weight) sold by the gallon in home improvement centers for cleaning brick and masonary surfaces. The density of the solution is typically about 1.15 grams/ml. What is the molar concentration of HCl?

In [ ]:

```
# molecular weight
mwHCl = 36.46 * ur.grams/ur.mol

# problem data
rho = 1.15 * ur.grams/ur.ml
wHCl = 0.315 * ur.grams/ur.gram

# calculations
massHCl = wHCl * rho              # mass of HCl per volume
moleHCl = massHCl/mwHCl           # gmols of HCl per volume

print(moleHCl)

# convert to desired units
moleHCl.ito(ur.mol/ur.liter)
print(moleHCl)
```

```
0.009935545803620405 mole / milliliter
9.935545803620405 mole / liter
```

Mole and Mass Fractions

In [ ]:

```
# molar weights
mwBen = 78.11 * ur.grams/ur.liter
mwTol = 92.14 * ur.grams/ur.liter

xBen = 0.4 * ur.mol/ur.mol
xTol = 0.6 * ur.mol/ur.mol

wBen = mwBen*xBen / (mwBen*xBen + mwTol*xTol)

print(wBen)
```

```
0.36108542899408286 dimensionless
```

Prefixes

In [ ]:

```
mwBen = 78.11 * ur.kg/ur.kmol
print(mwBen)

mwBen.ito_base_units()
print(mwBen)
```

```
78.11 kilogram / kilomole
0.07811 kilogram / mole
```

Temperature and Other Offset (Non-multiplicative) Units

Examples of non-multiplicative units are degrees Celsius, degrees Fahrenheit, and gauge pressure. For these units the zero of the measurement scale is offset from absolute zero of the underlying physical quantity. In these cases the units are assigned using the `Quantity()` function of the `UnitRegistry` module.

In [ ]:

```
T = ur.Quantity(25,ur.degC)

print(T, " = ", T.to(ur.degF))
```

```
25 degC  =  77.00000039999996 degF
```

There is an embedded ambiguity in working with non-multiplicative units. For example, given temperatures Ta = 10 and Tdelta = 25 in degrees C, should Ta + Tdelta give an answer of 35 degrees C, or the sum of the absolute temperatures which would correspond to 308.15 degrees C?

Pint uses unit type called `delta_degC` for this situation.

In [ ]:

```
Ta = ur.Quantity(10.0, ur.degC)
Tdelta = 25.0 * ur.delta_degC

print(Ta + Tdelta)
```

```
35.0 degC
```

In [ ]:

```
Tb = ur.Quantity(100.0, ur.degC)
Tf = ur.Quantity(32.0, ur.degF)

print("{0:6.3f}".format(Tb-Tf))
print("{0:6.3f}".format((Tb-Tf).to(ur.delta_degF)))
```

```
100.000 delta_degC
180.000 delta_degF
```

In [ ]:

```
P1 = (12.5 + 14.696) * ur.psi

T1 = ur.Quantity(70, ur.degF).to(ur.degR)
T2 = ur.Quantity(40, ur.degF).to(ur.degR)

P2 = (T2/T1)*P1 - 14.696 * ur.psi
P2
```

Out[ ]:

10.959644683262889 pound_force_per_square_inch

In [ ]:

```
import math
pi = math.pi

# constants
R = 8.314 * ur.joule/(ur.mol*ur.degK)
mw = 28.966 * ur.grams/ur.mol

# problem data
P = 1.0 * ur.atm
T = ur.Quantity(25,ur.degC).to(ur.degK)
r = 10 * ur.cm

# calculations
V = (4.0/3.0)*pi*r**3
print("Volume = ", V)

m = mw*P*V/(R*T)

print("Mass of Air = ", m.to(ur.grams))
```

```
Volume =  4188.790204786391 centimeter ** 3
Mass of Air =  4.95962584191983 gram
```

Exercises

Units of Refrigeration

A hockey arena is equipped with a cooling system providing 200 tons of refrigeration. (Refrigeration capacity is measured in units of 'tons' that correspond to the rate of energy removal necessary to freeze one ton of ice in 24 hour. It is commonly assumed to be 12,000 British Thermal Units (BTU) per hour). It is proposed to add seating for 1,500 spectators. Assuming each spectator releases about 100 watts of power, how many additional tons of refrigeration will be required to maintain ice conditions?

Units of the Ideal Gas Law

The ideal gas equation may be written as

$$PV = nRT$$

where P is pressure, V is volume, n is the number of moles, and T is temperature. Show that the units of PV can be expressed as energy.

Deflategate

NFL rules specify that footballs should inflated to between 12.5 and 13.5 pounds per square inch. In the so-called "Deflategate" controversy involving the AFC championship game of 2015, the New England Patriots were accused of underinflating the game balls in order to achieve an advantage. Independent estimates show the game balls were inflated to 11.5 pounds per square inch at the game day temperature of 51 degrees F. Suppose the balls were originally filled in a training room at 72 degrees F. Assuming the ideal gas law applies, did the balls meet NFL rules at the time they were filled?

Agricultural Chemical Calculations

A farmer plans to plant 1 square mile of farmland as corn and needs to calculate how much nitrogen fertilizer to buy. Lab samples show the upper 6 inches of soil average 8 parts per million (ppm) of nitrogen by weight, and 4 ppm in the next 18 inches. The dry weight of the soil is 4 million pounds per acre-foot. The farmer is planning for a yield of 180 bushels/acre, and the local university agricultural department recommends a total of 1.2 pounds of nitrogen per bushel for the local soil conditions.

Nitrogen is available to the farm in the form of urea (chemical formula CH_4N_2O) at a price of $592 per short ton, and as anhydrous ammonia (chemical formula NH_3) at a price of $847 per short ton.

What type of fertilizer should the farmer buy, how much, and at what cost?

Units of Illumination

A regional entreprenuer is proposing to develop an indoor hydroponics facility enclosing 1 acre of useable growing area. The proposal includes illuminating the growing area at 20,000 lux using led lamps capable of producing 80 lumens/watt and operating 18 hours/day. Local electricity prices are 8.5 cents per kilowatt-hour. What will be the annual cost of electricity for illumination?

Estimation

Atmospheric pressure averages about 14.696 pounds per square inch which is equal to the weight of the corresponding air column at sea level. The current level of carbon dioxide (CO_2) in the atmosphere is about 400 ppm (by mass), and rising at a level of 2.1 ppm per year. How much CO_2 would have to be sequestered each year to maintain carbon dioxide at current atmospheric levels? Express your answer in giga-metric-tons, and giga-metric-ton-moles.

If the density of air-dried wood is 0.53 grams/cm^3, is 20% water by weight, and the dry mass is 45% carbon, how many acre-feet of wood would be required to sequester the excess CO_2? Does that seem plausible?

In [ ]:

1.2 Units and Conversions for Home Heating

Summary

This notebook demonstrates unit conversions for several typical calculations of energy consumption.

Examples

Heating with Electricity

A typical energy efficient home in the midwest requires 50 million BTUs to heat the home for the winter. If the price of electricity is $0.08 USD per kilowatt hour, what would be the cost to heat a typical home with electricity?

Solution

In [1]:

```
Q_btu = 50e6                # BTU

price = 0.08                # USD per kwh
btu_per_joule = 9.486e-4    # conversion factor
kwh_per_joule = 2.778e-7    # conversion factor

cost = price*kwh_per_joule*Q_btu/btu_per_joule

print("Winter heating cost =", round(cost,2), "USD")
```

```
Winter heating cost = 1171.41 USD
```

Heating with Natural Gas

Natural gas is transported by pipeline from producing areas of the country to the midwest where it stored prior to distribution. For heating purposes, the energy content of natural gas is 1000 BTU per cubic foot at 1 atm and 15 $^{\circ}$C.

If the natural gas is stored at 1000 psia and 15 $^{\circ}$C, how large a storage tank is required to store natural gas for the winter? Report your answer in cubic meters.

Solution

The volume of natural gas required is determined by the heating requirement.

In [ ]:

```
V_ft3 = Q_btu/1000     # ft^3
```

Next compute the amount of natural gas required in lb moles

In [ ]:

```
R = 10.73           # ft^3 psia/(lbmol R)
T_degC = 15         # deg C

# convert temperature to absolute
T_degR = 9.0*T_degC/5.0 + 491.67   # deg R

# compute lb moles
n_lbmol = 14.696*V_ft3/(R*T_degR)

print(round(T_degR, 1), "degrees Rankine")
print(round(n_lbmol, 1), "lb-mols")
```

```
518.7 degrees Rankine
132.0 lb-mols
```

Now calculate the volume in cubic meters at the storage conditions.

In [ ]:

```
V_ft3 = n_lbmol*R*T_degR/1000.0

m3_per_ft3 = 0.028317

V_m3 = V_ft3*m3_per_ft3

print("Storage volume of natural gas at 1000 psia and 15 deg C =", round(V_m3,1),
"cubic meters")
```

```
Storage volume of natural gas at 1000 psia and 15 deg C = 20.8 cubic meters
```

Sizing a Propane Storage Tank

Liquid propane has a heating value of 46.3 MJ/kg, and a specific gravity of 0.493 at ambient temperatures. How large a storage tank will be required, in cubic meters?

Solution

In [ ]:

```
btu_per_joule = 9.486e-4

Q_joule = Q_btu/btu_per_joule
print("Heat requirement =", round(Q_joule/1e6,1), "Megajoules")

m_kg = Q_joule/46.3e6
print("Mass of propane required = {0:.2f} kg".format(m_kg))

V_m3 = m_kg/0.493/1000.0
print("Volume of propane required = {0:.2f} cubic meters".format(V_m3))
```

```
Heat requirement = 52709.3 Megajoules
Mass of propane required = 1138.43 kg
Volume of propane required = 2.31 cubic meters
```

In [ ]:

```

```

Chapter 2. Stoichiometry

2.1 Balancing Reactions

Summary

This Jupyter notebook demonstrates the balancing of chemical reactions using principles of reaction stoichiometry. After completing this notebook, you should know

- the definition of stoichiometric coefficient,
- a reaction is *balanced* when the number of atomic species and electric charge are conserved in the reaction,
- how to solve for stoichiometric coefficients by hand,
- how to solve for stoichiometric coefficients as the solution to a system of linear equations.

Stoichiometry

Stoichiometry is the quantitative analysis of chemical reactions. Stoichiometry provides a set of analytical tools essential to the design of chemical processes. One of the basic calculations of stoichiometry is to balance a chemical reaction.

The following screencast reviews what you probably learned in the past as a method for balancing reactions.

In [1]:

```
from IPython.display import YouTubeVideo
YouTubeVideo("e8cUyGBt8e8",560,315,start=0,end=108,rel=0)
```

Out[1]:

In process and bioengineering applications, however, we often deal with large and complex networks of reactions. For these applications it is important to have a more structured approach that is amenable to computations.

Stoichiometric Coefficients

A chemical reaction is commonly expressed as a formula of the type

$$a\,A + b\,B \longrightarrow c\,C + d\,D$$

where A, B, C, and D are the molecular species involved in the reaction, and the coefficients a, b, c, and d indicate the molar quantities of each species on each sides of the reaction. In this convention, the coefficients a, b, c, and d are all positive numbers. The reactants are on the left side of the reaction, the products are on the right.

A *balanced reaction* shows the molecular identity of compounds involved in the reactio. The stoichiometric coefficients

1. show the molar quantity of the compounds involved in the reaction,
2. conserve the total number of each type of atom between reactants and products,
3. conserve total electrical charge, and
4. conserve mass.

The last property, the conservation of mass in a balanced chemical reaction, is a consequence of the conservation of atomic species.

An alternative convention for expressing a balanced reaction is

$$\nu_A\,A + \nu_B\,B + \nu_C\,C + \nu_D\,D = 0$$

where the stoichiometric coefficients ν_A, ν_B, ν_C, and ν_D may take on positive or negative values. Negative coeifficients correspond to reactants, positive coefficients to products. The advantage of this notation will be become clear below where show how to setup algebraic equations correponding to the conditions for a balanced reaction.

Examples

Combustion of Methane

The unbalanced reaction for the combustion of methane (CH_4) is

$$CH_4 + O_2 \rightarrow CO_2 + H_2O$$

What we seek are a set of *stoichiometric coefficients* for the balanced reaction. The stoichiometric coefficients are numbers ν_s associated with each species s such that the chemical expression can be written as a formula

$$\nu_{CH_4}CH_4 + \nu_{O_2}O_2 + \nu_{CO_2}CO_2 + \nu_{H_2O}H_2O = 0$$

Negative stoichiometric coefficients correspond to reactants, positive stoichiometric coefficeints correspond to products of the reaction. The stoichiometric coefficients must conserve the number of atoms of each element appearing in the reaction, and electrical charge if the reaction involves charged species.

One means of computing the stoichiometric coefficients is to use the SymPy package for symbolic calculations to set up and solve the atom and charge balances corresponding to a chemical reaction.

For the computation, The first step is to import the sympy package into the current workspace. The next step is to introduce a stoichiometric coefficient for each chemical species participating in the reaction, and a list of atom balances expressed in terms of the stoichiometric coefficients. There is one atom balance for each atomic species appearing in the reaction. Each atom balance consists of an expression that will be zero for a balanced reaction.

In [2]:

```
import sympy

sympy.var(['vCH4', 'vO2', 'vCO2', 'vH2O'])

atomBalances = [
    sympy.Eq(vCH4 + vCO2,0),              # Carbon
    sympy.Eq(4*vCH4 + 2*vH2O,0),          # Hydrogen
    sympy.Eq(2*vO2 + +2*vCO2 + vH2O,0)    # Oxygen
]

for eqn in atomBalances:
    print(eqn)
```

```
Eq(vCH4 + vCO2, 0)
Eq(4*vCH4 + 2*vH2O, 0)
Eq(2*vCO2 + vH2O + 2*vO2, 0)
```

A unique solution is obtained by specifying a *basis* for the reaction. The basis is an additional equation that determines the stoichiometric coefficient for a particular chemical species. In this case, since this example refers to the combustion of methane, an obvious basis is to set the stoichiometric coefficient of methane to -1.

In [3]:

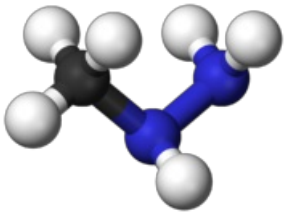
```
basis = [sympy.Eq(vCH4,-1)]

for eqn in atomBalances + basis:
    print(eqn)

sympy.solve(atomBalances + basis)
```

```
Eq(vCH4 + vCO2, 0)
Eq(4*vCH4 + 2*vH2O, 0)
Eq(2*vCO2 + vH2O + 2*vO2, 0)
Eq(vCH4, -1)
```

Out[3]:

```
{vH2O: 2, vCO2: 1, vCH4: -1, vO2: -2}
```

Hypergolic Reaction of Monomethylhydrazine and Nitrogen Tetraoxide



Source: Wikimedia Commons

Hypergolic reactions are reactions where the reactants spontaneously ignite. They are frequently used in space propulsion where it desired to have a thruster that can be readily controlled over a range of operating conditions.

One example is the reaction of the fuel monomethylhydrazine (MMH) fuel with the oxidizer nitrogen tetraoxide that is used in the SuperDraco engine developed by SpaceX, and in the reaction control system and orbital maneuvering systems of the Space Shuttle.

The **unbalanced** reaction is given by

$$CH_6N_2 + N_2O_4 \rightarrow CO_2 + NO + H_2O$$

Determine the stoichiometric coefficients for a balanced reaction. How much oxidizer is required per kilogram of fuel?

In [4]:

```
from IPython.display import YouTubeVideo
YouTubeVideo("VP_kGlmOH9U",560,315,start=0,end=108,rel=0)
```

Out[4]:

In [5]:

```
# Identify unknown stoichiometric coefficients
sympy.var(['vCH6N2','vN2O4','vCO2','vNO','vH2O'])

# Atom balances
eqns = [
    sympy.Eq(1*vCH6N2 + vCO2,0),                # Carbon
    sympy.Eq(6*vCH6N2 + 2*vH2O,0),              # Hydrogen
    sympy.Eq(4*vN2O4 + 2*vCO2 + vNO + vH2O,0),  # Oxygen
    sympy.Eq(2*vN2O4 + vNO,0)                   # Nitrogen
]

# Basis
eqns.append(sympy.Eq(vCH6N2,-1))

for eqn in eqns:
    print(eqn)

soln = sympy.solve(eqns)
print("\nStoichiometric Coefficients: ", soln)

mwN2O4 = 92.011
mwCH6N2 = 46.07

nCH6N2 = 1.0/mwCH6N2                          # kg-mol of MMH
nN2O4 =  (soln[vN2O4]/soln[vCH6N2])*nCH6N2    # kg-mol of N2O4
mN2O4 = mwN2O4*nN2O4                          # kg of N2O4

print("\n{0:7.3f} kilograms of N2O4 required per kilogram of CH6N2".format(mN2O4))
```

```
Eq(vCH6N2 + vCO2, 0)
Eq(6*vCH6N2 + 2*vH2O, 0)
Eq(2*vCO2 + vH2O + 4*vN2O4 + vNO, 0)
Eq(2*vN2O4 + vNO, 0)
Eq(vCH6N2, -1)

Stoichiometric Coefficients:  {vH2O: 3, vN2O4: -5/2, vCO2: 1, vCH6N2: -1, vNO: 5}

  4.993 kilograms of N2O4 required per kilogram of CH6N2
```

Reaction with Charge in Aqueous Solution

Metallic cadmium (Cd) will dissolve in concentrated solution of nitric acid to form the ion Cd^{+2} with the production of the free radical nitric oxide (NO). Write a balanced reaction.

The statement of the problem suggests the unbalanced reaction

$$Cd + NO_3^- \longrightarrow Cd^{+2} + NO$$

but clearly this can't be balanced because there are no offsetting negative charges on the right hand side of the equation. In problems like this need to incorporate other ions and water into the reaction. For this case, there are a total of eight molecular species to consider are

$$Cd,\ H_2O,\ HNO_3,\ NO,\ Cd^{+2},\ H^+,\ OH^-,\ NO_3^-$$

Next we write down the atom balances (4), charge balance (1), and basis specification (1) to create a list of six equations in the eight stoichiometic coefficients.

In [6]:

```
# Identify unknown stoichiometric coefficients
v = sympy.var(['vCd','vH2O','vHNO3','vNO','vCdpos2','vHpos','vOHneg','vNO3neg'])

# Atom balances
eqns = [
    sympy.Eq(vCd + vCdpos2, 0),                              # Cadmium
    sympy.Eq(2*vH2O + vHNO3 + vHpos + vOHneg, 0),            # Hydrogen
    sympy.Eq(vH2O + 3*vHNO3 + vNO + vOHneg + 3*vNO3neg, 0), # Oxygen
    sympy.Eq(vHNO3  + vNO + vNO3neg, 0)                      # Nitrogen
]

# Charge balance
eqns.append(sympy.Eq(2*vCdpos2 + vHpos - vOHneg - vNO3neg, 0))

# Basis
eqns.append(sympy.Eq(vCd, -1))

for eqn in eqns:
    print(eqn)
```

```
Eq(vCd + vCdpos2, 0)
Eq(2*vH2O + vHNO3 + vHpos + vOHneg, 0)
Eq(vH2O + 3*vHNO3 + vNO + 3*vNO3neg + vOHneg, 0)
Eq(vHNO3 + vNO + vNO3neg, 0)
Eq(2*vCdpos2 + vHpos - vNO3neg - vOHneg, 0)
Eq(vCd, -1)
```

This problems has more variables than equations, so there some *degrees of freedom*. We need to make some additional assumptions about this reaction to resolve those degrees of freedom. The first assumption is that the nitric acid is completely dissociated in solution so that $\nu_{HNO_3} = 0$.

In [7]:

```
eqns.append(sympy.Eq(vHNO3, 0))
sympy.solve(eqns)
```

Out[7]:

```
{vHpos: vOHneg - 8/3,
 vCd: -1,
 vCdpos2: 1,
 vH2O: -vOHneg + 4/3,
 vHNO3: 0,
 vNO3neg: -2/3,
 vNO: 2/3}
```

This partial solution shows that remaining six stoichiometric coefficients can be written in terms of ν_{OH^-}. What's happening is that there is a second reaction, the dissociation of water

$$H_2O \longrightarrow H^+ + OH^-$$

that is taking place among the same list of reactants. In order to separate the two reactions, we set $\nu_{H_2O} = 0$

In [8]:

```
sympy.solve(eqns + [sympy.Eq(vH2O,0)])
```

Out[8]:

```
{vOHneg: 4/3,
 vHpos: -4/3,
 vCdpos2: 1,
 vH2O: 0,
 vHNO3: 0,
 vCd: -1,
 vNO3neg: -2/3,
 vNO: 2/3}
```

which gives the resulting balanced equation

$$Cd + \frac{4}{3}H^+ + \frac{2}{3}NO_3^- \longrightarrow Cd^{+2} + \frac{4}{3}OH^- + \frac{2}{3}NO$$

Exercises

Combustion of Octane

Extend this example to balance the reaction for the combustion of octane C_8H_{18}.

Hard to Balance Reaction

Balance the following reaction:

$$HIO_3 + FeI_2 + HCl \longrightarrow FeCl_3 + ICl + H_2O$$

Another Hard to Balance Reaction

Balance the following reaction:

$$K_4[Fe(SCN)_6] + K_2Cr_2O_7 + H_2SO_4 \longrightarrow Fe_2(SO_4)_3 + Cr_2(SO_4)_3 + CO_2 + H_2O + K_2SO_4 + KNO_3$$

Reactions with Charge

Given the unbalanced equation

$$MnO_4^- + I^- \longrightarrow Mn^{+2} + I_2$$

find the balanced reaction assuming it takes place in an acidic, aqueous solution.

2.2 Generation Consumption Analysis

Summary

This Jupyter notebook demonstrate the use of the symbolic algebra package `Sympy` for the generation/consumption analysis for the production of ammonia using basic principles of reaction stoichiometry.

Example: Ammonia Production

BASF, headquartered in Ludwigshafen, Germany, is the largest chemical company in the world. In 1913, under its original name Badische Anilin- und Soda-Fabrik, BASF commercialized the Haber-Bosch process for the production of ammonia from natural gas, water, and air.

Prior to this invention, American and European agriculture was dependent on guano mined from the 'Guano Islands' in the Caribbean Sea and Pacific Ocean, and saltpeter mined from the deserts of Peru, Chile, and Bolivia. The competition for these limited resouces led to the notorious U.S. Guano Islands Act of 1856, and multiple wars (the Guano War, the War of the Pacific, later resulting in acute fertilizer shortages that was called 'the Wheat Problem' in England by Sir William Crookes in 1898.

The following video produced by BASF provides a technical overview of the Haber-Bosch process.

In [1]:

```
from IPython.display import YouTubeVideo
YouTubeVideo("uMkzxV_y7tY",560,315,rel=0)
```

Out[1]:

Problem Statement

Consider three reactions for the production of ammonia

1. Steam-reforming of methane

   $CH_4 + H_2O \rightarrow CO + 3\ H_2$

2. Water-gas shift

   $CO + H_2O \rightarrow CO_2 + H_2$

3. Haber-Bosch reaction

   $N_2 + 3\ H_2 \rightarrow 2\ NH_3$

Determine if it is possible to construct a process for the production of ammonia with no wasted hydrogen and no release of carbon monoxide.

Solution

We begin by setting up the stoichiometric matrix for generation/consumption analysis

Species	R_1	R_2	R_3	Net
	χ_1	χ_2	χ_3	$\sum_k \nu_k \chi_k$
CH_4	-1	0	0	≤ 0
H_2O	-1	-1	0	≤ 0
CO	1	-1	0	0
H_2	3	1	-3	0
CO_2	0	1	0	≥ 0
N_2	0	0	1	≤ 0
NH_3	0	0	2	1

which includes three equality constraints which need to be solved for χ_1, χ_2, and χ_3.

The first step is to import `sympy`.

In [2]:

```
import sympy
```

When imported in this way, the functions from `sympy` must be accessed with the prefix `sympy.`. This avoids overwriting functions with the same name as those in `sympy`, such as `plot`.

Next we use the `sympy.var` function to create three symbolic variables corresponding to χ_1, χ_2, and χ_3.

In [3]:

```
sympy.var('x1 x2 x3')
```

Out[3]:

```
(x1, x2, x3)
```

The net stoichiometric coefficients can be written in terms of the symbolic variables.

In [4]:

```
v = dict()
v['CH4'] = -x1
v['H2O'] = -x1 - x2
v['CO']  = x1 - x2
v['H2']  = 3*x1 + x2 - 3*x3
v['CO2'] = x2
v['N2'] = -x3
v['NH3'] = 2*x3
```

The three process constraints are encoded as equations using the sympy function `Eq()`

In [5]:

```
eqns = [
    sympy.Eq(v['NH3'],1),
    sympy.Eq(v['CO'],0),
    sympy.Eq(v['H2'],0)
]
```

These equations are solved for χ_1, χ_2, and χ_3.

In [6]:

```
soln = sympy.solve(eqns)
print(soln)
```

```
{x2: 3/8, x3: 1/2, x1: 3/8}
```

To finish the problem, the solutions are substituted back into the expressions for the stoichiometric coefficients, and the non-zero coefficients are displayed.

In [7]:

```
for k in v.keys():
    a = v[k].subs(soln)
    if a != 0:
        print("{0:<3s} {1:>6s}".format(k,str(a)))
```

```
N2    -1/2
H2O   -3/4
NH3     1
CH4   -3/8
CO2    3/8
```

In [ ]:

```

```

Chapter 3. Process Flows and Balances

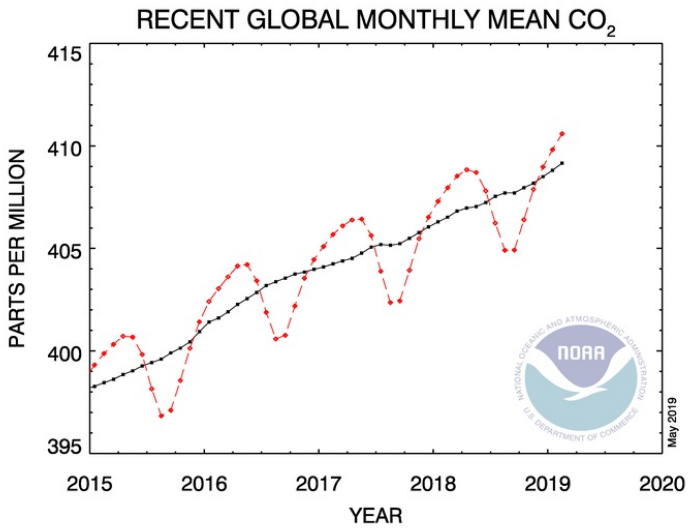
3.1 Global CO_2 Budget

Summary

This Jupyter notebook provides an analysis of global CO_2 emissions go by solving a mass balance.

Problem Statement

The total global emissons of CO_2 is estimated by the Netherlands Environmental Assessment Agency to be 34.5 billion metric tons from all sources, including fossil fuels, cement production, and land use changes. As measured by NOAA, in recent years the atmospheric concentration of CO_2 is increasing at an annual rate of about 2.4 ppmv (parts per million by volume).

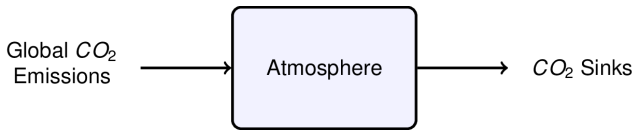


Assuming these numbers are accurate and that the atmosphere is well mixed, what fraction of global CO_2 emissions are being retained in the atmosphere?

Solution

This problem requires us to perform a material balance for CO_2 in the atmosphere. We'll perform the using a 10 step approach outlined in the textbook.

Step 1. Draw a diagram.



Step 2. Define the system of interest.

The system of interest is the global atmosphere which we assume is well mixed and of uniform composition.

Step 3. Choose components and define stream variables.

The chemical component to model is CO_2. The stream variables are the mass flowrates of CO2</sub> into and out of the atmosphere.

Step 4. Convert all units to consistent units of mass or moles.

This problem could be done on either a molar or mass basis. We'll arbitrarily choose to do this in mass units of kg/year. First we convert global emissions to kg/year.

In [ ]:

```
%matplotlib inline
from numpy import *
```

In [ ]:

```
mCO2_in = 34.5e9            # inflow, metric tonnes per year
mCO2_in = mCO2_in*1000      # inflow, kg per year
print("Global CO2 emissions = {:8.3g} kg/yr".format(mCO2_in))
```

```
Global CO2 emissions = 3.45e+13 kg/yr
```

The rate of accumulation is given as ppm by volume per year. For ideal gases, volume fraction is equivalent to mole fraction. We need to convert the mole fraction, which is the ratio of kg-moles of CO_2 to kg-moles of air, to mass fraction which has units of kg of CO_2 to kg of air per year.

In [ ]:

```
nCO2_accum = 2.4e-6                  # accumulation, kg-mol CO2/kg-mol air/yr

mwAir = 28.97                        # kg air/kg-mol air
mwCO2 = 44.01                        # kg CO2/kg-mol CO2

wCO2_accum = nCO2_accum*mwCO2/mwAir  # kg CO2/kg air/yr

print("Accumulation Rate of CO2 = {:8.3g} kg CO2/kg air".format(wCO2_accum))
```

```
Accumulation Rate of CO2 = 3.65e-06 kg CO2/kg air
```

Step 5. Define a basis.

The basis for the calculation are flows and change in one year. No additional work is required.

Step 6. Define system variables.

The system variable is the rate of accumulation of CO_2 in kg CO2/year. We need to convert from change in concentration per year to change in total mass per year. The first step is to estimate the total mass of air.

In [ ]:

```
# Earth Radius in meters
R = 6371000       # m

# Earth Area in square meters
A = 4*pi*R**2     # m**2

# Mass of the atmosphere in kg
g = 9.81          # N/kg
P = 101325        # N/m**2
mAir = A*P/g      # kg

print("Estimated mass of the atmosphere = {:8.3g} kg".format(mAir))
```

```
Estimated mass of the atmosphere = 5.27e+18 kg
```

To get the rate of change of total CO_2, multiply the total mass of by the rate of change of mass fraction of CO_2.

In [ ]:

```
mCO2_accum = wCO2_accum*mAir      # kg CO2/year

print("Change in CO2 = {:8.3g} kg CO2/year".format(mCO2_accum))
```

```
Change in CO2 = 1.92e+13 kg CO2/year
```

Step 7. List specifications

The inflow and rate of change of CO_2 are specified, and calculated above.

Step 8. Write material balance equations for each species.

$$\text{Accumulation} = \text{Inflow} - \text{Outflow} + \underbrace{\text{Generation}}_{=0} - \underbrace{\text{Consumption}}_{=0}$$

Step 9. Solve material balance equations.

$$\text{Outflow} = \text{Inflow} - \text{Accumulation}$$

In [ ]:

```
mCO2_out = mCO2_in - mCO2_accum

print("Global CO2 outflow = {:8.3g} kg CO2/yr".format(mCO2_out))
```

```
Global CO2 outflow = 1.53e+13 kg CO2/yr
```

Step 10. Check your work.

Fraction retained in the atmosphere

In [ ]:

```
fCO2 = mCO2_accum/mCO2_in
print("Fraction of CO2 retained in the atmosphere = {:<.2g} ".format(fCO2))
```

```
Fraction of CO2 retained in the atmosphere = 0.56
```

In [ ]:

```
```

3.2 CO_2 Production by Automobiles

Summary

This notebook demonstrates the solution of a mass balance for a vehicle powered by an internal combusion engine.

Examples

How much CO_2 is generated per mile driven with an ICE?

A recent model automobile is advertised with a fuel consumption of 30 miles per gallon of gasoline. Assume gasoline consists of pure octane C_8H_{18}, has a specific gravity of 0.74, and is consumed via the chemical reaction

$$C_8H_{18} + \frac{25}{2}\ O_2 \longrightarrow 8\ CO_2 + 9\ H_2O$$

How much CO_2 is generated per mile driven? Report your answer in grams/mile.

Solution

In [1]:

```
liters_per_m3 = 1000.0
gallons_per_m3 = 264.17

V_lpm = (1.0*liters_per_m3/gallons_per_m3)/30.0      # volume of gasline in liters/mile
m_kg = 0.74*V_lpm                                    # mass of gasoline in kg/mile
m_grams = m_kg*1000.0                                # mass of gasoline in grams/mile
n_octane = m_grams/114.0                             # moles of gasoline in gmol/mile
n_co2 = 8.0*n_octane                                 # modles of CO2 in gmol/mile
m_co2 = 44.0*n_co2                                   # mass of CO2 in grams/mile

print("Gasoline consumed per mile = ", round(m_grams,1), "g/mile")
print("Gram moles of octane per mile = ", round(n_octane,3) ,"gmol/mile")
print("CO2 Production =", round(m_co2,1), "g/mile")
```

```
Gasoline consumed per mile =  93.4 g/mile
Gram moles of octane per mile =  0.819 gmol/mile
CO2 Production = 288.3 g/mile
```

How much CO_2 is generated per mile driven by an electric car?

Owners of the Tesla S electric car report an average electricity consumption of 0.367 kilowatt-hours per mile driven.

Assume the electricity is produced from natural gas which, according to the U.S. Energy Information Administration, produces 117.0 pounds of CO_2 per million BTU consumed, and requires 10,400 BTU to produce a kilowatt-hour of electricity. Assume an overall transmission efficiency of 80% from the power plant to the Tesla motor. How many grams of CO_2 are generated per mile driven by the Tesla?

Solution

In [2]:

```
grams_per_lb = 453.593

w_kwh = 0.367                       # kwh per mile
q_btu = (w_kwh/0.8)*10400.0         # natural gas per mile

print("Thermal energy requirement =",round(q_btu,2),"BTU per mile driven")
```

```
Thermal energy requirement = 4771.0 BTU per mile driven
```

In [3]:

```
m_co2_lb = 117.0*q_btu/1.0e6            # mass CO2 lb/mile
m_co2_grams = m_co2_lb*grams_per_lb     # mass CO2 grams/mile

print("CO2 Production =", round(m_co2_grams,2), "grams per mile")
```

```
CO2 Production = 253.2 grams per mile
```

3.4 Separating Milk

Summary

This example describes a situation where the material balances and problem specifications do not result in a unique solution. Instead, there is at least one degree of freedom that must be resolved by further considerations. In this case, those considerations include the testing for physically meaningful answers, and to maximize a business objective. What you should learn from this example -

- Degrees of freedom may remain after all of the material balance and process specifications have are satisfied.
- The importance of testing for feasible solutions.
- How to use a degree of freedom to maximize a process objective.

Problem Statement

A dairy processor can separate raw milk into a number of commodity products, each with a price determined by the spot market. An example of commodity prices and product specifications is given in this table.

A commercial milk separator manufactured by Gruppo Pieralsi.

Component	Fat (wt%)	Non-fat Solids (wt%)	Price
Raw Milk	3.85	8.85	
Regular Milk	3.25	8.25	21.48 $/cwt
Skim Milk	0.10	8.25	16.01 $/cwt
Nonfat Dry Milk	0.0	100.0	1.95 $/lb
Butterfat	100	0.0	1.77 $/lb

Suppose a source can provide 5000 lb/hour of raw whole milk containing 3.85 wt% fat, and 8.85 wt% non-fat solids. The processor can produce a slate of products consisting of regular milk, skim milk, non-fat dry milk, and butterfat in whatever quantities are needed. Specify the product flows that maximize the revenue returned by this operation. What is the highest price the processor can pay for raw milk and still generate a positive cash flow?

(See The Daily Dairy Report for more background on the diary industry.)

Solution

In [1]:

```
%matplotlib inline
import sympy as sym
sym.init_printing()
```

Define Streams

The first step is to label the streams, and create mass flow variables corresponding to each stream.

In [2]:

```
sym.var(['M1','M2','M3','M4','M5'])
```

Out[2]:

$$[M_1, \quad M_2, \quad M_3, \quad M_4, \quad M_5]$$

In [3]:

```
prods = {};
prods[M1] = "Raw Milk"
prods[M2] = "Regular Milk Product"
prods[M3] = "Skim Milk Product"
prods[M4] = "Nonfat Dry Milk Product"
prods[M5] = "Butterfat Product"
prods
```

Out[3]:

```
{M1: 'Raw Milk',
 M2: 'Regular Milk Product',
 M3: 'Skim Milk Product',
 M4: 'Nonfat Dry Milk Product',
 M5: 'Butterfat Product'}
```

Material Balances

In [4]:

```
mbal = [
    sym.Eq(M1, M2 + M3 + M4 + M5),
    sym.Eq(0.0385*M1, 0.0325*M2 + 0.001*M3 + 0*M4 + 1.00*M5),
    sym.Eq(0.0885*M1, 0.0825*M2 + 0.0825*M3 + 1.00*M4 + 0*M5)
]

for m in mbal:
    print(m)
```

```
Eq(M1, M2 + M3 + M4 + M5)
Eq(0.0385*M1, 0.0325*M2 + 0.001*M3 + 1.0*M5)
Eq(0.0885*M1, 0.0825*M2 + 0.0825*M3 + 1.0*M4)
```

Specifications

In [5]:

```
specs = [
    sym.Eq(M1, 5000)
]
```

Finding Feasible Solutions to the Material Balances

In [6]:

```
soln = sym.solve(specs + mbal)

for m in soln.keys():
    print(m, ":", soln[m])
```

```
M4 : 0.0899182561307902*M5 + 32.6975476839237
M3 : 30.6215129103412*M5 - 986.116517451667
M2 : -31.711431166472*M5 + 5953.41896976774
M1 : 5000.00000000000
```

Let's repeat this step, this time specifying precisely which variables to solve for. The solutions will be found in terms of the remaining variables of the problem.

In [7]:

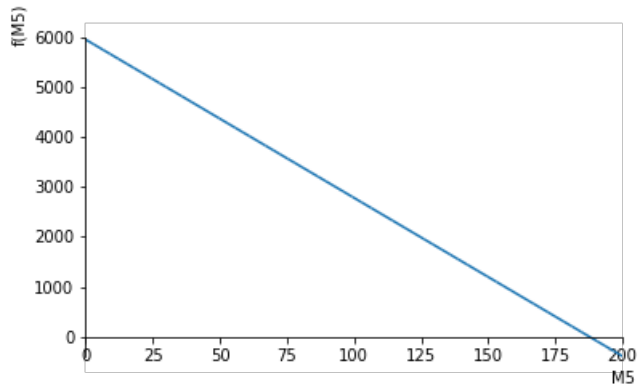
```
soln = sym.solve(specs + mbal,[M1,M2,M3,M4])

for m in soln.keys():
    print(m, ":", soln[m])
```

```
M1 : 5000.00000000000
M2 : -31.711431166472*M5 + 5953.41896976774
M3 : 30.6215129103412*M5 - 986.116517451667
M4 : 0.0899182561307902*M5 + 32.6975476839237
```

What are the feasible values of M_5?

In [8]:

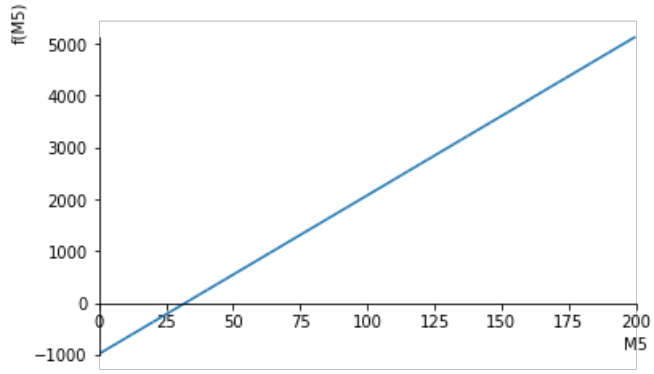
```
sym.plot(M2.subs(soln),(M5,0,200))
print(sym.solve(M2.subs(soln),M5))
```




```
[187.737315875614]
```

In [9]:

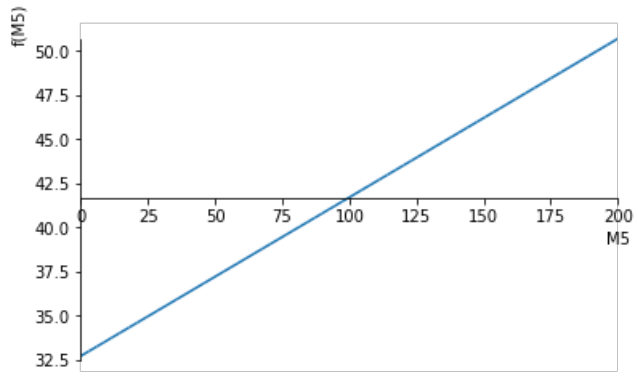
```
sym.plot(M3.subs(soln),(M5,0,200))
print(sym.solve(M3.subs(soln),M5))
```




```
[32.2033898305085]
```

In [10]:

```
sym.plot(M4.subs(soln),(M5,0,200))
print(sym.solve(M4.subs(soln),M5))
```




```
[-363.636363636364]
```

In [11]:

```
M5_max = sym.solve(M2.subs(soln))[0]
M5_min = sym.solve(M3.subs(soln))[0]
```

Finding a Solution to Maximize Revenue

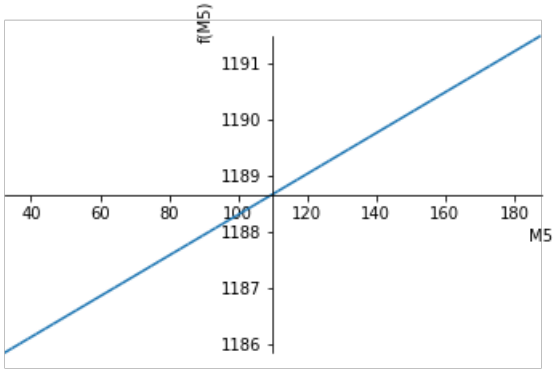
In [12]:

```
Revenue = 21.48*M2.subs(soln)/100.0 \
          + 16.01*M3.subs(soln)/100.0 \
          + 1.95*M4.subs(soln) \
          + 1.77*M5

print(Revenue)
```

```
0.0362294018424814*M5 + 1184.67735824575
```

In [13]:

```
sym.plot(Revenue,(M5,M5_min,M5_max))
```



Out[13]:

```
<sympy.plotting.plot.Plot at 0x1162a8668>
```

In [14]:

```
opt = [sym.Eq(M5_min,M5)]

soln = sym.solve(mbal + specs + opt)

for m in soln.keys():
    print("{0:2s} : {1:8.1f} lb/hr  {2}".format(str(m), soln[m], str(prods[m])))
```

```
M1 :   5000.0 lb/hr  Raw Milk
M2 :   4932.2 lb/hr  Regular Milk Product
M3 :      0.0 lb/hr  Skim Milk Product
M4 :     35.6 lb/hr  Nonfat Dry Milk Product
M5 :     32.2 lb/hr  Butterfat Product
```

Revenue and Milk Price Calculations

In [15]:

```
print("   Maximum Revenue = ${0:8.2f}/hr".format(Revenue.subs(soln)))
print("Max Raw Milk Price = ${0:8.2f}/lb".format(Revenue.subs(soln)/5000.0))
```

```
   Maximum Revenue = $ 1185.84/hr
Max Raw Milk Price = $    0.24/lb
```

In [ ]:

```

```

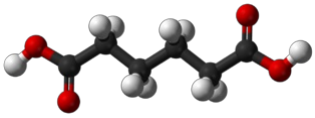
3.5 Adipic Acid Flowsheet

Summary

This Jupyter notebook demonstrates the formulation and solution of material balances for a hypothetical adipic acid process described by Murphy (2005, Example 2.15) using the symbolic algebra package Sympy.

Problem Statement

Adipic acid ($C_6H_{10}O_4$) (also called hexanedioic acid) rarely occurs in nature, but approximately 2.5 billion kilograms are produced per year from petrochemical feedstocks primarily for the production of nylon.



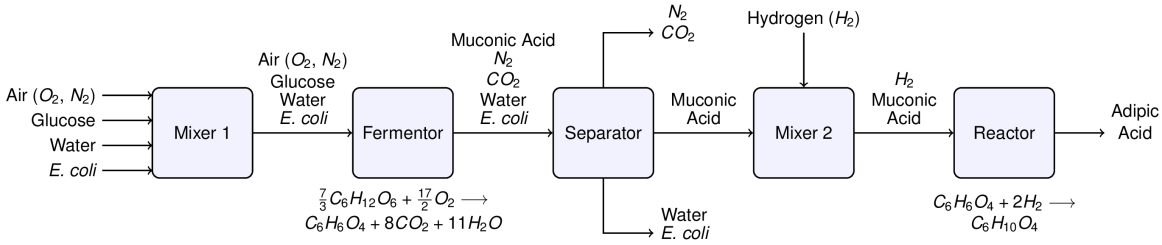
Recently there has been interest in producing adipic acid from renewable resources. For example, starting with glucose ($C_6H_{12}O_6$), muconic acid ($C_6H_6O_4$) is produced through fermentation with a genetically modified e. coli. via the reaction

$^{7}/_{3}\ C_6H_{12}O_6 + {}^{17}/_{2}\ O_2 \rightarrow C_6H_6O_4 + 8\ CO_2 + 11\ H_2O$

that is subsequently hydrogentated to form adipic acid

$C_6H_6O_4 + 2\ H_2 \rightarrow C_6H_{10}O_4$

Murphy (2005, Example 2.15) outlines a hypothetical flowsheet for this process:



Neglecting the *E. coli*, solve for the flowrates necessary to produce 12,000 kg/hour of adipic acid assuming that glucose is available as a 10 mg/mL solution.

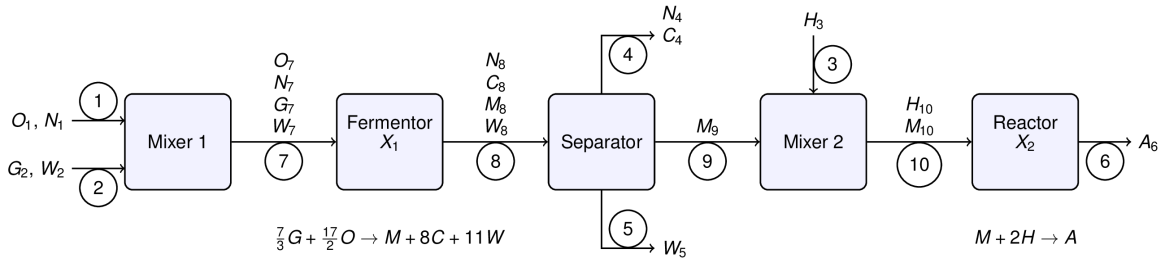
Solution

Process Variables

We begin by relabeling the process flowsheet with stream numbers, stream variables, and extents of reaction. There are chemical species are abbreviated as follows:

- A: adipic acid
- C: carbon dioxide
- G: glucose
- H: hydrogen
- M: muconic acid
- N: nitrogen
- O: oxygen
- W: water

and where X_1 and X_2 denote the extents of reactions 1 and 2, respectively. The stream variables denote chemical flowrates in units of kg/hour. The extents of reaction will be in units of kg-mol/hour.



In [1]:

```
# Import the symbolic algebra package sympy
import sympy as sym

# Extents of reactions 1 and 2
sym.var('X1 X2')

# Stream variables
sym.var('O1 N1')           # Stream 1
sym.var('G2 W2')           # Stream 2
sym.var('H3')              # Stream 3
sym.var('N4 C4')           # Stream 4
sym.var('W5')              # Stream 5
sym.var('A6')              # Stream 6
sym.var('O7 N7 G7 W7')     # Stream 7
sym.var('N8 W8 C8 M8')     # Stream 8
sym.var('M9')              # Stream 9
sym.var('H10 M10')         # Stream 10
```

Out[1]:

```
(H10, M10)
```

Because the flowsheet includes reactions, it will be necessary to include molecular weights in the mass balance expressions. For this purpose we gather the molecular weights of all species into a python dictionary indexed by the chemical species.

In [2]:

```
MW = {
    'A': 146.14,
    'C': 44.01,
    'G': 180.16,
    'H': 2.02,
    'M': 142.11,
    'N': 14.01,
    'O': 16.00,
    'W': 18.02
}
```

Specifications

In [3]:

```
specs = [
    sym.Eq(A6, 12000),
    sym.Eq(N1/MW['N'], (0.79/0.21)*(O1/MW['O'])),
    sym.Eq(G2, 0.01*W2)
]
```

Material Balances

In [4]:

```
mixer1 = [
    sym.Eq(0, O1 - O7),
    sym.Eq(0, N1 - N7),
    sym.Eq(0, G2 - G7),
    sym.Eq(0, W2 - W7)
]

reactor1 = [
    sym.Eq(0,  O7 - MW['O']*(17/2)*X1),
    sym.Eq(0,  N7 - N8),
    sym.Eq(0,  G7 - MW['G']*(7/3)*X1),
    sym.Eq(0, -C8 + MW['C']*8*X1),
    sym.Eq(0, -M8 + MW['M']*X1),
    sym.Eq(0,  W7 - W8 + MW['W']*11*X1)
]

separator = [
    sym.Eq(0, N8 - N4),
    sym.Eq(0, C8 - C4),
    sym.Eq(0, M8 - M9),
    sym.Eq(0, W8 - W5)
]

mixer2 = [
    sym.Eq(0, M9 - M10),
    sym.Eq(0, H3 - H10)
]

reactor2 = [
    sym.Eq(0, H10 - MW['H']*2*X2),
    sym.Eq(0, M10 - MW['M']*X2),
    sym.Eq(0, -A6 + MW['A']*X2)
]

mbals = mixer1 + reactor1 + separator + mixer2 + reactor2
```

In [5]:

```
specs + mbals
```

Out[5]:

```
[Eq(A6, 12000),
 Eq(0.0713775874375446*N1, 0.235119047619048*O1),
 Eq(G2, 0.01*W2),
 Eq(0, O1 - O7),
 Eq(0, N1 - N7),
 Eq(0, G2 - G7),
 Eq(0, W2 - W7),
 Eq(0, O7 - 136.0*X1),
 Eq(0, N7 - N8),
 Eq(0, G7 - 420.373333333333*X1),
 Eq(0, -C8 + 352.08*X1),
 Eq(0, -M8 + 142.11*X1),
 Eq(0, W7 - W8 + 198.22*X1),
 Eq(0, -N4 + N8),
 Eq(0, -C4 + C8),
 Eq(0, M8 - M9),
 Eq(0, -W5 + W8),
 Eq(0, -M10 + M9),
 Eq(0, -H10 + H3),
 Eq(0, H10 - 4.04*X2),
 Eq(0, M10 - 142.11*X2),
 Eq(0, -A6 + 146.14*X2)]
```

Solution

In [6]:

```
soln = sym.solve(mbals + specs)
soln
```

Out[6]:

```
{H3: 331.736690844396,
 C8: 28910.3599288354,
 W2: 3451813.32968386,
 W5: 3468089.77692623,
 G7: 34518.1332968386,
 N7: 36785.5285538330,
 O7: 11167.3737511975,
 W8: 3468089.77692623,
 W7: 3451813.32968386,
 N8: 36785.5285538330,
 G2: 34518.1332968386,
 N1: 36785.5285538330,
 M9: 11669.0844395785,
 C4: 28910.3599288354,
 X2: 82.1130422882168,
 M8: 11669.0844395785,
 M10: 11669.0844395785,
 A6: 12000.0000000000,
 H10: 331.736690844396,
 N4: 36785.5285538330,
 O1: 11167.3737511975,
 X1: 82.1130422882168}
```

In [7]:

```
for key in soln.keys():
    print("{:3s}    {:10.1f}".format(str(key), float(soln[key])))
```

```
H3           331.7
C8         28910.4
W2       3451813.3
W5       3468089.8
G7         34518.1
N7         36785.5
O7         11167.4
W8       3468089.8
W7       3451813.3
N8         36785.5
G2         34518.1
N1         36785.5
M9         11669.1
C4         28910.4
X2            82.1
M8         11669.1
M10        11669.1
A6         12000.0
H10          331.7
N4         36785.5
O1         11167.4
X1            82.1
```

In [ ]:

```
```

Chapter 4. Material Balances

4.1 Lean NO_x Trap

Summary

This Jupyter notebook demonstrates the calculation of mass balances for a lean NOx trap used to meet air quality standards for nitrogen oxides in the exhaust of diesel powered trucks and automobiles.

Problem Statement

Some diesel car and truck manufacturers, including Volkswagen and others, have introduced Lean NO_x Traps (LNT) in exhaust systems to meet stringent air quality standards for nitrogen oxides.

An LNT includes a small amount of platinum (Pt) in a barium oxide (BaO) wash coat spread over a ceramic monolith. Under under lean fuel conditions, NO is converted to NO_2 by the reaction

$$NO\,(g) + \frac{1}{2}O_2\,(g) \longrightarrow NO_2\,(g)$$

catalyzed by Pt. Nitrogen dioxide is then adsorbed into the trap by the reaction

$$BaO\,(s) + 2\,NO_2\,(g) + \frac{1}{2}\,O_2\,(g) \longrightarrow Ba(NO_3)_2\,(s)$$

The washcoat has to be regenerated once it has been saturated with nitrogen oxides. To regenerate, a small amount of diesel fuel to release the NO_x which reacts with the hydrocarbons to form water and nitrogen.

Consider an LNT operating with a total inflow of 1 kg/min with 850ppm NO, 150ppm NO_2, and O_2 in substantial excess.

Needed data is in the following table.

Molecular Species	MW
$Ba(NO_3)_2$	261.3
BaO	153.3
NO	30.0
NO_2	46.0
O_2	32.0

The exhaust gas is to be reduced to 50ppm NO with no residual NO_2.

1. Identify the relevant stream and system variables. Perform a degree of freedom analysis.
2. Calculate the extents of reaction. Be sure to show units.
3. The washcoat is initially loaded with 100 grams of BaO. How long is it before the trap must be regenerated?

Solution

Part a. Degree of Freedom Analysis

The relevant stream and system variables are labeled on the following diagram. The inlet specifications have been translated to flow rates. The outlet flow of NO_2 is zero, and the flow of $\dot{m}_{NO,2}$ is 50 ppm of the total outlet mass flow.

Variables	
Stream Variables	6
Accumulation Variables	2
Extents of Reaction	2
TOTAL VARIABLES	10

Equations	
Mass Balances	5
Inlet Specifications	2
Outlet Specification	2
TOTAL EQUATIONS	8

$$DOF = Variables - Equations = 10 - 9 = \boxed{1}$$

Part b. Material Balances

$$\begin{aligned}
\dot{r}_{Ba(NO_3)_2} &= M_{Ba(NO_3)_2}\dot{\xi}_2 \\
\dot{r}_{BaO} &= -M_{BaO}\dot{\xi}_2 \\
0 &= \dot{m}_{NO,1} - \dot{m}_{NO,2} - M_{NO}\dot{\xi}_1 \\
0 &= \dot{m}_{NO_2,1} - \dot{m}_{NO_2,2} + M_{NO_2}\dot{\xi}_1 - 2M_{NO_2}\dot{\xi}_2 \\
0 &= \dot{m}_{O_2,1} - \dot{m}_{O_2,2} - \frac{1}{2}M_{O_2}\dot{\xi}_1 - \frac{1}{2}M_{O_2}\dot{\xi}_2
\end{aligned}$$

The flow of stream 2 is essentially 1 kg/min because only a small part is reactive. The material balances for NO and NO_2 are

$$\begin{aligned}
0 &= \underbrace{\dot{m}_{NO,1}}_{0.85\ g/min} - \underbrace{\dot{m}_{NO,2}}_{0.05\ g/min} - \underbrace{M_{NO}}_{30.0}\dot{\xi}_1 \\
0 &= \underbrace{\dot{m}_{NO_2,1}}_{0.15\ g/min} - \underbrace{\dot{m}_{NO_2,2}}_{0\ g/min} + \underbrace{M_{NO_2}}_{46.0}\dot{\xi}_1 - 2\underbrace{M_{NO_2}}_{46.0}\dot{\xi}_2
\end{aligned}$$

Solving the first equation

$$\dot{\xi}_1 = \frac{\dot{m}_{NO,1} - \dot{m}_{NO,2}}{M_{NO}} = \frac{0.85\ g/min - 0.05\ g/min}{30.0\ g/gmol} = \boxed{0.0267\ \text{gmol/min}}$$

The solving for the second

$$\dot{\xi}_2 = \frac{0.15\ g/min + 46.0\ g/gmol \times 0.0267\ gmol/min}{2 \times 46.0 g/gmol} = \boxed{0.015\ \text{gmol/min}}$$

Part c. The rate of consumption of BaO

$$\dot{r}_{Ba(NO_3)_2} = M_{Ba(NO_3)_2}\dot{\xi}_2 = 153.3 \times 0.015\ gmol \qquad /$$

so the 100 grams of \ce{BaO} will need to be regenerated in

$$t = \frac{100\ g}{2.3\ g/min} = \boxed{43.6\ \text{min}}$$

In [ ]:

4.2 Ethylene Oxide Flowsheet

Summary

This Jupyter notebook demonstrates the formulation and solution of material balances for an ethylene oxide flowsheet in Python using the symbolic algebra package Sympy.

Introduction

While material balances for chemical processes are readily expressed as linear equations, extracting the matrix of coefficients for the linear equations can be tedious and error-prone. Fortunately, symbolic calculations can be used to solve material balance problems typical of introductory chemical engineering courses. This is demonstrated below using an example from the textbook *Introduction to Chemical Processes: Principles, Analysis, and Synthesis* by Regina Murphy.

Ethylene Oxide

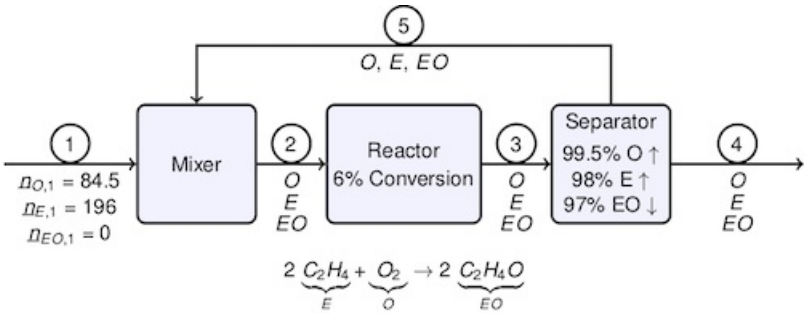
"TASNEE 001" by Secl - Own work. Licensed under CC BY 3.0 via Wikimedia Commons.

Problem Statement (Murphy 2005, Example 3.19)

The problem is to analyze the capability of an existing process for the production of ethylene oxide by the reaction of ethylene and oxygen

$$2\,C_2H_4 + O_2 \longrightarrow 2\,C_2H_4O$$

The target production is 1.7 × 10^6 kgmol/year of an ethylene oxide product with 98 mol% purity. The feedrate of ethylene is specified as 196 kgmol/hr, and of oxygen is specified as 84.5 kgmol/hr. The reactor has a nominal single pass conversion of 6% ethylene. The separator recovers 98% of the ethylene and 99.5% of the oxygen and 98% of the ethylene for recycle, and 97% of the ethylene oxide for the product stream.



The problem is to find the nominal product purity and production rates, and to examine the sensitivity of product purity and production to the equipment specifications.

Solution

Sympy is a library written in pure python for symbolic algebra. The solution strategy is to represent the stream variables and reaction extents as symbolic variables, then express the material balances and process specifications as symbolic equations, and finally to use the Sympy solver to find a nominal solution and to perform parametric analyses. The following cell initializes `sympy` and necessary printing and display functions.

In [1]:

```
%matplotlib inline

import sympy as sym
sym.init_printing()

from IPython.display import display
```

Variables

Stream Variables

The stream variables are systematically created using the Sympy `var` function, and added to the list `streams`.

In [2]:

```
stream_table = []
stream_table.append(sym.var('nE1 nO1 nEO1'))   # Stream 1
stream_table.append(sym.var('nE2 nO2 nEO2'))   # Stream 2
stream_table.append(sym.var('nE3 nO3 nEO3'))   # Stream 3
stream_table.append(sym.var('nE4 nO4 nEO4'))   # Stream 4
stream_table.append(sym.var('nE5 nO5 nEO5'))   # Stream 5

display(stream_table)
```

$$[(nE_1,\ nO_1,\ nEO_1),\ (nE_2,\ nO_2,\ nEO_2),\ (nE_3,\ nO_3,\ nEO_3),\ (nE_4,\ nO_4,\ nEO_4),$$

Extent of Reaction

This problem includes only a single reaction in a single reactor. A corresponding extent of reaction variable is created, and a master list of all process variables is created.

In [3]:

```
extents = [sym.var('X')]
display(extents)
```

$$[X]$$

Create list of all variables

In [4]:

```
variables = []
for x in extents:
    variables.append(x)
for s in stream_table:
    for v in s:
        variables.append(v)

display(variables)
```

$$[X,\ nE_1,\ nO_1,\ nEO_1,\ nE_2,\ nO_2,\ nEO_2,\ nE_3,\ nO_3,\ nEO_3,\ nE_4,\ nO_4,\ nEO_4,\ n$$

Equations

Material Balances

Material balances are written for each process unit using the symbolic stream and extent of reaction variables created above. Each material balance is expressed as the net rate of accumulation that will be set to zero to specify steady-state operation. The material balances are gathered into a list for each process unit, then the lists are concantenated to create a list of all material balances.

In [5]:

```
mixer = [
    sym.Eq(0, nE1 + nE5 - nE2),
    sym.Eq(0, nO1 + nO5 - nO2),
    sym.Eq(0, nEO5 - nEO2)]

reactor = [
    sym.Eq(0, nE2 - nE3 - 2*X),
    sym.Eq(0, nO2 - nO3 - X),
    sym.Eq(0, nEO2 - nEO3 + 2*X)]

separator = [
    sym.Eq(0, nE3 - nE4 - nE5),
    sym.Eq(0, nO3 - nO4 - nO5),
    sym.Eq(0, nEO3 - nEO4 - nEO5)]

material_balances = mixer + reactor + separator
for eqn in material_balances:
    display(eqn)
```

$$0 = nE_{1} - nE_{2} + nE_{5}$$
$$0 = nO_{1} - nO_{2} + nO_{5}$$
$$0 = -nEO_{2} + nEO_{5}$$
$$0 = -2X + nE_{2} - nE_{3}$$
$$0 = -X + nO_{2} - nO_{3}$$
$$0 = 2X + nEO_{2} - nEO_{3}$$
$$0 = nE_{3} - nE_{4} - nE_{5}$$
$$0 = nO_{3} - nO_{4} - nO_{5}$$
$$0 = nEO_{3} - nEO_{4} - nEO_{5}$$

Specifications

Process specifications are written as equalities using the Sympy `Eq` function.

In [6]:

```
feed_spec = [
    sym.Eq(nE1, 196.0),
    sym.Eq(nO1, 84.5),
    sym.Eq(nEO1, 0.0)]

reactor_spec = [
    sym.Eq(nE2 - nE3, 0.06*nE2)]

separator_spec = [
    sym.Eq(nE5, 0.98*nE3),
    sym.Eq(nO5, 0.995*nO3),
    sym.Eq(nEO4, 0.97*nEO3)]

specifications = feed_spec + reactor_spec + separator_spec
for eqn in specifications:
    display(eqn)
```

$$nE_{1} = 196.0$$

$$nO_{1} = 84.5$$

$$nEO_{1} = 0.0$$

$$nE_{2} - nE_{3} = 0.06 nE_{2}$$

$$nE_{5} = 0.98 nE_{3}$$

$$nO_{5} = 0.995 nO_{3}$$

$$nEO_{4} = 0.97 nEO_{3}$$

Degree of Freedom Analysis

A simple degree of freedom analysis is to compare the number of variables to the number of equations.

In [7]:

```
nVars = 0
for s in stream_table:
    for v in s:
        nVars += 1
        print("Stream: {0:2d}    Variable: {1:5s}".format(nVars,v.name))

print("\n%d Extents of Reaction\n" % len(extents))
for v in extents:
    print("Extent: ", v.name)

print("\n%d Variables = %d Stream Variables + %d Extents of Reaction \n" \
    % (len(variables),nVars,len(extents)))
```

```
Stream:  1    Variable: nE1
Stream:  2    Variable: nO1
Stream:  3    Variable: nEO1
Stream:  4    Variable: nE2
Stream:  5    Variable: nO2
Stream:  6    Variable: nEO2
Stream:  7    Variable: nE3
Stream:  8    Variable: nO3
Stream:  9    Variable: nEO3
Stream: 10    Variable: nE4
Stream: 11    Variable: nO4
Stream: 12    Variable: nEO4
Stream: 13    Variable: nE5
Stream: 14    Variable: nO5
Stream: 15    Variable: nEO5

1 Extents of Reaction

Extent:  X

16 Variables = 15 Stream Variables + 1 Extents of Reaction
```

In [8]:

```
equations = material_balances + specifications
print("\n%d Equations = %d Material Balances + %d Specifications" \
    % (len(equations),len(material_balances),len(specifications)))

print("\n%d Material Balances\n" % len(material_balances))
for mb in material_balances:
    print(mb)

print("\n%d Specifications\n" % len(specifications))
for spec in specifications:
    print(spec)
```

```
16 Equations = 9 Material Balances + 7 Specifications

9 Material Balances

Eq(0, nE1 - nE2 + nE5)
Eq(0, nO1 - nO2 + nO5)
Eq(0, -nEO2 + nEO5)
Eq(0, -2*X + nE2 - nE3)
Eq(0, -X + nO2 - nO3)
Eq(0, 2*X + nEO2 - nEO3)
Eq(0, nE3 - nE4 - nE5)
Eq(0, nO3 - nO4 - nO5)
Eq(0, nEO3 - nEO4 - nEO5)

7 Specifications

Eq(nE1, 196.000000000000)
Eq(nO1, 84.5000000000000)
Eq(nEO1, 0.0)
Eq(nE2 - nE3, 0.06*nE2)
Eq(nE5, 0.98*nE3)
Eq(nO5, 0.995*nO3)
Eq(nEO4, 0.97*nEO3)
```

Solution

In [9]:

```
soln = sym.solve(material_balances + specifications)

for k in soln.keys():
    print("Variable {0:4s}:  {1:8.2f}".format(str(k),round(soln[k],2)))
```

```
Variable nO3 :   1976.14
Variable nEO3:    153.85
Variable nEO1:      0.00
Variable nE3 :   2338.07
Variable X   :     74.62
Variable nO4 :      9.88
Variable nEO5:      4.62
Variable nO2 :   2050.76
Variable nE1 :    196.00
Variable nEO2:      4.62
Variable nEO4:    149.24
Variable nO5 :   1966.26
Variable nE2 :   2487.31
Variable nO1 :     84.50
Variable nE4 :     46.76
Variable nE5 :   2291.31
```

In [10]:

```
# display solution for each variable, rounded to 1 decimal place
for v in variables:
    display(sym.Eq(v,round(soln[v],1)))
```

$$X = 74.6$$
$$nE_1 = 196.0$$
$$nO_1 = 84.5$$
$$nEO_1 = 0.0$$
$$nE_2 = 2487.3$$
$$nO_2 = 2050.8$$
$$nEO_2 = 4.6$$
$$nE_3 = 2338.1$$
$$nO_3 = 1976.1$$
$$nEO_3 = 153.9$$
$$nE_4 = 46.8$$
$$nO_4 = 9.9$$
$$nEO_4 = 149.2$$
$$nE_5 = 2291.3$$
$$nO_5 = 1966.3$$
$$nEO_5 = 4.6$$

Production and Purity

In [11]:

```
purity = soln[nEO4]/(soln[nEO4]+soln[nE4]+soln[nO4])
production = 24*350*(soln[nEO4] + soln[nE4] + soln[nO4])/1000000

print("Annual Production %4.2f million kgmol/year at %5.3f purity." \
    % (production,purity))
```

```
Annual Production 1.73 million kgmol/year at 0.725 purity.
```

Discussion Questions

- Do these numbers surprise you? Why is the recycle rate so high?
- The purity specification is not met. Why?

Parametric Analysis: Fractional Conversion of Ethylene

The problem asked for an analysis of the sensitivity of the problem results to changes in unit performance. This is implemented by restating the specifications where a key parameter is replaced by a symbolic variable, and the process the performance plotted as a function of the parameter.

In [12]:

```
feed_spec = [
    sym.Eq(nE1, 196.0),
    sym.Eq(nO1, 84.5),
    sym.Eq(nEO1, 0.0)]

fconv = sym.var('fconv')
reactor_spec = [
    sym.Eq(nE2 - nE3, fconv*nE2)]

separator_spec = [
    sym.Eq(nE5, 0.98*nE3),
    sym.Eq(nO5, 0.995*nO3),
    sym.Eq(nEO4, 0.97*nEO3)]

specifications = feed_spec + reactor_spec + separator_spec
for s in specifications:
    display(s)
```

$$nE_1 = 196.0$$
$$nO_1 = 84.5$$
$$nEO_1 = 0.0$$
$$nE_2 - nE_3 = fconvnE_2$$
$$nE_5 = 0.98nE_3$$
$$nO_5 = 0.995nO_3$$
$$nEO_4 = 0.97nEO_3$$

Recycle calculations are introduce a strong dependence of flow rates on parameters such as fraction conversion in the reactor and fractional recovery in separation units. To see this, here we solve for the flowrate of E_2 as a function of fractional conversion of ethylene in the reactor.

From the material balances

$E_2 = E_1 + E_5$
$E_3 = (1\text{-}f_{conv})\ E_2$
$E_5 = 0.98\ E_3$

Take a moment and solve these by hand.

In [13]:

```
soln = sym.solve(material_balances + specifications, exclude=[fconv])
soln
```

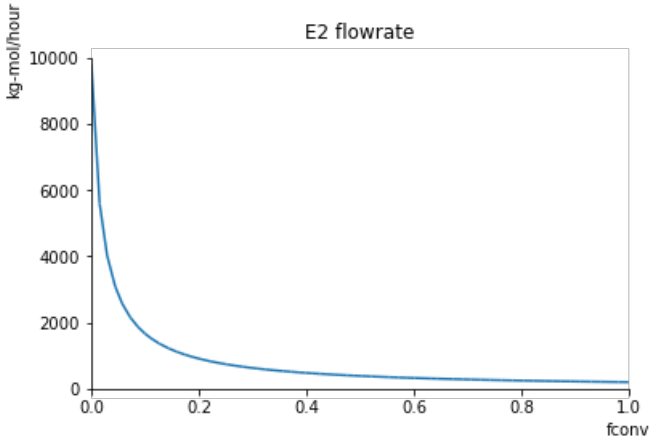
Out[13]:

$$\left\{X : \frac{4900.0fconv}{49.0fconv + 1.0}, \quad nE_1 : 196.0, \quad nE_2 : \frac{9800.0}{49.0fconv + 1.0}, \quad nE_3 : \frac{-9800.0fconv + 9800.0}{49.0fconv + 1.0}, \quad nE\right.$$

In [14]:

```
display(sym.Eq(nE2,soln[nE2]))
sym.plot(soln[nE2],(fconv,0,1),xlabel='fconv',ylabel='kg-mol/hour',title='E2 flowrate')
;
```

$$nE_2 = \frac{9800.0}{49.0 fconv + 1.0}$$



In [15]:

```
for v in variables:
    display(sym.Eq(v, soln[v]))
```

$$X = \frac{4900.0 fconv}{49.0 fconv + 1.0}$$

$$nE_{1} = 196.0$$

$$nO_{1} = 84.5$$

$$nEO_{1} = 0.0$$

$$nE_{2} = \frac{9800.0}{49.0 fconv + 1.0}$$

$$nO_{2} = \frac{-147000.0 fconv + 16900.0}{49.0 fconv + 1.0}$$

$$nEO_{2} = \frac{303.092783505155 fconv}{49.0 fconv + 1.0}$$

$$nE_{3} = \frac{-9800.0 fconv + 9800.0}{49.0 fconv + 1.0}$$

$$nO_{3} = \frac{-151900.0 fconv + 16900.0}{49.0 fconv + 1.0}$$

$$nEO_{3} = \frac{10103.0927835052 fconv}{49.0 fconv + 1.0}$$

$$nE_{4} = \frac{-196.0 fconv + 196.0}{49.0 fconv + 1.0}$$

$$nO_{4} = \frac{-759.5 fconv + 84.5}{49.0 fconv + 1.0}$$

$$nEO_{4} = \frac{9800.0 fconv}{49.0 fconv + 1.0}$$

$$nE_{5} = \frac{-9604.0 fconv + 9604.0}{49.0 fconv + 1.0}$$

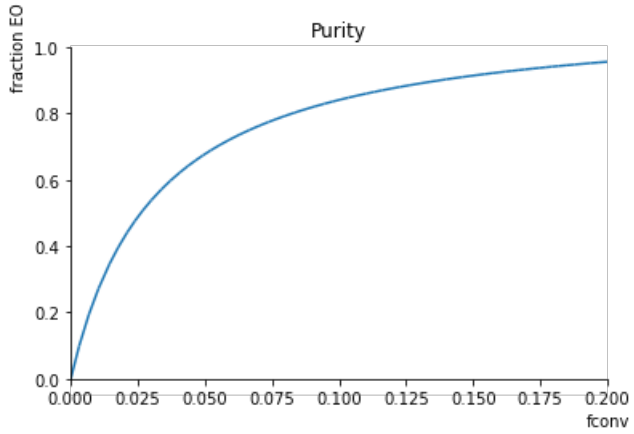
$$nO_{5} = \frac{-151140.5 fconv + 16815.5}{49.0 fconv + 1.0}$$

$$nEO_{5} = \frac{303.092783505155 fconv}{49.0 fconv + 1.0}$$

In [16]:

```
purity = soln[nEO4]/(soln[nEO4]+soln[nE4]+soln[nO4])

display(sym.simplify(purity))
sym.plot(purity,(fconv,0,.2),xlabel='fconv',ylabel='fraction EO',title='Purity');
```

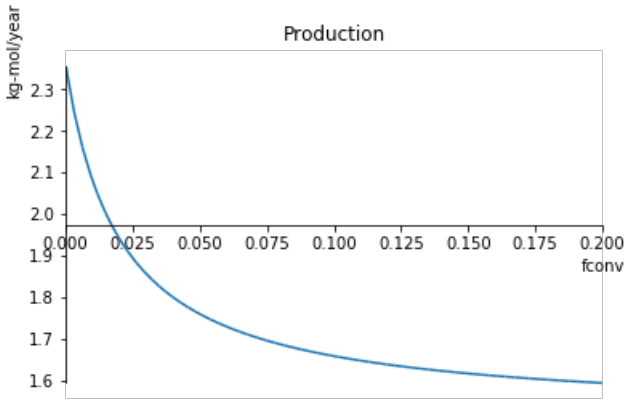
$$\frac{9800.0 fconv}{8844.5 fconv + 280.5}$$



In [17]:

```
production = 24*350*(soln[nEO4] + soln[nE4] + soln[nO4])/1000000

display(sym.simplify(production))
sym.plot(production,(fconv,0,0.2),xlabel='fconv',ylabel='kg-mol/year',title='Production
');
```

$$\frac{74.2938 fconv + 2.3562}{49.0 fconv + 1.0}$$



Parametric Analysis: Fractional Recovery of Ethylene Oxide to Product

Will improving the recovery of Ethylene Oxide to the product stream improve the product purity? Make a prediction, then check against the results of the following calculations.

In [18]:

```
feed_spec = [
    sym.Eq(nE1,196.0),
    sym.Eq(nO1, 84.5)]

reactor_spec = [
    sym.Eq(nE2 - nE3, 0.06*nE2)]

frcvr = sym.var('frcvr')
separator_spec = [
    sym.Eq(nE5, 0.98*nE3),
    sym.Eq(nO5, 0.995*nO3),
    sym.Eq(nEO4, frcvr*nEO3)]

specifications = feed_spec + reactor_spec + separator_spec
for s in specifications:
    display(s)
```

$$nE_1 = 196.0$$

$$nO_1 = 84.5$$

$$nE_2 - nE_3 = 0.06nE_2$$

$$nE_5 = 0.98nE_3$$

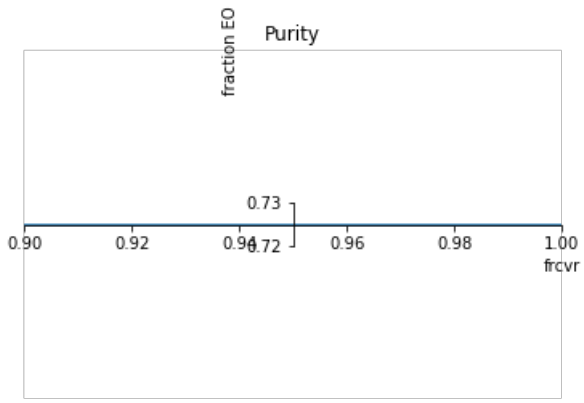
$$nO_5 = 0.995nO_3$$

$$nEO_4 = frcvrnEO_3$$

In [19]:

```
soln = sym.solve(material_balances + specifications, exclude=[frcvr])

purity = soln[nEO4]/(soln[nEO4]+soln[nE4]+soln[nO4])

display(purity)
sym.plot(purity,(frcvr,0.9,1.00),xlabel='frcvr',ylabel='fraction EO',title='Purity');
```

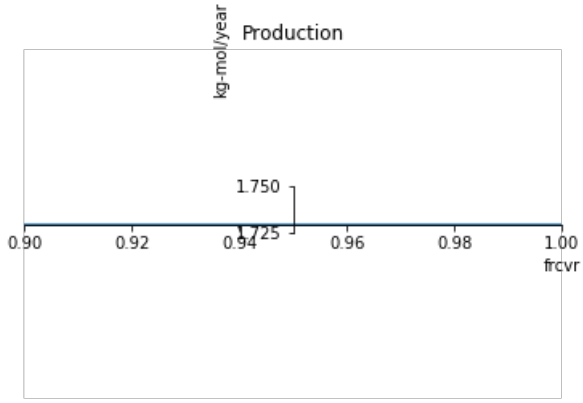
$$0.724878878656755$$



In [20]:

```
production = 24*350*(soln[nEO4] + soln[nE4] + soln[nO4])/1000000.0
display(production)
sym.plot(production,(frcvr,0.9,1.00),xlabel='frcvr',ylabel='kg-mol/year',title='Product
ion');
```

$$1.72939796954315$$



Discussion Questions

- Is this what you expected?
- Why doesn't the product purity or production depend on the fractional recovery of ethylene oxide in the separator?

Parametric Analysis: Fractional Recovery of Ethylene to Recycle

Increasing the fraction of ethylene recovered for recycle should improve product purity. Let's see what happens when we raise it.

In [21]:

```
feed_spec = [
    sym.Eq(nE1,196.0),
    sym.Eq(nO1, 84.5)]

reactor_spec = [
    sym.Eq(nE2 - nE3, 0.06*nE2)]

frcvr = sym.var('frcvr')
separator_spec = [
    sym.Eq(nE5, frcvr*nE3),
    sym.Eq(nO5, 0.995*nO3),
    sym.Eq(nEO4, 0.97*nEO3)]

specifications = feed_spec + reactor_spec + separator_spec
for s in specifications:
    display(s)
```

$$nE_1 = 196.0$$

$$nO_1 = 84.5$$

$$nE_2 - nE_3 = 0.06nE_2$$

$$nE_5 = frcvrnE_3$$

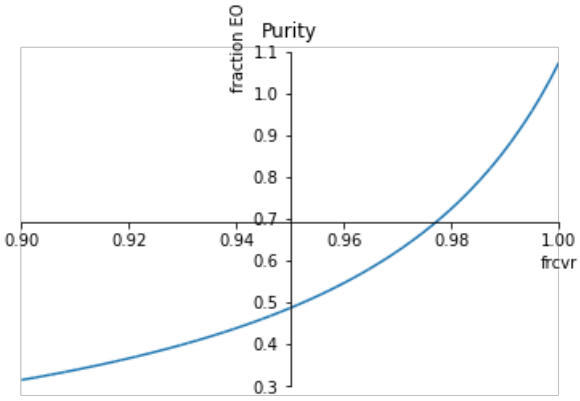
$$nO_5 = 0.995nO_3$$

$$nEO_4 = 0.97nEO_3$$

In [22]:

```
soln = sym.solve(material_balances + specifications,exclude=[frcvr])

purity = soln[nEO4]/(soln[nEO4]+soln[nE4]+soln[nO4])
display(sym.simplify(purity))
sym.plot(purity,(frcvr,0.9,1),xlabel='frcvr',ylabel='fraction EO',title='Purity');
```

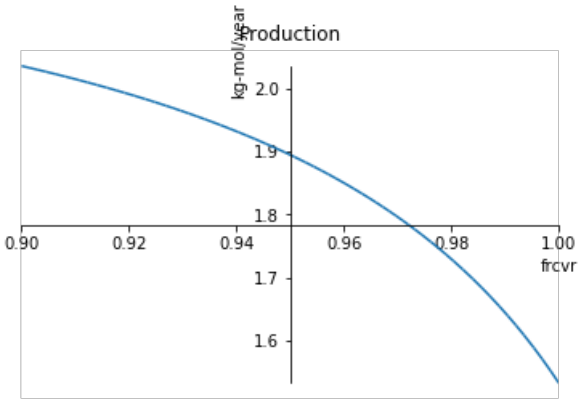
$$-\frac{588.0}{13183.5frcvr - 13731.0}$$



In [23]:

```
production = 24*350*(soln[nEO4] + soln[nE4] + soln[nO4])/1000000.0
display(sym.simplify(production))
sym.plot(production,(frcvr,0.9,1.00),xlabel='frcvr',ylabel='kg-mol/year',title='Product
ion');
```

$$\frac{110.7414 frcvr - 115.3404}{47.0 frcvr - 50.0}$$



Discussion Questions

- Did this behave as you expected?
- Increasing the ethylene recovery to 1.00 leads to a product purity greater than 1.0. Obviously that's not possible. What's going wrong?
- What is the maximum possible recovery of ethylene for recycle?

Parametric Analysis: Change the Oxygen Feed Rate

As we've discovered, the oxygen feedrate is not sufficient to consume all of the ethylene. Let's explore what happens if we change the oxygen feedrate.

In [24]:

```
oxyfeed = sym.var('oxyfeed')
feed_spec = [
    sym.Eq(nE1,196.0),
    sym.Eq(nO1,oxyfeed)]

reactor_spec = [
    sym.Eq(nE2 - nE3, 0.06*nE2)]

separator_spec = [
    sym.Eq(nE5, 0.98*nE3),
    sym.Eq(nO5, 0.995*nO3),
    sym.Eq(nEO4, 0.97*nEO3)]

specifications = feed_spec + reactor_spec + separator_spec
for s in specifications:
    display(s)
```

$$nE_1 = 196.0$$

$$nO_1 = oxyfeed$$

$$nE_2 - nE_3 = 0.06nE_2$$

$$nE_5 = 0.98nE_3$$

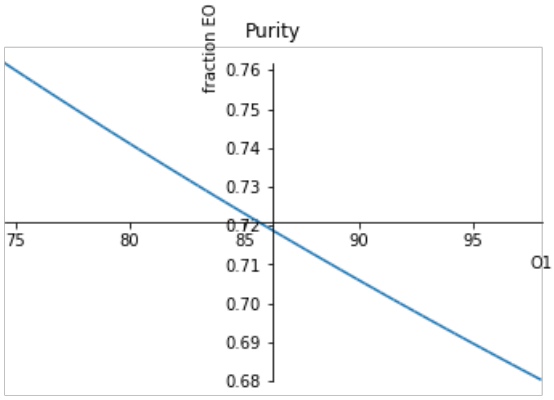
$$nO_5 = 0.995nO_3$$

$$nEO_4 = 0.97nEO_3$$

In [25]:

```
soln = sym.solve(material_balances + specifications, exlude=[oxyfeed])

purity = soln[nEO4]/(soln[nEO4]+soln[nE4]+soln[nO4])
display(sym.simplify(purity))
sym.plot(purity,(oxyfeed,74.5,196/2),xlabel='O1',ylabel='fraction EO',title='Purity');
```

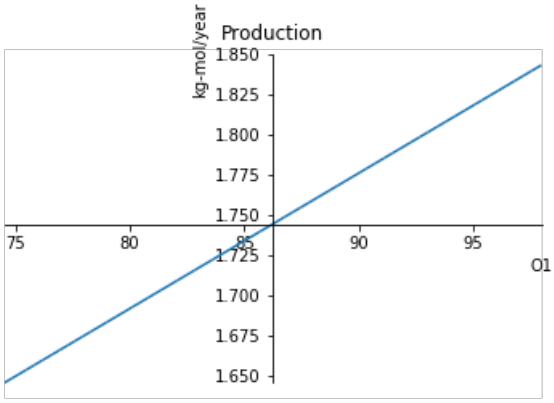
$$\frac{149.238578680203}{oxyfeed + 121.380710659898}$$



In [26]:

```
production = 24*350*(soln[nEO4] + soln[nE4] + soln[nO4])/1000000
display(sym.simplify(production))
sym.plot(production,(oxyfeed,74.5,196/2),xlabel='O1',ylabel='kg-mol/year',title='Produc
tion');
```

$$0.0084 oxyfeed + 1.01959796954315$$



Discussion Questions

- Did this behave as you expected?
- Why did the product purity decrease as the oxygen feedrate was increased?

Conclusions

- The nominal process specifications yield a product purity of 72.5 mol% and a production of 1.73 million kgmol/year. The product purity falls significantly short of the desired purity of 98 mol%.
- Product purity can be increased by increasing the single-pass reactor conversion, increasing the recovery of ethylene to recycle, and decreasing oxygen feed. Individual, however, these changes are not sufficient to meet to the desired purity.
- Can you find specifications that will meet the 98% product purity specification?

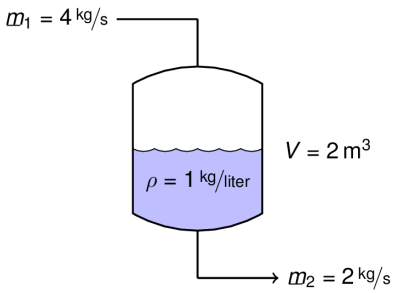
4.3 General Mass Balance on a Single Tank

Summary

This Jupyter notebook demonstrates the application of a mass balance to a simple water tank. This example is adapted with permission from learnCheme.com, a project at the University of Colorado funded by the National Science Foundation and the Shell Corporation.

Problem Statement

Draw a Diagram



Mass Balance

Using our general principles for a mass balance

$$\frac{d(\rho V)}{dt} = \dot{m}_1 - \dot{m}_2$$

which can be simplified to

$$\frac{dV}{dt} = \frac{1}{\rho}(\dot{m}_1 - \dot{m}_2)$$

where the initial value is $V(0) = 1\,\mathrm{m}^3$. This is a differential equation.

Numerical Solution using `odeint`

There are a number of numerical methods available for solving differential equations. Here we use `odeint` which is part of the `scipy` package. `odeint` requires a function that returns the rate of accumulation in the tank as a function of the current volume and time.

In [1]:

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

from scipy.integrate import odeint
```

In [2]:

```
# Flowrates in kg/sec
m1 = 4.0
m2 = 2.0

# Density in kg/m**3
rho = 1000.0

# Function to compute accumulation rate
def dV(V,t): return (m1 - m2)/rho;
```

Next we import `odeint` from the `scipy.integrate` package, set up a grid of times at which we wish to find solution values, then call `odeint` to compute values for the solution starting with an initial condition of 1.0.

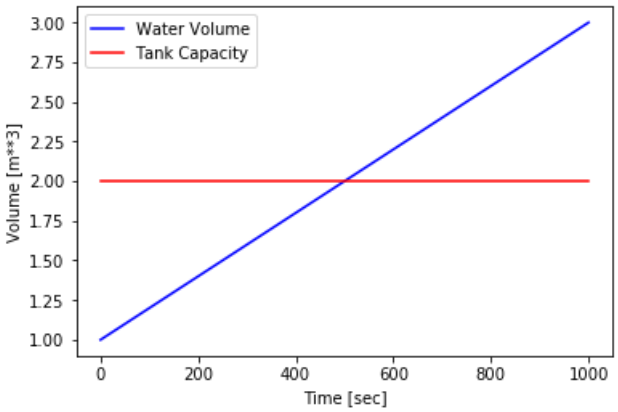
In [3]:

```
t = np.linspace(0,1000)
V = odeint(dV,1.0,t)
```

We finish by plotting the results of the integration and comparing to the capacity of the tank.

In [4]:

```
plt.plot(t,V,'b',t,2*np.ones(len(t)),'r')
plt.xlabel('Time [sec]')
plt.ylabel('Volume [m**3]')
plt.legend(['Water Volume','Tank Capacity'],loc='upper left');
```



This same approach can be used solve systems of differential equations. For an light-hearted (but very useful) example, check out this solution for the Zombie Apocalypse.

Solving for the Time Required to Fill the Tank

Now that we know how to solve the differential equation, next we create a function to compute the air volume of the tank at any given time.

In [5]:

```
Vtank = 2.0
Vinitial = 1.0

def Vwater(t):
    return odeint(dV,Vinitial,[0,t])[-1][0]

def Vair(t):
    return Vtank - Vwater(t)

print("Air volume in the tank at t = 100 is {:4.2f} m**3.".format(Vair(100)))
```

```
Air volume in the tank at t = 100 is 0.80 m**3.
```

The next step is find the time at which `Vair(t)` returns a value of 0. This is root finding which the function `brentq` will do for us.

In [6]:

```
from scipy.optimize import brentq

t_full = brentq(Vair,0,1000)

print("The tank will be full at t = {:6.2f} seconds.".format(t_full))
```

```
The tank will be full at t = 500.00 seconds.
```

Exercise

Suppose the tank was being used to protect against surges in water flow, and the inlet flowrate was a function of time where

$\dot{m}_1 = 4e^{-t/500}$

- Will the tank overflow?
- Assuming it doesn't overflow, how long would it take for the tank to return to its initial condition of being half full? To empty completely?
- What will be the peak volume of water in the tank, and when will that occur?

In [ ]:

```

```

4.4 Unsteady-State Material Balances

Summary

This Jupyter notebook demonstrates the application of the general material balance equations to a variety of example problems.

Unsteady-State Material Balances

The general principle of material balances is stated in words as

Accumulation = Inflow - Outflow + Generation - Consumption

An equation like this can be written for any conserved quantity, whether it is chemical species that chemical or bio engineers work with, or moeny in the case of finance, or populations for social scientists. Whenever terms on the right-hand side of the equation do not yield a zero result we are left with an *unsteady-state* balance, which are some of the most fascinating and challenging problems in engineering practice.

Example 1: Population Growth

The first example concerns population growth. Looking birth and death rates in various countries around the globe, one comes across the following data maintained by the World Bank.

For Afghanistan (the first country on the World Bank charts), the 2011 population was estimated to be 29,105,480 with a birth rate of 37.0 births per year per 1,000 people, and a death rate as 8.0 per year per 1,000. Assuming there is no in-migration or out-migration, can we predict the population in 2014?

Population Balance Equation

Writing out the balance equation, we get

$$\underbrace{\text{Accumulation}}_{\frac{dP}{dt}} = \underbrace{\text{Inflow}}_{=0} - \underbrace{\text{Outflow}}_{=0} + \underbrace{\text{Generation}}_{births = \frac{37}{1000}P} - \underbrace{\text{Consumption}}_{deaths = \frac{8}{1000}P}$$

the algebraic combination of the two terms on the left leaves us an equation for the rate of change of population

$$\frac{dP}{dt} = 0.029\,P$$

with an initial condition $P(2011) = 29,105,480$.

Calculus Solution

We'll use the shorthand notation $t_0 = 2011$ and $P(t_0) = 29,105,480$. Then separating the variables

$$\int_{P(t_0)}^{P(t)} \frac{1}{P}\, dP = 0.029 \int_{t_0}^{t} dt$$

Doing the integrals (if this is the first time you've seen this, be sure you do this by hand!) leads to

$$P(t) = P(t_0)\, e^{0.029(t-t_0)}$$

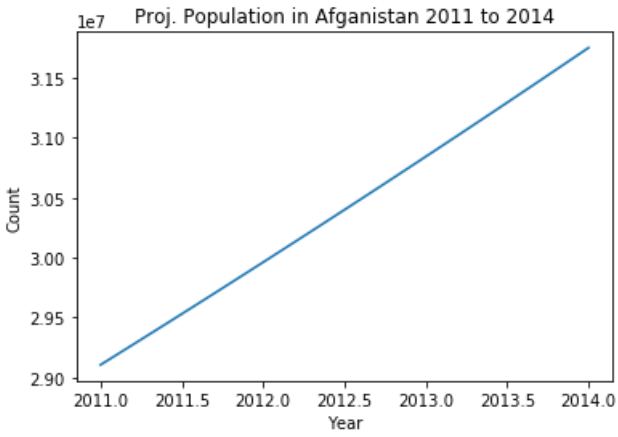
This solution demonstrates the exponential growth of populations in circumstances where growth rate is constant and there are no other factors to consider.

In [1]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

P_0 = 29105480
t_0 = 2011
t_1 = 2014

t = np.linspace(t_0,t_1)
P = P_0*np.exp(0.029*(t-t_0))

plt.plot(t,P)
plt.xlabel('Year')
plt.ylabel('Count')
plt.title("Proj. Population in Afganistan {:4d} to {:4d}".format(t_0,t_1))
```

Out[1]:

```
<matplotlib.text.Text at 0x11172b400>
```



Numerical Solution

Another way of generating solutons to unsteady-state problems is by direct, numerical solution of the differential equations. This requires us to set up a function to numerically evaluate the right hand-side of the differential equation. That function, along with initial conditions and a list of time values for which we want to the know the solution, are passed to a solver that does the hard work.

Step 1. Right a function to evaulate the RHS of the differential equation.

In [2]:

```
def Inflow(t) : return 0
def Outflow(t): return 0
def Births(P) : return (37.0/1000)*P
def Deaths(P) : return (8.0/1000)*P

def Accumulation(P,t) : return Inflow(t) - Outflow(t) + Births(P) - Deaths(P)
```

Step 2. Establish the initial condition and time values for a desired solution.

In [3]:

```
P_0 = 29105480
t_0 = 2011
t_1 = 2014
t = np.linspace(t_0,t_1)
```

Step 3. Pass this information to a solver to compute values of the solution.

The `SciPy` package provides a number of tools for the numerical solution of differential equations. Of these, perhaps the easiest to use is `odeint` which is demonstrated here. `odeint` needs to be imported from `scipy.integrate`, and then given three pieces of information:

1. A function that takes two arguments. The first argument is the value of quantity we're trying to integrate, and the second is time. The function must return the value of the right-hand side of the differential equations.
2. The initial value of the quantity we're trying to integrate.
3. A list of times at which we wish to evaluate the solution. The first value in the list must be the initial value of time.

In [4]:

```
P = odeint(Accumulation,P_0,t)
```

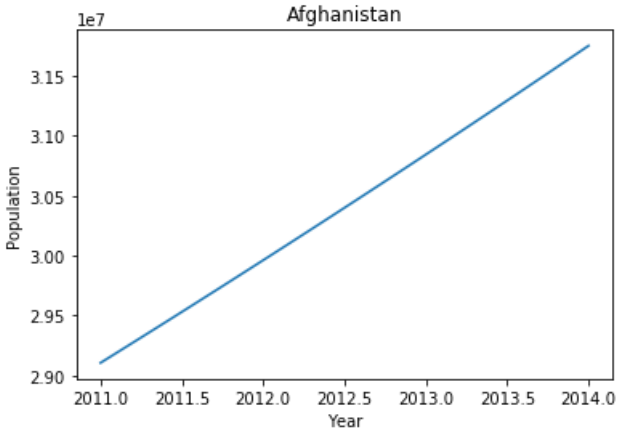
Step 4. Plot the result.

In [5]:

```
plt.plot(t,P)
plt.xlabel('Year')
plt.ylabel('Population')
plt.title('Afghanistan')
```

Out[5]:

```
<matplotlib.text.Text at 0x1118c46a0>
```



Example 2: Dilution of a Trace Contaminant

We have a tank partially filled with an initial volume $V(t_0)$ = 1000 liters of water that is contaminated by a toxic species X at an concentration $C_X(t_0)$ = 100 ppm by volume. We wish to dilute this to 20 ppm by volume before sending it for further cleanup. We know adding 4000 liters of water would accomplish the goal, but for process monitoring purposes we'd like to compute concentration as a function of time assuming water enters at a steady rate of q(t) = 1 liter/second.

The total amount of X in the tank is the product of concentration times volume, that is C_XV. The material balance for X is

For component X: $\underbrace{\text{Accumulation}_X}_{\frac{d(C_XV)}{dt}} = \underbrace{\text{Inflow}_X}_{=0} - \underbrace{\text{Outflow}_X}_{=0} + \underbrace{\text{Generation}_X}_{=0} - \underbrace{\text{Consumption}_X}_{=0}$

This is a two component system, so we need two material balances. We'll do the second balance on overall volume on the assumption that density is a constant this dilute solution. The overall balance is

For total volume: $\underbrace{\text{Accumulation}}_{\frac{dV}{dt}} = \underbrace{\text{Inflow}}_{=q} - \underbrace{\text{Outflow}}_{=0} + \underbrace{\text{Generation}}_{=0} - \underbrace{\text{Consumption}}_{=0}$

This gives us a pair of differential equations

$$\begin{aligned} \frac{d(C_XV)}{dt} &= 0 \\ \frac{dV}{dt} &= q \end{aligned}$$

What we're looking for is C_X. But what we have are differential equations for product (C_XV) and volume V. How can we get a differential equation for C_X?

Using the chain rule

$$\frac{d(C_XV)}{dt} = C_X \underbrace{\frac{dV}{dt}}_{=q} + V\frac{dC_X}{dt} = 0$$

Substituting in the second equation leaves us with a pair of differential equations

$$\begin{aligned} \frac{dC_X}{dt} &= -\frac{q}{V}C_X \\ \frac{dV}{dt} &= q \end{aligned}$$

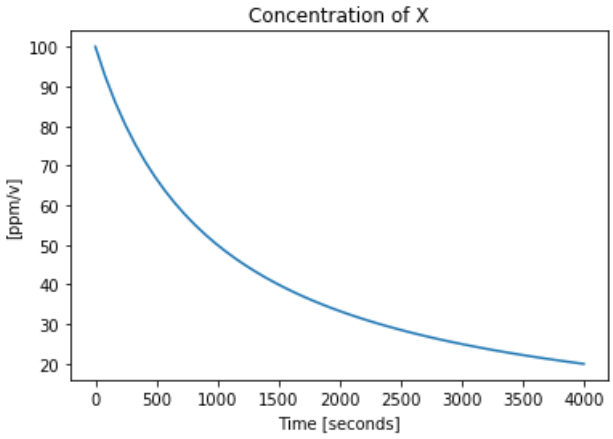
(Be sure you go through the algebra and understand both how and why we did this.)

In [6]:

```
# parameter values
q = 1           # liters/second

# initial conditions
C_0 = 100       # ppm
V_0 = 1000      # liters

# time grid in seconds
t = np.linspace(0,4000)

# compute list of derivative values
def deriv(X,t):
    C,V = X
    return [-q*C/V,q]

# solve with odeint
soln = odeint(deriv,[C_0,V_0],t)

# display the result
plt.plot(t,soln[:,0])
plt.xlabel('Time [seconds]')
plt.ylabel('[ppm/v]')
plt.title('Concentration of X')
```

Out[6]:

```
<matplotlib.text.Text at 0x111a15630>
```



Example 3: Hare and Lynx

$$\frac{dH}{dt} = rH\left(1 - \frac{H}{k}\right) - \frac{aHL}{c+H}$$
$$\frac{dL}{dt} = \frac{baHL}{c+H} - dL$$

Parameter Values

In [7]:

```
a = 3.2
b = 0.6
c = 50
d = 0.56
k = 125
r = 1.6
```

Derivatives

In [8]:

```
def deriv(x,t):
    H,L = x
    dH = r*H*(1-H/k) - a*H*L/(c+H)
    dL = b*a*H*L/(c+H) - d*L
    return [dH,dL]
```

Solve

In [9]:

```
t = np.linspace(0,150,500)
ic = [20,30]
soln = odeint(deriv,ic,t)
```

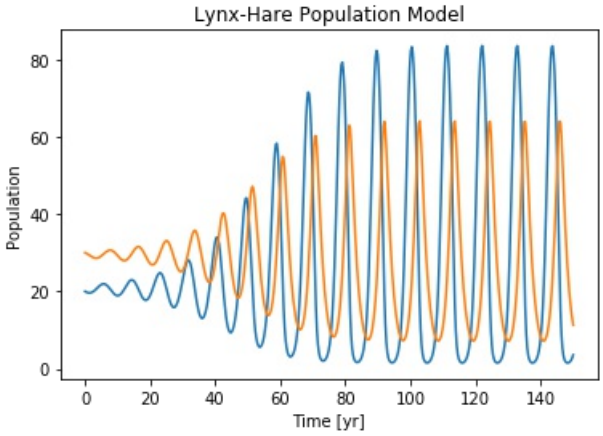
Plot

In [10]:

```
plt.plot(t,soln)
plt.xlabel('Time [yr]')
plt.ylabel('Population');
plt.title('Lynx-Hare Population Model')
```

Out[10]:

```
<matplotlib.text.Text at 0x111b2ecc0>
```

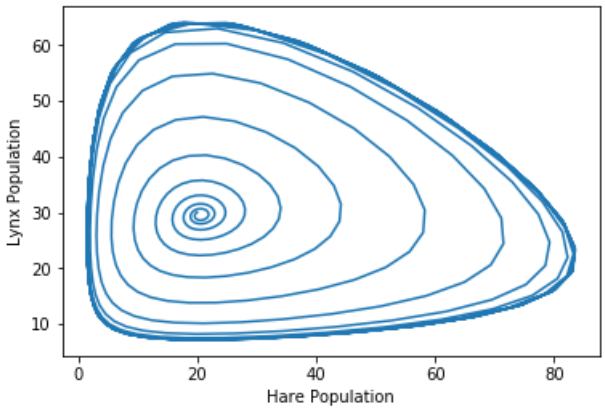


In [11]:

```
H = soln[:,0]
L = soln[:,1]
plt.plot(H,L)
plt.xlabel('Hare Population')
plt.ylabel('Lynx Population')
```

Out[11]:

```
<matplotlib.text.Text at 0x111ca3978>
```



In [ ]:

Chapter 5. Reactors

5.1 Dehydrogenation of Propane

Process Model

In [1]:

```
from sympy import *

# define constants

nfeed = 100.0

# define variables

var('X')
var('n1:11')

# define constants

# unit balances

mixer = [
    Eq(nfeed + n9, n1),      # C3H8
    Eq(n10, n2)              # C3H6
]

reactor = [
    Eq(n3, n1 - X),          # C3H8
    Eq(n4, n2 + X),          # C3H6
    Eq(n5, X)                # H2
]

separator = [
    Eq(n3, n6 + n9),         # C3H8
    Eq(n4, n7 + n10),        # C3H6
    Eq(n5, n8)               # H2
]

# process specifications

specs = [
    Eq(n6, (1-0.95)*nfeed), # 95% process conversion
    Eq(n6, 0.00555*n3),      # 0.555% of propane recovered in propylene product
    Eq(n10, 0.05*n7)         # propylene recycle is 5% of outlet flow
]

soln = solve(mixer + reactor + separator + specs)
soln
```

Out[1]:

```
{X: 95.0000000000000,
 n1: 995.900900900901,
 n10: 4.75000000000000,
 n2: 4.75000000000000,
 n3: 900.900900900901,
 n4: 99.7500000000000,
 n5: 95.0000000000000,
 n6: 5.00000000000000,
 n7: 95.0000000000000,
 n8: 95.0000000000000,
 n9: 895.900900900901}
```

Product Composition

In [2]:

```
nTotal = soln[n6] + soln[n7] + soln[n8]
print('C3H8 Product = ', round(100*soln[n6]/nTotal,2), '%')
print('C3H6 Product = ', round(100*soln[n7]/nTotal,2), '%')
print('  H2 Product = ', round(100*soln[n8]/nTotal,2), '%')
```

```
C3H8 Product =  2.56 %
C3H6 Product =  48.72 %
  H2 Product =  48.72 %
```

Recycle Ratio

In [3]:

```
print('Recycle Ratio = ', (soln[n9] + soln[n10])/nfeed)
```

```
Recycle Ratio =  9.00650900900901
```

Single Pass Conversion

In [4]:

```
print('Single Pass Conversion', (soln[n1] - soln[n3])/soln[n1])
```

```
Single Pass Conversion 0.0953910172328011
```

In [ ]:

```

```

5.2 Steam Reforming of Methane

Summary

This notebook demonstrates the formulation and solution of a reactor equilibrium problem at a known temperature and pressure. The notebook uses functional programming techniques including Python's lambda and map functions.

Problem Statement

The steam reforming of methane

$$CH_4(g) + H_2O(g) \longrightarrow CO(g) + 3\,H_2(g)$$

is an endothermic reaction for the production of hydrogen. The reaction is normally conducted at relatively high temperatures and pressures over a heterogeneous catalyst. Assuming the reaction is carried out with 20% excess water at 20 atm, solve for the equilibrium extent of reaction as a function of temperature. Then plot methane predicted conversion as a function of operating temperature.

Solution

For an ideal mixture of gases at equilibrium,

$$K_{rxn}(T) = \prod_{c=1}^{C} (y_c P)^{\nu_c}$$

or, after taking logarithms,

$$\ln K_{rxn}(T) = \sum_{c=1}^{C} \nu_c \ln(y_c P)$$

where $K_{rxn}(T)$ is the equilibrium constant, y_c is the mole fraction of component c, ν_c is the stoichiometric coefficient of component c, P and T are pressure and temperature, respectively.

The strategy for solving this problem is to use the Van't Hoff equation to write a Python function to compute the left-hand side of the equation as a function of temperature, and the component material balances to solve for the quotient on the right-hand side as a function of the extent of reaction. Then, given a temperature and pressure, the equation is solved for the equilibrium extent of reaction using a root-finding algorithm.

Notebook Initialization

Initialize the Python workspace.

In [1]:

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

Problem Data

The problem data is aggregated into the following cells for convenience. Adjust these parameters to do parametric studies of reactor performance.

In [2]:

```
# Reaction Conditions
P = 20
Tmin = 800
Tmax = 1200

# Reactor Feed Specifications
nIn = dict()
nIn['CH4'] = 1
nIn['H2O'] = 1.2*nIn['CH4']
nIn['CO'] = 0
nIn['H2'] = 0

# List of Components
C = nIn.keys()
```

Component Data

We begin by gathering the molar enthalpy and Gibb's free energy of the species participating in the reaction at standard conditions.

Species	$\Delta\hat{G}_f^\circ$ [kJ/gmol]	$\Delta\hat{H}_f^\circ$ [kJ/gmol]
Carbon Monoxide (g)	-137.27	-110.53
Hydrogen (g)	0	0
Methane (g)	-50.49	-74.52
Water (g)	-228.59	-241.83

In [3]:

```
# Enthalpy and Free Energy of Formation
Gf = dict()
Hf = dict()

Gf['CO'] = -137270
Hf['CO'] = -110530

Gf['H2'] = 0
Hf['H2'] = 0

Gf['CH4'] = -50490
Hf['CH4'] = -74520

Gf['H2O'] = -228590
Hf['H2O'] = -241830
```

Enthalpy and Gibb's Free Energy of Reaction

The enthalpy and Gibb's free energy per mole extent of reaction are given by

$$\Delta\hat{H}^\circ_{rxn} = \sum_{c=1}^{C} \nu_c \Delta\hat{H}^\circ_{f,c}$$

$$\Delta\hat{G}^\circ_{rxn} = \sum_{c=1}^{C} \nu_c \Delta\hat{G}^\circ_{f,c}$$

where ν_c is the stoichiometric coefficient for species c.

In [4]:

```
# Reaction Stoichiometry
nu = dict()
nu['CO']  = +1
nu['H2']  = +3
nu['CH4'] = -1
nu['H2O'] = -1

Hr = sum([nu[c]*Hf[c] for c in C])
print("Hr = {:7.3f} kJ/gmol".format(Hr/1000))

Gr = sum([nu[c]*Gf[c] for c in C])
print("Gr = {:7.3f} kJ/gmol".format(Gr/1000))
```

```
Hr = 205.820 kJ/gmol
Gr = 141.810 kJ/gmol
```

Equilibrium Constant using van't Hoff Equation

The equilibrium constant as a function of temperature can be estimated using the Van't Hoff equation.

$$\ln K_r(T) = -\frac{1}{R}\left[\frac{\Delta\hat{G}^\circ_{rxn} - \Delta\hat{H}^\circ_{rxn}}{298} + \frac{\Delta\hat{H}^\circ_{rxn}}{T}\right]$$

This is implemented in Python as a lambda function of T

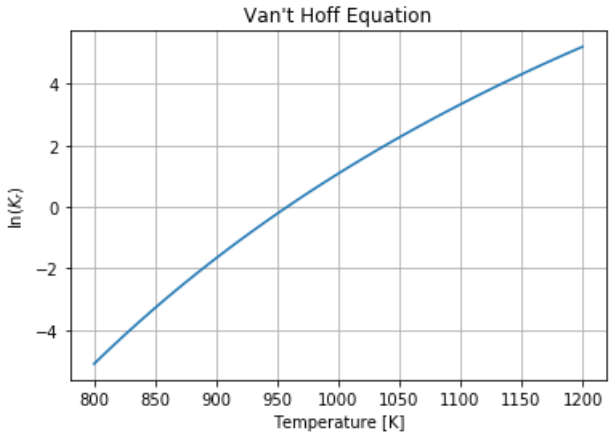
In [5]:

```
R = 8.314 # J/K/gmol
def lnKr(T):
    return -((Gr-Hr)/298 + Hr/T)/R
```

which can be plotted over the temperature range of interest.

In [6]:

```
T = np.linspace(Tmin,Tmax,100)
plt.plot(T,[lnKr(T) for T in T])
plt.xlabel('Temperature [K]')
plt.ylabel('$\ln(K_{r})$')
plt.title('Van\'t Hoff Equation')
plt.grid()
```



Material Balances

Given the problem data, the material balances can be solved to determine the molar flows at the outlet of the reactor as functions of the molar extent of reaction $\dot{\xi}$

$$\begin{aligned}
\dot{n}_{out,CO} &= \dot{n}_{in,CO} + \nu_{CO}\dot{\xi} \\
\dot{n}_{out,H_2} &= \dot{n}_{in,H_2} + \nu_{H_2}\dot{\xi} \\
\dot{n}_{out,CH_4} &= \dot{n}_{in,CH_4} + \nu_{CH_4}\dot{\xi} \\
\dot{n}_{out,H_2O} &= \dot{n}_{in,H_2O} + \nu_{H_2O}\dot{\xi}
\end{aligned}$$

The material balances are implemented as lambda functions of molar extent x. The lambda functions are assigned to a dictionary nOut. (The extra default argument c=c is added so that the c referenced in the lambda function is the same as the key of the dictionary. Without the c=c argument, the c inside the lambda function refers to counter variable rather than the desired value of the key.)

In [7]:

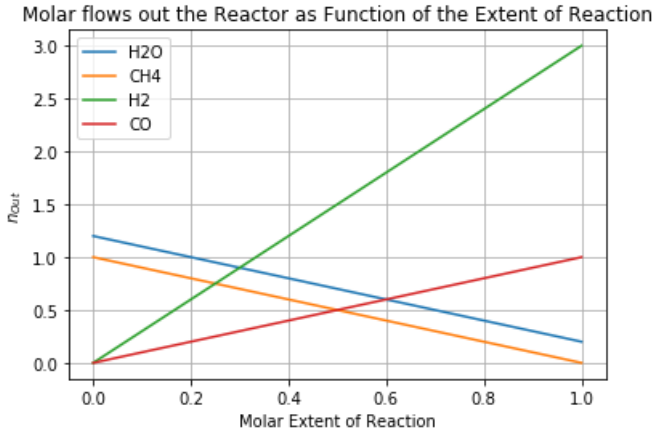
```
nOut = dict()
for c in C:
    nOut[c] = lambda x,c=c: nIn[c] + nu[c]*x
```

The outlet molar flows can be plotted as functions of molar extent of reaction. The first step is to compute the maximum possible extent, then to plot each of the molar flows as a function of extent.

In [8]:

```
xMax = np.Inf
for c in C:
    if nu[c] < 0:
        xMax = min(xMax,-nIn[c]/nu[c])

print('Maximum molar extent of reaction = {:.2f}'.format(xMax))

x = np.linspace(0,xMax)

for c in C:
    plt.plot(x,[nOut[c](x) for x in x])
plt.legend(C)
plt.xlabel('Molar Extent of Reaction')
plt.ylabel('$n_{Out}$')
plt.title('Molar flows out the Reactor as Function of the Extent of Reaction')
plt.grid()
```

```
Maximum molar extent of reaction = 1.00
```



Composition of the Reactor Outlet Gases

The composition of the reactor outlet gases is given by

$$y_n(\dot{\xi}) = \frac{\dot{n}_{out,n}(\dot{\xi})}{\dot{n}_{Total}(\dot{\xi})}$$

where the total molar flow is sum of the component molar flows.

$$\begin{aligned}\dot{n}_{Total} &= \sum_{c=1}^{C} \dot{n}_{out,c} \\ &= \sum_{c=1}^{C} \dot{n}_{in,c} + \left(\sum_{c=1}^{C} \nu_c\right)\dot{\xi}\end{aligned}$$

The total molar flow is implement as a Python function.

In [9]:

```
def nTotal(x):
    nTotal = 0
    for c in C:
        nTotal += nOut[c](x)
    return nTotal
```

which is used to create a dictionary of lambda functions for the composition of the reactor effluent as functions of extent of reaction.

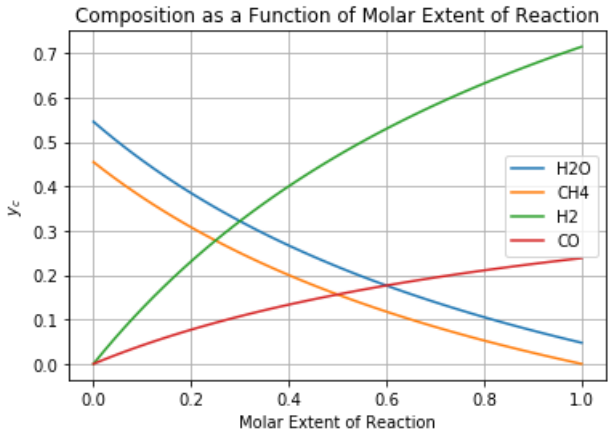
In [10]:

```
y = dict()
for c in C:
    y[c] = lambda x,c=c: nOut[c](x)/nTotal(x)
```

Plotting composition as a function of extent of reaction.

In [11]:

```
x = np.linspace(0,1)
for c in C:
    plt.plot(x,[y[c](x) for x in x])
plt.legend(C)
plt.xlabel('Molar Extent of Reaction')
plt.ylabel('$y_c$')
plt.title('Composition as a Function of Molar Extent of Reaction')
plt.grid()
```



Reaction Quotient

For a gas phase reaction involving a mixture of ideal gases, we define a reaction quotient as

$$K_a(\dot{\xi}) = \prod_{c=1}^{C} \left(y_c(\dot{\xi}) P \right)^{\nu_c}$$

where $y_c(\dot{\xi})P$ is the partial pressure of component c. Taking the logarithm

$$\ln K_a(\dot{\xi}) = \sum_{c=1}^{C} \nu_c \ln(y_c(\dot{\xi})P)$$

This is implemented as a Python function.

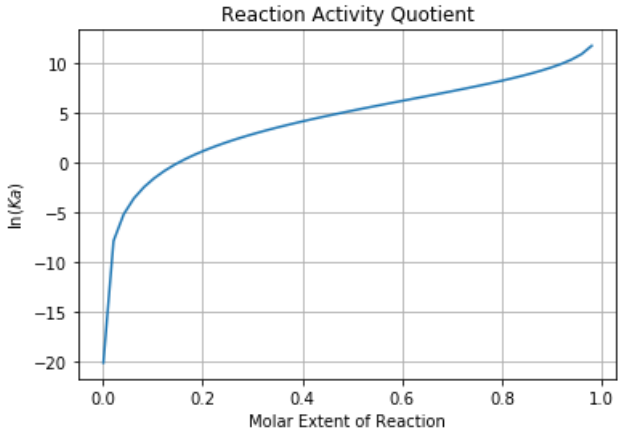
In [12]:

```
def lnKa(x):
    lnKa = 0;
    for c in C:
        lnKa += nu[c]*np.log(P*y[c](x))
    return lnKa
```

In [13]:

```
x = np.linspace(0.001,1)
plt.plot(x,[lnKa(x) for x in x])
plt.xlabel('Molar Extent of Reaction')
plt.ylabel('$\ln(Ka)$')
plt.title('Reaction Activity Quotient');
plt.grid()
```

```
/Users/jeff/anaconda/lib/python3.5/site-packages/ipykernel/__main__.py:4:
RuntimeWarning: divide by zero encountered in log
```



Solving for the Equilibrium Extent of Reaction

For a given temperature and pressure, the solution for the equilibrium extent of reaction is the value of $\dot{\xi}$ for which $K_r(T) = K_a(\dot{\xi})$ which can be written in terms of logarithms as

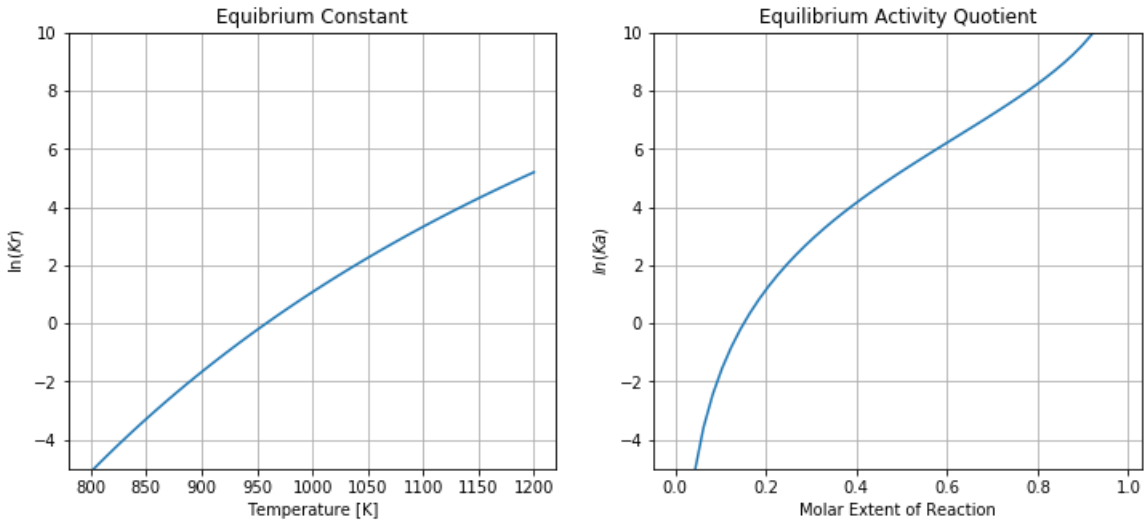
$$\ln K_r(T) = \ln K_a(\dot{\xi})$$

Plotting these functions side-by-side shows a simple graphical technique for finding solutions for $\dot{\xi}$.

In [14]:

```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(T,[lnKr(T) for T in T])
plt.ylim(-5,10)
plt.xlabel('Temperature [K]')
plt.ylabel('$\ln(Kr)$')
plt.title('Equibrium Constant')
plt.grid()

plt.subplot(1,2,2)
plt.plot(x,[lnKa(x) for x in x])
plt.ylim(-5,10)
plt.xlabel('Molar Extent of Reaction')
plt.ylabel('$ln(Ka)$')
plt.title('Equilibrium Activity Quotient')
plt.grid()
```

```
/Users/jeff/anaconda/lib/python3.5/site-packages/ipykernel/__main__.py:4:
RuntimeWarning: divide by zero encountered in log
```



We'll let $\dot{\xi}_{eq}(T)$ denote equilibrium value of the extent as a function of temperature. Those values are defined as roots to the equation

$$\ln K_a(\dot{\xi}_{eq}(T)) - \ln K_r(T) = 0$$

This implemented as a Python lambda function where a root=finding algorithm is used to solve the equilibrium condition as a function of temperature.

In [15]:

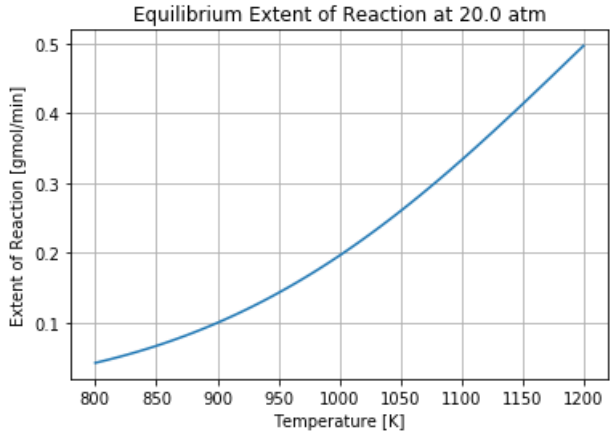
```
from scipy.optimize import brentq as fzero

xEquil = lambda T: fzero(lambda x: lnKa(x) - lnKr(T), 0, xMax)
```

Plotting

In [16]:

```
plt.plot(T,[xEquil(T) for T in T])
plt.xlabel('Temperature [K]')
plt.ylabel('Extent of Reaction [gmol/min]')
plt.title('Equilibrium Extent of Reaction at {:.1f} atm'.format(P))
plt.grid()
```

```
/Users/jeff/anaconda/lib/python3.5/site-packages/ipykernel/__main__.py:4:
RuntimeWarning: divide by zero encountered in log
```



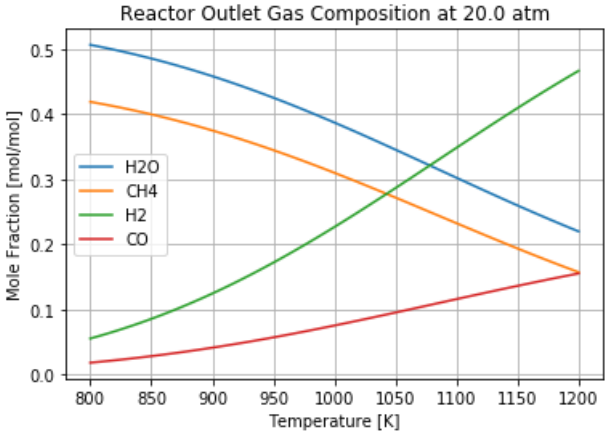
Equilibrium Composition

Now that we have a function to compute the equilibrium extent of reaction as a function of temperature, we have everything needed to calculate a variety of performance metrics for the reactor. For example, we can use the functions we've already created for the outlet gas composition to compute and plot the reactor outlet gas composition as a function of temperature.

In [17]:

```
yEquil = dict()
for c in C:
    yEquil[c] = lambda T,c=c: y[c](xEquil(T))

for c in C:
    plt.plot(T,[yEquil[c](T) for T in T])

plt.legend(C)
plt.xlabel('Temperature [K]')
plt.ylabel('Mole Fraction [mol/mol]')
plt.title('Reactor Outlet Gas Composition at {:.1f} atm'.format(P))
plt.grid()
```

```
/Users/jeff/anaconda/lib/python3.5/site-packages/ipykernel/__main__.py:4:
RuntimeWarning: divide by zero encountered in log
```



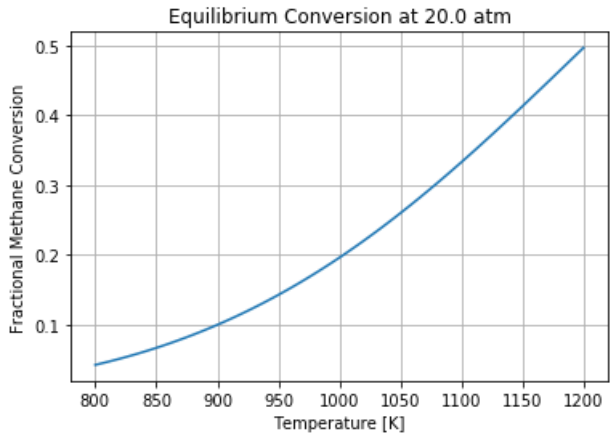
Methane Conversion

$$f_{conv} = \frac{\dot{n}_{in,CH_4} - \dot{n}_{out,CH_4}}{\dot{n}_{in,CH_4}}$$

In [18]:

```
fconv = lambda T: (nIn['CH4'] - nOut['CH4'](xEquil(T)))/nIn['CH4']

plt.plot(T,[fconv(T) for T in T])
plt.xlabel('Temperature [K]')
plt.ylabel('Fractional Methane Conversion')
plt.title('Equilibrium Conversion at {:.1f} atm'.format(P))
plt.grid()
```

```
/Users/jeff/anaconda/lib/python3.5/site-packages/ipykernel/__main__.py:4:
RuntimeWarning: divide by zero encountered in log
```



Exercises

1. Repeat the calculations for different operating pressures. How does equilibrium extent of reaction depend on operating pressure? Why is this the case?
2. Repeat the calculations for different feed ratios for water and methane. How does extent of reaction depend on the feed ratio of water?
3. Find component heat capacity data, then write a function to compute the heat requirement for the reactor. If the reactor is operated adiabatically, what is the temperature difference between the inlet and outlet streams?

In [ ]:

```
```

Chapter 6. Vapors and Gases

6.1 PVT Computations for Non-ideal Gases

Problem Statement

Commerical CO_2 cartridges have a large number of uses including bicycle tire and life jacket inflators, soda dispensers, compressed gas cleaners for electronic devices, and gas powered guns. They can be purchased at low-cost from sports store and on-line.

A popular size of CO_2 cartridge holds 12g in an internal volume of 17.6 cm^3. Estimate the pressure inside the cartridge under hot (40°C = 104°F = 313.15 K) conditions.

In [22]:

```
# problem data
n = 12/44                    # gram-moles
V = 0.0176/n                 # specific volume in liters/gmol
T = 40 + 273.15              # Kelvin

# gas constant
R = 0.08206                  # gas constant liter-atm per kelvin-gmol

# critical properties of CO2
Tcritical = 304.25           # Kelvin
Pcritical = 72.9             # atm
acentric_factor  = 0.225    # Pitzer acentric factor

print('molar specific volume =', round(V,4), 'liters/gmol')
print('molar density =', round(1/V, 2), 'gmol/liter')

# a dictionary to store results
predictions = {}
```

```
molar specific volume = 0.0645 liters/gmol
molar density = 15.5 gmol/liter
```

Estimating Pressure

Ideal Gas Law

Our first attempt at computing the cartridge pressure will use the ideal gas law. Written in terms of molar specific volume

$$\hat{V} = \frac{V}{n}$$

the ideal gas law is given by

$$P\hat{V} = RT$$

Here we solve for P using the problem data. Before going further, decide if the results of this calculation seem realistic to you.

In [3]:

```
P_ideal_gas = R*T/V

predictions['Ideal Gas'] = P_ideal_gas

print('CO2 Pressure (Ideal Gas) =', round(P_ideal_gas, 1), 'atm')
print('CO2 Pressure (Ideal Gas) =', round(14.696*P_ideal_gas, 1), 'psia')
```

```
CO2 Pressure (Ideal Gas) = 398.2 atm
CO2 Pressure (Ideal Gas) = 5851.9 psia
```

Compressiblity Charts

Compressibility is defined as

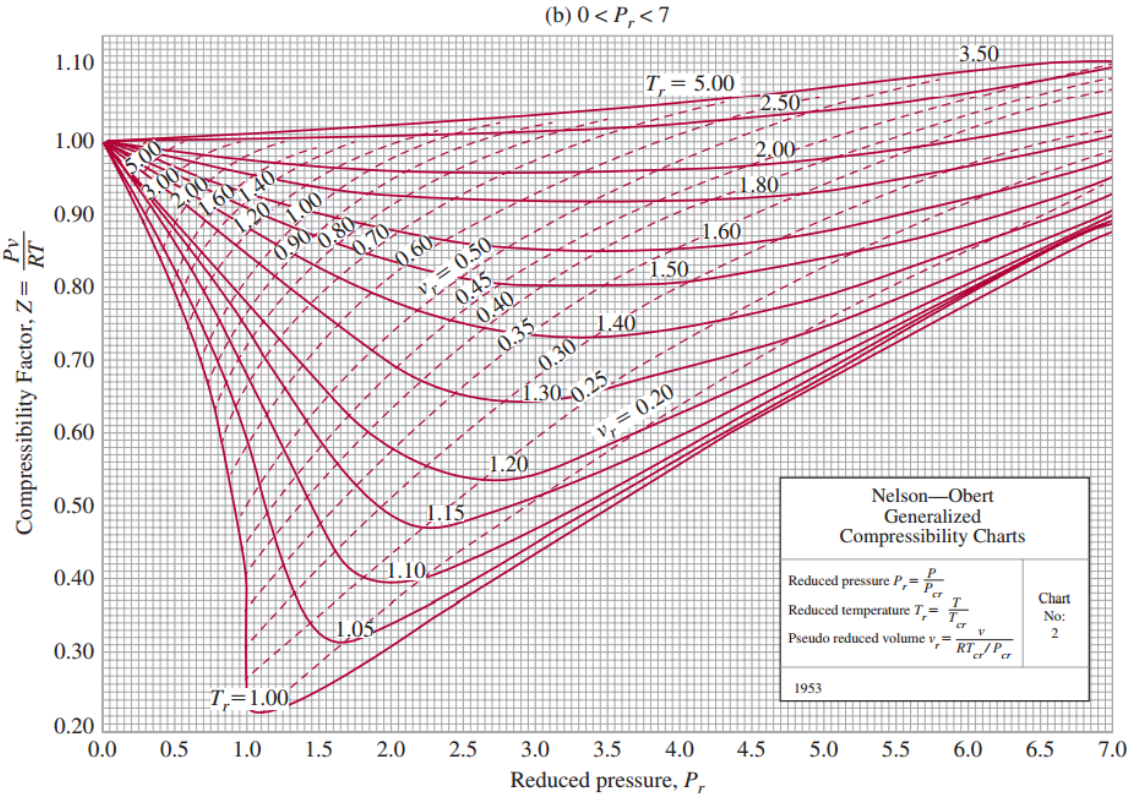
$$z = \frac{P\hat{V}}{RT}$$

which, by definition, has a value $z = 1$ for an ideal gas. Real gases display a significant deviation from 1, generally increasing with increasing pressure as the gas molecules get closer together, and decreasing with temperature.

The principle of corresponding states, developed by Johannes Diderik van der Waals in 1873, correlates the compressibility of real gases to the critical temperature T_c and critical pressure P_c. For this purpose, the **reduced temperature** T_r and **reduced pressure** P_r are defined as

$$T_r = \frac{T}{T_c}$$
$$P_r = \frac{P}{P_c}$$

A compressibility chart presents the averaged compressibility measured for a number of species. Given values of the reduced temperature and pressure, T_r and P_r, one locates the value of compressibility factor z, then use the above relationship to compute molar volume $\hat{V}$.



For the case where V is one of the known variables it is convenient to define an **ideal reduced volume**. The first step is to define an ideal critical volume

$$\hat{V}_c^{ideal} = \frac{RT_c}{P_c}$$

The reason for the superscript 'ideal' is because this is a ficti

Then the ideal reduced volume $\hat{V}_r^{ideal}$ is

$$\hat{V}_r^{ideal} = \frac{\hat{V}}{\hat{V}_c} = \frac{P_c\hat{V}}{RT_c}$$

The compressibility chart is augmented with contours of constant $\hat{V}_r^{ideal}$. Given values for $\hat{V}_r^{ideal}$ and either P_r or T_r, a corresponding value of z is located. The compressibility definition is then used to solve for the unknown variable.

In [23]:

```
Tr = T/Tcritical
Vr = Pcritical*V/(R*Tcritical)

print('reduced temperature =', round(Tr,2))
print('ideal reduced volume =', round(Vr,2))
```

```
reduced temperature = 1.03
ideal reduced volume = 0.19
```

In [5]:

```
# look up values (https://pubs.acs.org/doi/pdf/10.1021/ie50467a036)
z = 0.28

# compute pressure
P_compressibility = z*R*T/V

predictions['Compressibility'] = P_compressibility


print('CO2 Pressure =', round(P_compressibility, 2), 'atm')
print('CO2 Pressure =', round(14.696*P_compressibility, 2), 'psia')
```

```
CO2 Pressure = 111.5 atm
CO2 Pressure = 1638.54 psia
```

While we still can't be sure this estimate is accurate, we can definitely see the estimate based on the ideal gas law is completley off-base. The reason is that the cartridge is filled to a point where the contents are close to the critical point on the phase diagram.

Virial Model

The virial expansion was first proposed by a Kamerlingh Onnes, a physicist who won the Nobel prize in 1911 for his work on superconductivity and liquid helium. (He also coined the work 'enthalpy'). The key idea is to create a infinite series approximation for compressibility in the form

$$\frac{P\hat{V}}{RT} = A + \frac{B}{\hat{V}} + \frac{C}{\hat{V}^2} + \frac{D}{\hat{V}^3} \cdots$$

where A, B, C are temperature dependent and known as the first, second, and third virial coefficients, respectively. The case $A = 1$ and $B = C = D = \cdots = 0$ corresponds to an ideal gas.

A commonly used version of this expansion is to assume $A = 1$, $B(T)$, and $C = D = \cdots = 0$, with the further approximation $\hat{V} = \frac{RT}{P}$. Then

$$\frac{P\hat{V}}{RT} = 1 + \frac{BP}{RT}$$

This can be simplified to read

$$P = \frac{RT}{\hat{V} - B}$$

The temperature dependent value of B is estimated by

$$\begin{aligned} B_0 &= 0.083 - \frac{0.422}{T_r^{1.6}} \\ B_1 &= 0.139 - \frac{0.172}{T_r^{4.2}} \\ B &= \frac{RT_c}{P_c}(B_0 + \omega B_1) \end{aligned}$$

ω is the **Pitzer acentric factor**, values of which are tabulated in standard sources for chemical data.

In [18]:

```
Tr = T/Tcritical
B0 = 0.083 - 0.422/Tr**1.6
B1 = 0.139 - 0.172/Tr**4.2
B = (R*Tcritical/Pcritical)*(B0 + acentric_factor*B1)
print('B =', B)

P_virial = R*T/(V - B)

predictions['Virial'] = P_virial

print('CO2 Pressure =', round(P_virial, 2), 'atm')
print('CO2 Pressure =', round(14.696*P_virial, 2), 'psia')
```

```
B = -0.11061596372197276
CO2 Pressure = 146.72 atm
CO2 Pressure = 2156.13 psia
```

van der Waals Equation of State

The **van der Waals** equation of state has the form

$$P = \frac{RT}{\hat{V} - b} - \frac{a}{\hat{V}^2}$$

where values for the coefficients a and b are determined by the critical point temperature and pressure

$$a = \frac{27R^2T_c^2}{64P_c} \qquad b = \frac{RT_c}{8P_c}$$

The parameter a accounts for long-range attractive forces acting between molecules, and b for short-range repulsive forces.

In [19]:

```
a = 27*R**2*Tcritical**2/(64*Pcritical)
b = R*Tcritical/(8*Pcritical)
print('a =', a, '  b =', b)

P_vdw = R*T/(V-b) - a/V**2

predictions['van der Waals'] = P_vdw

print('CO2 Pressure =', round(P_vdw, 2), 'atm')
print('CO2 Pressure =', round(14.696*P_vdw, 2), 'psia')
```

```
a = 3.6072850418404214   b = 0.042809936556927296
CO2 Pressure = 316.73 atm
CO2 Pressure = 4654.72 psia
```

Soave-Redlich-Kwong Equation of State

The Soave-Redlick-Kwong equation of state is one of most widely used equations of states, and proven to be applicable to a wide variety of systems. The general expression

$$P = \frac{RT}{\hat{V} - b} - \frac{\alpha a}{\hat{V}(\hat{V} + b)}$$

where the parameters are given by

$$\begin{aligned}
a &= 0.42747\frac{(RT_c)^2}{P_c} \\
b &= 0.08664\frac{RT_c}{P_c} \\
m &= 0.48508 + 1.55171\omega - 0.1561\omega^2 \\
T_r &= \frac{T}{T_c} \\
\alpha &= \left[1 + m(1 - \sqrt{T_r})\right]^2
\end{aligned}$$

In [8]:

```
from math import sqrt

a = 0.42747*(R*Tcritical)**2/Pcritical
b = 0.08664*R*Tcritical/Pcritical
m = 0.48508 + 1.5517*acentric_factor - 0.1561*acentric_factor**2
Tr = T/Tcritical
alpha = (1 + m*(1-sqrt(Tr)))**2

P_srk = R*T/(V-b) - alpha*a/V/(V+b)

predictions['SRK'] = P_srk

print('CO2 Pressure =', round(P_srk, 2), 'atm')
print('CO2 Pressure =', round(14.696*P_srk, 2), 'psia')
```

```
CO2 Pressure = 150.24 atm
CO2 Pressure = 2207.97 psia
```

Reference Data from NIST Webbook

The National Institute of Standards and Technology (NIST) maintains a web site devoted to the distribution of standard reference data. The NIST Chemistry WebBook is an excellent source of carefully curated data on over 7000 organic and small inorganic compounds.

NIST Fluid Properties Data for CO_2

In [9]:

```
print('density =', 1/V, 'mol/liter')
```

```
density = 15.495867768595039 mol/liter
```

In [20]:

```
P_nist = 108.13  # atm

predictions['NIST Webbook'] = P_nist

print('CO2 Pressure =', round(P_nist, 2), 'atm')
print('CO2 Pressure =', round(14.696*P_nist, 2), 'atm')
```

```
CO2 Pressure = 108.13 atm
CO2 Pressure = 1589.08 atm
```

Comparison of Estimates

In [21]:

```
print('{0:15s}  {1:9s}   {2:5s}'.format('EOS', 'Pressure', 'Error'))

for key,val in predictions.items():
  err = 100*(val-P_nist)/P_nist
  print('{0:15s}  {1:5.1f} atm   {2:5.1f}%'.format(key, val, err))
```

```
EOS              Pressure    Error
Ideal Gas        398.2 atm   268.3%
Compressibility  111.5 atm     3.1%
Virial           146.7 atm    35.7%
van der Waals    316.7 atm   192.9%
SRK              150.2 atm    38.9%
NIST             108.1 atm     0.0%
NIST Webbook     108.1 atm     0.0%
```

Non-Ideal Behavior

How does the CO_2 pressure depend on temperature in the range from 30C to 50C?

Discussion Points

1. Why is the ideal gas law so wrong?
2. Why do the compressibility charts work so well in this case? Do you expect them to work well in others?
3. How would the calculations change if we were given pressure and temperature, and asked to compute molar volume?

6.2 Hydrogen Storage in a Fuel Cell Vehicle

Data

In [ ]:

```
# Hydrogen Data

R = 0.08206                 # liter-atm/(gmol-K)
MW = 2.01588                # molecular mass

Tcritical = 33.20           # Kelvin
Pcritical = 12.8            # atm

acentric_factor = -0.220    # wikipedia
```

In [2]:

```
# Problem Data

mass = 5.0                  # mass (kg)
T = 298                     # operating temperature (K)
P = 1 + 70000.0/101.325     # operating pressure (atm)

print('Operating temperature =', T, 'K')
print('Operating pressure =', round(P,2), 'atm')
```

```
Operating temperature = 298 K
Operating pressure = 691.85 atm
```

Problem 1. Required Tank Volume

Given the problem and chemical data above, including a required tank volume sufficient to hold 5 kg of compressed hydrogen gas, compute the required volume the tanks must accomodate at a temperature of 298 K.

Solution using Ideal Gas Law

The very high pressures make it unlikely that the ideal gas law will give an accurate answer. Nevertheless, we use the ideal gas model to establish a reference value to assess other solutions. For an ideal gas,

$$P\hat{V} = RT$$

In [3]:

```
Vmolar = R*T/P

gmols = mass*1000/MW
Vtank = gmols*Vmolar

print("Molar Volume =", round(Vmolar,5), "liters/gmol")
print("Tank Volume =", round(Vtank,2), "liters")
```

```
Molar Volume = 0.03535 liters/gmol
Tank Volume = 87.67 liters
```

Solution using Compressibility Charts

Newton's Correction

The operating point for the hydrogen tank will be a several multiples of both the critical temperature and critical pressure. As noted in Felder, et al., when using generalized compressibility charts for hydrogen or helium, a so-called 'Newton's Correction' is needed.

$$T_c^a = T_c + 8K$$
$$P_c^a = P_c + 8 \text{ atm}$$

We then compute the reduced temperature and pressure based on the adjusted critical point data.

In [4]:

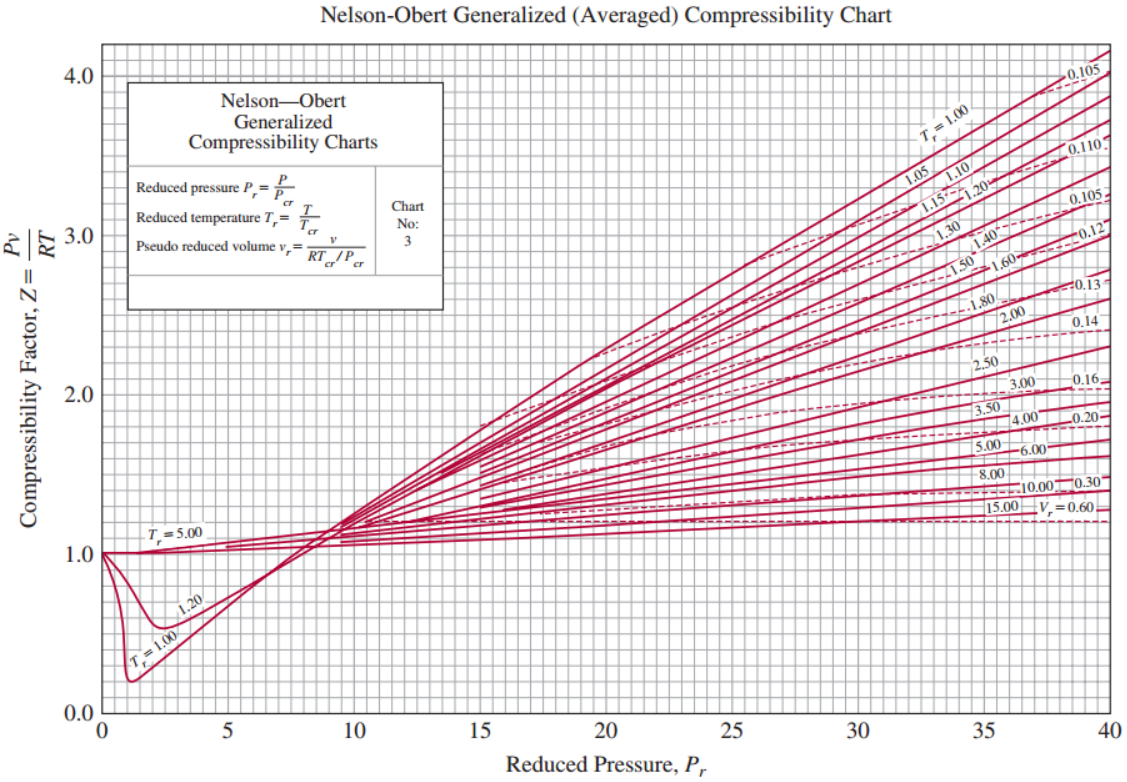
```
Treduced = T/(Tcritical + 8)
Preduced = P/(Pcritical + 8)
print("Adjusted Reduced Temperature =", Treduced)
print("Adjusted Reduced Pressure =", Preduced)
```

```
Adjusted Reduced Temperature = 7.233009708737864
Adjusted Reduced Pressure = 33.261840706788895
```

Nelson-Obert Generalized Compressibility Chart

The generalized compressibility charts provide estimates for compressibility as a function of reduced pressure and temperature. The estimates are the result of averaging data for a set of gases.

$$z = \frac{P\hat{V}}{RT}$$



In [5]:

```
# chart lookup
z = 1.45

Vmolar = z*R*T/P

gmols = mass*1000/MW
Vtank = gmols*Vmolar

print("Molar Volume =", round(Vmolar,5), "liters/gmol")
print("Tank Volume =", round(Vtank,2), "liters")
```

```
Molar Volume = 0.05125 liters/gmol
Tank Volume = 127.12 liters
```

Solution using Soave-Redlich-Kwong Equation

The Soave-Redlich-Kwong equation of state is given by

$$P = \frac{RT}{\hat{V}-b} - \frac{\alpha a}{\hat{V}(\hat{V}+b)}$$

where

$$\begin{aligned} a &= 0.42747\frac{(RT_c)^2}{P_c} \\ b &= 0.08664\frac{RT_c}{P_c} \\ m &= 0.48508 + 1.55171\omega - 0.1561\omega^2 \\ T_r &= \frac{T}{T_c} \\ \alpha &= \left[1 + m\left(1-\sqrt{T_r}\right)\right]^2 \end{aligned}$$

Equations of state provide a relationship among three intensive variables -- P, T, and $\hat{V}$. Given two of these variables, an equation of state provides a means to compute the third.

There are three cases:

- **Compute pressure** P given temperature T and molar volume $\hat{V}$. This is the most straightforward case. The equations explicitly give P and be used exactly as written.
- **Compute temperature** T given pressure P and molar volume $\hat{V}$. This is not a common case but can be solved with straightforward iteration. The iteration starts by estimating T with, say, the ideal gas law. Then compute the temperature dependent parameters T_r, α, the constant parameters a, b, and m, then finally solve for T as
$$T = \frac{\hat{V}-b}{R}\left(P + \frac{\alpha a}{\hat{V}(\hat{V}+b)}\right)$$
Repeat the calculation until T converges to a constant value.
- **Compute molar volume** $\hat{V}$ given temperature T and pressure P. This calculation requires the solution a cubic equation for $\hat{V}$ which requires some care in coding. It is somewhat more convenient to rework the equation by defining parameters
$$\begin{aligned} A &= \frac{\alpha a P}{R^2T^2} \\ B &= \frac{bP}{RT} \end{aligned}$$
to yield a cubic equation for compressibility z
$$z^3 - z^2 + (A - B - B^2)z - AB = 0$$
The following code cell demonstrates the use of `fsolve` from the `scipy.optimize` library for the solution to the compressibility equation.

In [6]:

```
a = 0.42747*R**2*Tcritical**2/Pcritical
b = 0.08664*R*Tcritical/Pcritical
m = 0.48508 + 1.55171*acentric_factor - 0.1561*acentric_factor**2
Tr = T/Tcritical
alpha = (1 + m*(1-Tr**0.5))**2

A = alpha*a*P/(R**2*T**2)
B = b*P/(R*T)

def f(z):
  return z**3 - z**2 + (A-B-B**2)*z - A*B

from scipy.optimize import fsolve
z = fsolve(f, 1)[0]

Vmolar = z*R*T/P

gmols = mass*1000/MW
Vtank = gmols*Vmolar

print("Molar Volume =", round(Vmolar,5), "liters/gmol")
print("Tank Volume =", round(Vtank,2), "liters")
```

```
Molar Volume = 0.05205 liters/gmol
Tank Volume = 129.09 liters
```

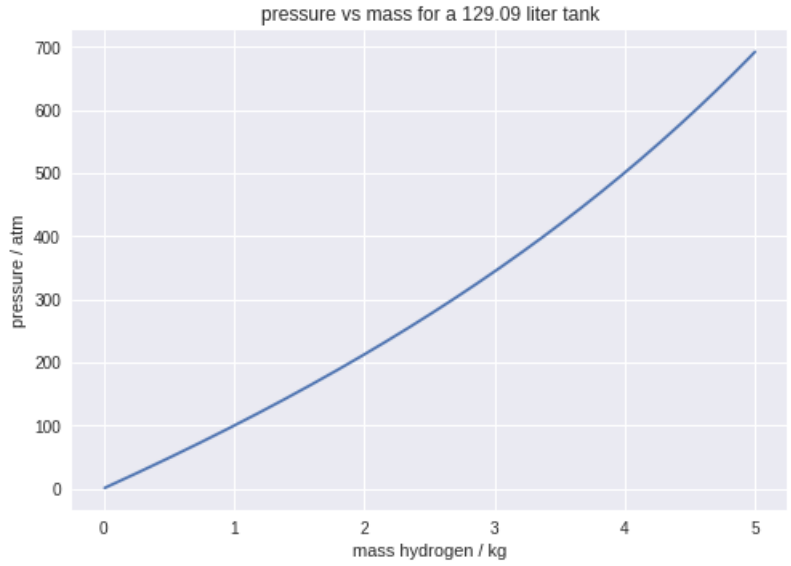
Problem 2. Estimating Mass of Hydrogen using a Pressure Sensor

Drivers need to accurately monitor the amount of hydrogen fuel remaining in the tank, and refueling stations need to accurately meter the amount of hydrogen transferred to the tank. For this purpose, create a graph of pressure as a function of mass in the tank assuming an operating temperature of 298 K.

Solution using SRK equation of state.

In [11]:

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

mass = np.linspace(.01,5,200)

Vmolar = Vtank/(mass*1000/MW)

a = 0.42747*R**2*Tcritical**2/Pcritical
b = 0.08664*R*Tcritical/Pcritical
m = 0.48508 + 1.55171*acentric_factor - 0.1561*acentric_factor**2
Tr = T/Tcritical
alpha = (1 + m*(1-Tr**0.5))**2

Ptank = R*T/(Vmolar - b) - alpha*a/(Vmolar*(Vmolar+b))

plt.plot(mass, Ptank)
plt.xlabel('mass hydrogen / kg')
plt.ylabel('pressure / atm')
plt.title('pressure vs mass for a ' + str(round(Vtank,2)) + ' liter tank');
```



For an ideal gas at constant temperature, the pressure mass relationship would be a straight line. The nonlinear nature of this solution demonstrates the tank is operating in a regime where hydrogen gas behaves as a nonideal gas.

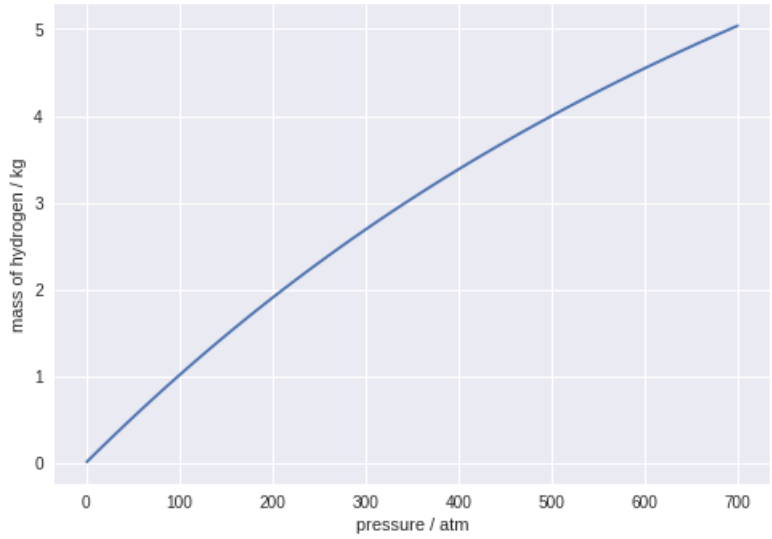
Problem 3. Python function to estimate hydrogen mass.

(Note: This problem requires some Python coding skills, including the definition of functions.)

Building on the last problem, write a Python function that returns an estimate the mass of hydrogen in the tank given measurements of pressure, temperature, and the tank volume.

In [15]:

```
from scipy.optimize import fsolve

def mass_in_tank(P, T=298, Vtank=129.09):
  a = 0.42747*R**2*Tcritical**2/Pcritical
  b = 0.08664*R*Tcritical/Pcritical
  m = 0.48508 + 1.55171*acentric_factor - 0.1561*acentric_factor**2

  Tr = T/Tcritical
  alpha = (1 + m*(1-Tr**0.5))**2

  A = alpha*a*P/(R**2*T**2)
  B = b*P/(R*T)

  f = lambda z: z**3 - z**2 + (A-B-B**2)*z - A*B
  z = float(fsolve(f,1.0))

  Vmolar = z*R*T/P      # molar volume
  return MW*Vtank/Vmolar/1000


# demonstrate solution

P = np.linspace(1,700)
plt.plot(P, [mass_in_tank(p) for p in P])
plt.xlabel('pressure / atm')
plt.ylabel('mass of hydrogen / kg');
```



Problem 4. Limit on tank leak rate.

The lower flammability limit of hydrogen in air is 4 volume%. Assume the vehicle is stored in a single car garage with interior dimensions 7m (H) x 4m (W) x 2.5m (H), and the air is replenished at a rate of 0.03 air changes per hour. What is the maximum tolerable leakage rate of hydrogen from the car storage tanks? Express the answer as a mass flow. What amount of time would be required for 5kg to safely dissipate by leakage from the tanks. Does this calculation suggestion any engineering challenges?

In [19]:

```
# volume of the garage

Vgarage = 7*4*2.5*1000   # liters
nflow = 0.03*1*Vgarage/(R*T)

print('total molar flowrate through garage =', nflow, 'gmol/hour')

nH2flow = 0.04*nflow

print('maximum hydrogen molar flowrate through garage =', nH2flow, 'gmol/hour')

mH2flow = MW*nH2flow

print('maximum hydrogen molar flowrate through garage =', mH2flow, 'g/hour')

tfinal = 5000/mH2flow

print('minimum time to safely dissipate 5kg hydrogen =', tfinal, 'hours')
print('minimum time to safely dissipate 5kg hydrogen =', tfinal/24, 'days')
```

```
total molar flowrate through garage = 85.87594279517198 gmol/hour
maximum hydrogen molar flowrate through garage = 3.435037711806879 gmol/hour
maximum hydrogen molar flowrate through garage = 6.924623822477252 g/hour
minimum time to safely dissipate 5kg hydrogen = 722.0608841985113 hours
minimum time to safely dissipate 5kg hydrogen = 30.08587017493797 days
```

The engineering challenge is to design a storage tank that dissipates hydrogen at a rate no greater than about 7g/hr when filled to a pressure of 10,000 psig.

In [ ]:

```

```

Chapter 7. Vapor/Liquid Equilibrium

7.1 Gases with One Condensable Component

Examples

Boiling Water

One Saturday afternoon, while distracted by other things, you put room temperature tap water into a tea kettle and place it on to the stove to boil. Like most tea kettles, this one has a spout with a one-way valve that releases any vapor produced during boiling, so the absolute pressure inside the tea kettle is always one atmosphere.

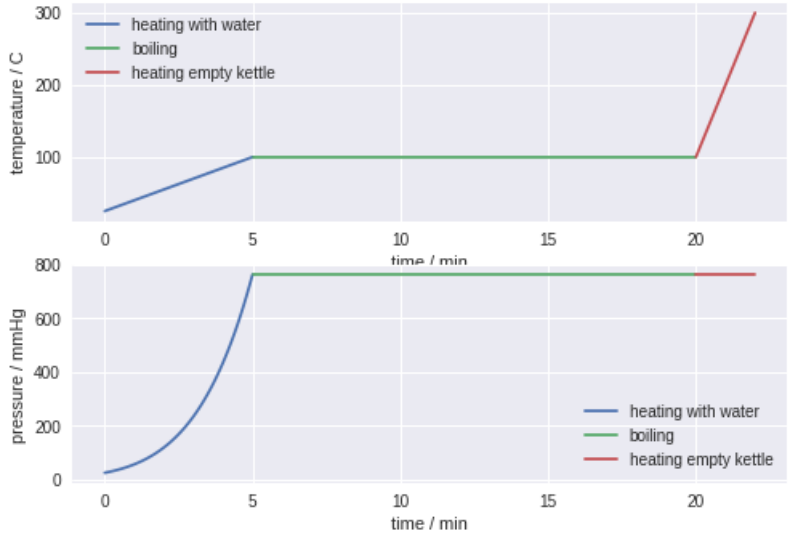
(a.) Sketch a plot of kettle temperture as a function of time. Are there transitions in the slope of this plot? Though you don't have enough information for a completely detailed plot, do indicate any numbers that you can.

(b.) Sketch the vapor pressure of water as a function of time. Again, be as quantitative as possible.

Solution

In [2]:

```
# Antoine equation for water
def Psat(T):
  return 10**(8.10765 - 1750.286/(T + 235.0))

# plot construction
import matplotlib.pyplot as plt
import numpy as np

ta = np.linspace(0,5)
Ta = 25 + 15*ta
Pa = Psat(Ta)

tb = np.linspace(5,20)
Tb = 100 + 0*(tb-5)
Pb = Psat(Tb)

tc = np.linspace(20,22)
Tc = 100 + 100*(tc-20)
Pc = Pb

plt.subplot(2,1,1)
plt.plot(ta, Ta, tb, Tb, tc, Tc)
plt.legend(['heating with water','boiling','heating empty kettle'])
plt.xlabel('time / min')
plt.ylabel('temperature / C')

plt.subplot(2,1,2)
plt.plot(ta, Pa, tb, Pb, tc, Pc)
plt.legend(['heating with water','boiling','heating empty kettle'])
plt.xlabel('time / min')
plt.ylabel('pressure / mmHg');
```



Partial Condensation

You suddenly remember that you left a tea kettle on the stove to boil. You rush over to the stove and luckily catch it in the nick of time. There is still a small amount of liquid water left, and steam is still being discharged from the spout. Yy this time all air in the kettle has been discharged.

You take tea kettle off the stove and let it cool to the room temperature (25 °C). The kettle has an internal volume of 2 liters. You estimate the contents of the kettle has a mass of just 4g.

a.) What is the pressure inside of the cool tea kettle?

b.) What is the mass of liquid water remaining inside of the kettle?

Solution Part a

In [3]:

```
# Antoine equation for water
def Psat(T):
  return 10**(8.10765 - 1750.286/(T + 235.0))

T = 25
print('Pressure inside the cool tea kettle =', Psat(T), 'mmHg')
```

```
Pressure inside the cool tea kettle = 23.756407663704227 mmHg
```

Solution Part b

There are two unknowns in part b. -- the mass of the liquid m_{liq} and the mass of the vapor m_{vap}. There are also two pieces of information -- the total mass of 4 grams and the total volume of 2 liters. Denoted the molar volume as $\hat{V}_{liq}$ and $\hat{V}_{vap}$, respectively, the equations read

$$\begin{aligned} m_{liq} + m_{vap} &= m_{total} \\ \frac{m_{liq}}{M}\hat{V}_{liq} + \frac{m_{vap}}{M}\hat{V}_{vap} &= V_{total} \end{aligned}$$

where M is molecular weight.

For the liquid, the $\hat{V}_{liq}/M$ is just $1/\rho$ where ρ is the liquid density, 1000 g/liter. For the vapor, the molar volume is given by the ideal gas law

$$\hat{V}_{vap} = \frac{RT}{P}$$

Doing some algebra

$$m_{liq} = \frac{\frac{RT}{PM}m_{total} - V_{total}}{\frac{RT}{PM} - \frac{1}{\rho}}$$

In [17]:

```
# Antoine equation for water
def Psat(T):
  return 10**(8.10765 - 1750.286/(T + 235.0))

# problem data
R = 0.08206         # liter-atm/(K-gmol)
T = 25              # temperature C
P = Psat(T)/760.0   # atm
M = 18.01           # molecular mass
rho = 1000.0        # grams/liter
Vtotal = 2          # total volume
mtotal = 4          # total mass

# algebraic solutin
Vmass = R*T/(P*M)
mliq = (Vmass*4 - 2)/(Vmass - 1/rho)
mvap = mtotal - mliq

print('\nAlgebraic Solution:')
print('Mass of liquid =', mliq, 'grams')
print('Mass of vapor =', mvap, 'grams')

# check the solution using sympy
from sympy import *
var('m_liq m_vap')
eqns = [
    Eq(m_liq + m_vap, 4),
    Eq(m_liq/rho + (R*T/P)*m_vap/M, 2)
]

print('\nSolution using Sympy: ')
print(solve(eqns))
```

```
Algebraic Solution:
Mass of liquid = 3.4521153928077695 grams
Mass of vapor = 0.5478846071922305 grams

Solution using Sympy:
{m_liq: 3.45211539280777, m_vap: 0.547884607192231}
```

Relative Humidity

Before the tea-making fiasco the air in the kitchen was a pleasant 25 °C with a relative humidity of 40%. You estimate the ill-fated attempt at tea-making released a half liter of water into a room encompassing 20 square meters with a ceiling height of 2.75 meters. What is the relative humidity after the release of the water vapor?

Solution

Relative humidity is defined as

$$RH = \frac{y_A P}{P_A^{sat}(T)}$$

The solution strategy is to

1. use the ideal gas law to determine how much water is present, in moles, at the start,
2. add the moles generated during the boiling operation to compute a final composition,
3. compute the final relative humidity from the above equation.

In [ ]:

```
T = 25              # deg C
P = 760             # mmHg
R = 0.08206         # liter-atm/(gmol-K)
V = 1000*20*2.75    # liters

RH = 0.4

yH2O = RH*Psat(T)/P
print(' initial water mole fraction =', yH2O)


n = (P/760)*V/(R*(T+273.15))
print('         Total initial moles =', n, 'gmols')

nH2O = yH2O*n
print('      Initial moles of water =', nH2O, 'gmole')

nH2O = nH2O + 500/MW
print('        Final moles of water =', nH2O, 'gmole')

yH2O = nH2O/(n + nH2O)
print('Final mole fraction of water =', yH2O)

print('           Relative humidity =', 100*yH2O*P/Psat(T), '%')

pH2O_25 = yH2O*P
print('   partial pressure of water =', pH2O, 'mmHg')
```

Dew Point

It was a cool evening, and you wake to a kitchen at 15 °C with a covering of dew on the inside of the windows. Assuming there was no ventilation following your tea-making adventure, how much water condensed on the windows? What will be the relative humidity once the room is back to a daytime temperture of 25°C?

Solution

In [ ]:

```
pH2O_15 = Psat(15)

print('initial partial pressure of water =', pH2O_25, 'mmHg')
print('  final partial pressure of water =', pH2O_15, 'mmHg')

nH2O_25 = (pH2O_25/760)*V/(R*(T + 273.15))
nH2O_15 = (pH2O_15/760)*V/(R*(T + 273.15))

print('initial amount of water =', nH2O_25*MW, 'grams')
print('  final amount of water =', nH2O_15*MW, 'gmoles')
print('        condensed water =', nH2O_25*MW - nH2O_15*MW, 'grams')

print(' final relative humidity =', 100*pH2O_15/Psat(25), '%')
```

In [ ]:

```

```

7.2 Vapor-Liquid Equilibrium for Pure Components

Summary

This Jupyter notebook describes the modeling of vapor-liquid equilibrium with Antoine's equation, including the calculation of saturation pressure, saturation temperature, and normal boiling points.

Gibb's Phase Rule

The Gibb's phase rule shows how many independent intensive thermodynamic variables (e.g. T, P, $\hat{V}$, or x_i) are required to completely specify the state of a substance.

$$F = C + 2 - \Pi - r$$

where

$$\begin{aligned} F &= \text{Thermodynamic Degrees of Freedom} \\ C &= \text{Number of Components} \\ \Pi &= \text{Number of Phases} \\ r &= \text{number of independent reactions at equilibrium} \end{aligned}$$

This simple rule that has profound implications for engineering analysis.

Phase Diagram for a Pure Component

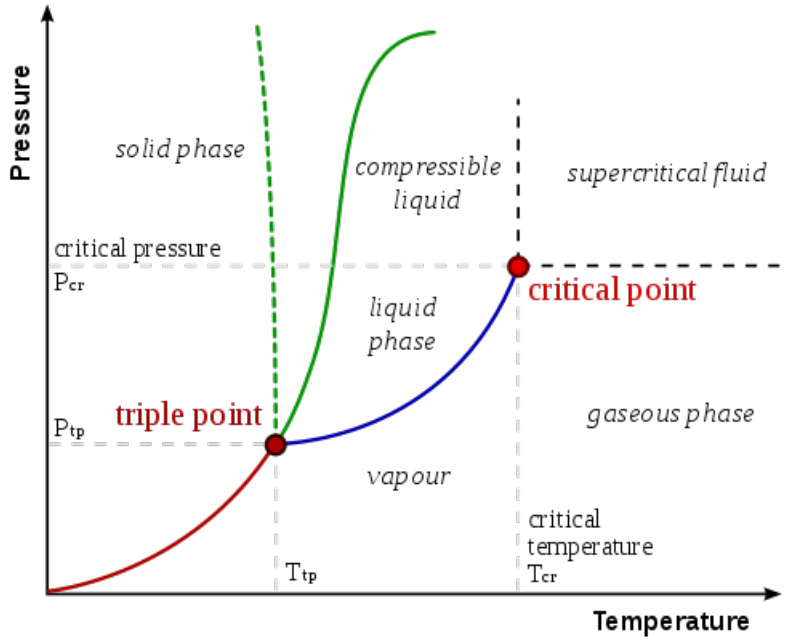
For a pure component (i.e., $C = 1$) and no reactions (i.e., $r = 0$), the Gibb's phase rule reads

$$F = 3 - \Pi$$

which shows:

- Two independent thermodynamic variables, such as T and P, are sufficient to specify the state of a single phase.
- If two phases are in coexistence, then there must be a relationship between T and P.
- The coexistence of three phases completely specifies the thermodynamic state.

These observations are demonstrated in a 2-dimensional phase diagram for a pure substance.



By Matthieumarechal, CC BY-SA 3.0

The green line shows the solid/liquid coexistence (the dashed green line showing the anomolous special case of water).

Triple Point

In [1]:

```
from IPython.display import YouTubeVideo
YouTubeVideo("BLRqpJN9zeA",560,315,rel=0)
```

Out[1]:

Critical Point

In [2]:

```
from IPython.display import YouTubeVideo
YouTubeVideo("-gCTKteN5Y4",560,315,rel=0)
```

Out[2]:

Saturation Vapor Pressure

At a given temperature, the **saturation vapor pressure** is the pressure of a vapor when in equilibrium with the liquid phase.

By itself, the term 'vapor pressure' may refer to a component of a gaseous mixture that is not in equilibrium with a liquid. Adding the extra adjective 'saturation' makes it clear that we're referring to a vapor-liquid equilibrium.

Clausius-Clapeyron Equation

The Clapeyron equation is a relationship between the saturation vapor pressure of a pure substance and absolute temperature T.

$$\frac{dP^{sat}}{dT} = \frac{\Delta\hat{H}_v}{T(\hat{V}_g - \hat{V}_l)}$$

where $\Delta\hat{H}_v$ is the latent heat of vaporization. This is the amount of heat needed to vaporize one mole of the substance in a constant pressure environment.

This equation is derived by comparing the change in free energy of the liquid phase compared to the free energy of the vapor phase. The changes must be equal for the two phases to remain in equilibrium which leads to this equation.

Suggested Study Exercise: *Compare the units on the left and right hand sides of this equation. Convince yourself they are the same.*

Under typical process conditions, the molar volume of vapor is much larger than the molar volume of liquid, i.e.,

$$\hat{V}_g \gg \hat{V}_l$$

In turn, the molar volume of the saturated vapor can be approximated by the ideal gas law

$$\hat{V}_g \approx \frac{RT}{P^{sat}}$$

Putting these together,

$$\frac{dP^{sat}}{dT} = \frac{P^{sat}\Delta\hat{H}_v}{RT^2}$$

Integrating,

$$\int \frac{1}{P^{sat}} dP^{sat} = \int \frac{\Delta\hat{H}_v}{RT^2} dT$$

Assuming $\Delta\hat{H}_v$ is independent of T (which is only an approximation) gives the **Clausius-Clapeyron** equation.

Doing an indefinite integration gives

$$\ln P^{sat} = K - \frac{\Delta\hat{H}_v}{RT}$$

where K is a constant of integration.

When integrating between specific bounds on P and T,

$$\int_{P_1}^{P_2} \frac{1}{P^{sat}}\, dP^{sat} = \int_{T_1}^{T_2} \frac{\Delta\hat{H}_v}{RT^2}\, dT$$

one obtains

$$\ln \frac{P_2^{sat}}{P_1^{sat}} = -\frac{\Delta\hat{H}_v}{RT}\left(\frac{1}{T_2} - \frac{1}{T_1}\right)$$

This equation provides a useful means of estimating heats of vaporization from measurements of vapor pressure.

Antoine's Equation

Vapor Pressure

Antoine's equation is used to estimate the saturation pressure (also called vapor pressure) of pure substances between the triple and critical points. Louis Charles Antoine, an engineer working in the French Navy, in 1886 published the equation as method for representing the vapor pressure of water.

A common form of the equation is

$$\log_{10} P^{sat}[mmHg] = A - \frac{B}{T[^\circ C] + C}$$

where pressure is units of millimeters of mercury (mmHg, also called torr), and temperature in degrees Celcius.

Compared to the Clausius-Clapeyron equation, this is an empirical expression with three parameters that can be adjusted to fit data for a variety of compounds. The algebraic structure is similar to the Clausius-Clapeyron equation, which gives considerable confidence that Antoine's equation should work well over reasonably wide temperature regimes.

Example: In the following cell, create a Python function to compute the vapor pressure of water in the range of 60 °C and higher. Verify by computing the vapor pressure of water at 100°C.

In [1]:

```
# water (data from Appendix B.4, Felder et al.)
A = 7.96681
B = 1668.21
C = 228.0

def Psat(T):
  return 10**(A - B/(T + C))

print('Vapor pressure of water at 100C =', Psat(100.0), 'mmHg')
```

```
Vapor pressure of water at 100C = 759.98304330841 mmHg
```

Saturation Temperature

An alternative form of the equation is to calculate the saturation temperature as a function of pressure

$$T^{sat}[^\circ C] = \frac{B}{A - \log_{10} P[mmHg]} - C$$

Values for the constants A, B, and C are tabulated in various references, including the NIST Chemistry Webbook. The values of the constants depend on the units used for pressure and temperature, and whether the logarithm is computed for base e or base 10.

Example: On a given day, the air pressure at the Denver airport is 623 mmHg. What is the boiling point of water at that pressure?

In [4]:

```
from math import log10

A = 7.96681
B = 1668.21
C = 228.0

def Tsat(P):
    return B/(A - log10(P)) - C

print('Boiling point of water at 623mmHg =', Tsat(623.0), 'deg C')
```

```
Boiling point of water at 623mmHg = 94.52632667664744 deg C
```

Working with Wide Temperature Ranges

Standard practice is to specify a range of temperatures over which a particular set of constants is known to offer an accurate representation. Multiple ranges may be pieced together to obtain saturation pressure over wider ranges.

Example: Create a python function `Psat` to compute the vapor pressure of water from the triple point to the critical point, and use it to create a corresponding plot.

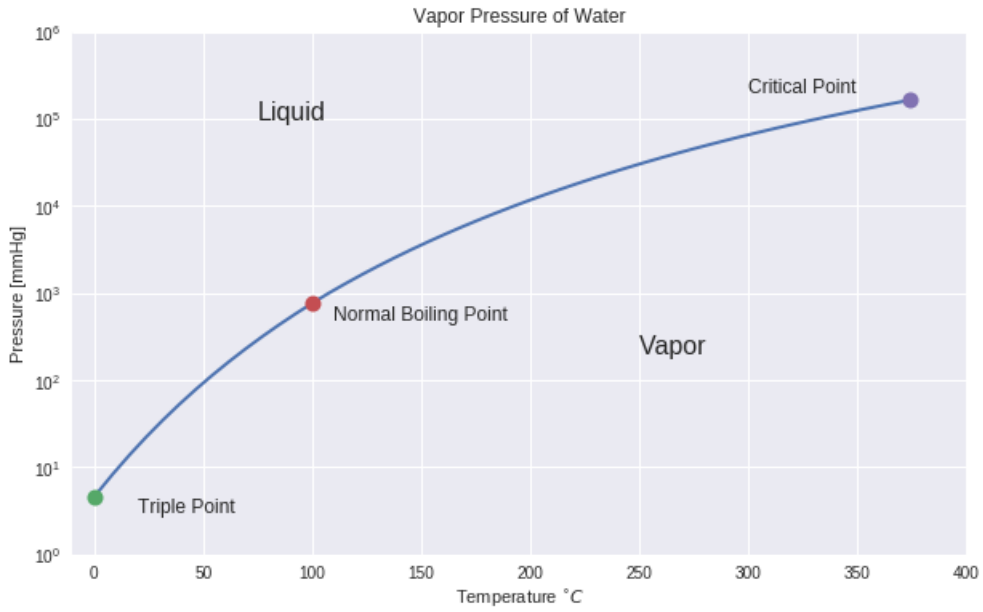
In [ ]:

```
# Antoine's equation for water from 1 to 374 degrees C

def Psat(T):
  if (1 <= T < 100):
    return 10**(8.07131 - 1730.63/(T + 233.426))
  elif (100 <= T <= 374):
    return 10**(8.14019 - 1810.94/(T + 244.485))
  else:
    return float('nan')
```

The following code uses the `numpy` and `matplotlib.pyplot` libraries to create the requested plot.

In [12]:

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

# Use Psat(T) to construct to show the vapor-liquid equilibrium diagram
T = np.linspace(1,374)
plt.figure(figsize = (10,6))
plt.semilogy(T,[Psat(T) for T in T],linewidth=2)
plt.xlabel('Temperature $^{\circ}C$')
plt.ylabel('Pressure [mmHg]')
plt.title('Vapor Pressure of Water')

# Additional annotations
plt.semilogy(0.01,4.58,'o',markersize=10)
plt.annotate('Triple Point', xy=(10,4.58), xytext=(20,3))
plt.semilogy(100,760,'o',markersize=10)
plt.annotate('Normal Boiling Point', xy=(100,760), xytext=(110,500))
plt.semilogy(374,1.67e5,'o',markersize=10)
plt.annotate('Critical Point', xy=(374,1.67e5), xytext=(300,2e5))
plt.text(250,200,'Vapor',fontsize=16)
plt.text(75,100000,'Liquid',fontsize=16)
plt.ylim([1,1000000])
plt.xlim([-10,400]);
```



Example: The catapults on aircraft carriers require steam at 520 psig. What is the minimum operating temperature?

In [17]:

```
# convert pressure to absolute mmHg
P = 520             # psig
P = 520 + 14.696    # convert psig -> psia
P = P*760/14.696    # convert psig -> mmHg

# function to solve
def f(T):
  return Psat(T) - P

# import a root-finding algorithm, provide initial guess
from scipy.optimize import fsolve
T = fsolve(f,200)

print("Operating Pressure =", P, "mmHg")
print("Minimum Operating Temperature =", T, "deg C")
```

```
Operating Pressure = 27651.671203048452 mmHg
Minimum Operating Temperature = [245.16090162] deg C
```

Normal Boiling Points

The **normal boiling point** of a pure component is the temperature at which the saturation vapor pressure is equal to one atmosphere. In other words, it is the saturation temperature corresponding to a pressure of one atmosphere.

Not every species will have a normal boiling point. For example, if the pressure at the triple point is above one atmosphere, then at one atmosphere the species will sublime directly from the solid phase to vapor phase without boiling. Carbon dioxide is one example of such a compound.

Example: Propylene glycol is a commonly used heat transfer fluid for industrial applications. What is the normal boiling point of propylene glycol?

Solution: The Antoine coefficients are available on the NIST Webbook entry for propylene glycol. NIST, however, reports constants for the case where temperature is in Kelvin (K) and pressure in bar.

In [26]:

```
# propylene glycol (pressure in bar, temperature in K)

A = 6.07936
B = 2692.187
C = -17.94

from math import log10

def Tsat(P):
  P = 1.01325*P/760          # convert to pressure from mmHg
  T = B/(A - log10(P)) - C   # compute temperature in K
  T = T - 273.15             # convert temperature from K to C
  return T

print('Normal boiling point of propylene glycol =', Tsat(760), 'deg C')
```

```
Normal boiling point of propylene glycol = 188.04733800941904 deg C
```

Example: What is the saturation pressure of water at the normal boiling point of propylene glycol. Why do you think it might be preferred to water?

Solution: The advantage of propylene glycol as a heat transfer fluid is that it can be used for transfering heat in the range from 100C to 188C without the need for handling fluids under pressure.

In [27]:

```
# Antoine's equation for water from 1 to 374 degrees C

def Psat(T):
  if (1 <= T < 100):
    return 10**(8.07131 - 1730.63/(T + 233.426))
  elif (100 <= T <= 374):
    return 10**(8.14019 - 1810.94/(T + 244.485))
  else:
    return float('nan')

P = 14.696*(Psat(188.05)/760 - 1)

print('Saturation pressure of water at 188.05C = ', P, 'psig')
```

```
Saturation pressure of water at 188.05C =  158.99201785868928 psig
```

7.3 Operating Limits for a Methanol Lighter

Summary

Demonstrates the use Antoine's equation and Raoult's law to compute flammability limits for methanol.

Problem

We'd like to estimate range of temperatures over which this methanol fueled fire starter will successfully operate.

In [1]:

```
from IPython.display import YouTubeVideo
YouTubeVideo("8gFqzbnUO-Y")
```

Out[1]:

The flammability limits of methanol in air at 1 atmosphere pressure correspond to vapor phase mole fractions in the range

$$6.7 \text{ mol}\% \leq y_{MeOH} \leq 36 \text{ mol}\%$$

Assuming the pure methanol located in the wick of this fire starter reaches a vapor-liquid with air in the device, find the lower and upper operating temperatures for this device.

Antoine's Equation for the Saturation Pressure of Methanol

The first thing we'll do is define a simple python function to calculate the saturation pressure of methanol at a given temperature using Antoine's equation

$$\log_{10} P^{sat} = A - \frac{B}{T + C}$$

Constants for methanol can be found in the back of the course textbook for the case where pressure is given in units of mmHg and temperature in degrees centigrade.

In [ ]:

```
# Antoine equation for methanol. Pressure in mmHg, temperature in degrees C
A = 7.89750
B = 1474.08
C = 229.13

def Psat(T):
    return 10**(A - B/(T+C))
```

To test the function, compute the saturation pressure of methanol for several temperatures and print the results.

In [3]:

```
T = [0, 5, 10, 15, 20, 25]

for t in T:
  print('Saturation pressure of methanol at', t, 'deg C =', Psat(t), 'mmHg')
```

```
Saturation pressure of methanol at 0 deg C = 29.11532005615582 mmHg
Saturation pressure of methanol at 5 deg C = 39.94943048219959 mmHg
Saturation pressure of methanol at 10 deg C = 54.09464146182495 mmHg
Saturation pressure of methanol at 15 deg C = 72.3445037835956 mmHg
Saturation pressure of methanol at 20 deg C = 95.6288978068613 mmHg
Saturation pressure of methanol at 25 deg C = 125.02710947768249 mmHg
```

Equilibrium Vapor Composition at Room Temperature

By Raoult's law, the partial pressure of pure methanol is equal to the saturation pressure,

$$p_{MeOH} = P^{sat}_{MeOH}(T)$$

For an ideal gas, the partial pressure is given by Dalton's law

$$p_{MeOH} = y_{MeOH} P$$

Putting these together and solving for the mole fraction of methanol in the vapor phase

$$y_{MeOH} = \frac{P^{sat}_{MeOH}(T)}{P}$$

For example, at an atmospheric pressure of 760 mmHg, the mole fraction of methanol is given by

In [4]:

```
P = 760.0             # mmHg
y = Psat(25)/P        # mole fraction

print("mole fraction of methanol at 25 deg C and 1 atmosphere =", y)
```

```
mole fraction of methanol at 25 deg C and 1 atmosphere = 0.16450935457589802
```

Since this is within the flammability limits of methanol, the fire starter should work at room temperatures.

Lower Operating Temperature Limit

The partial pressure of methanol at the lower flammability limit

$$p_{MeOH} = y_{MeOH} P$$

In [5]:

```
P = 760.0           # pressure mmHg
y = 0.067           # mole fractio at the lower flammability limit

p_MeOH = 0.067*760
print('model fraction at the LFL =', p_MeOH, "mm Hg")
```

```
model fraction at the LFL = 50.92 mm Hg
```

Next we need to solve for the temperature at which the partial pressure of methanol is at the lower flammability limit.

$$P^{sat}_{MeOH}(T) = p_{MeOH}$$

Let's start with a guesses at 0 and 20 deg C.

In [6]:

```
print(Psat(0), "mmHg")
print(Psat(20), "mmHg")
```

```
29.11532005615582 mmHg
95.6288978068613 mmHg
```

This may be close enough to use an equation solver to finish the job. Here we'll use `brentq` which is one of the root-finding funtions from the `scipy.optimize` library.

A python `lambda` function is a convenient means of recasting the equation

$$P^{sat}_{MeOH} = y_{MeOH} P$$

as

$$P^{sat}_{MeOH} - y_{MeOH} P = 0$$

which is the form need for using a root-finding algorithm.

In [7]:

```
# problem data
P = 760.0           # pressure mmHg
y = 0.067           # mole fractio at the lower flammability limit

# partial pressure of methanol at the lower flamability limit
p_MeOH = 0.067*760

# Antoine equation for methanol. Pressure in mmHg, temperature in degrees C
def Psat(T):
    return 10**(7.89750 - 1474.08/(T + 229.13))

from scipy.optimize import brentq

Tlow = brentq(lambda T: Psat(T) - p_MeOH, 0, 20)
print("Lower Flammability Temperature = ", Tlow, "deg C")
print("Upper Flammability Temperature = ", 32.0 + 9.0*Tlow/5.0, "deg F")
```

```
Lower Flammability Temperature =  8.985406691811518 deg C
Upper Flammability Temperature =  48.173732045260735 deg F
```

Exercise

Calculate the upper limit on temperature at which this lighter can operate.

7.4 Raoult Law for Ideal Mixtures

Summary

This notebook illustrates the use of Raoult's Law to calculate vapor pressure, and compares the results to experimental data for a non-ideal system. The video is used with permission from learnCheme.com, a project at the University of Colorado funded by the National Science Foundation and the Shell Corporation.

Introduction

Thermally based chemical separations, such as distillation and flash units, account for about 10 to 15 percent of the world's energy use. Raoult's law provides an idealized but nonetheless insightful understanding of how the vapor-liquid equilibrieum of mixtures is exploited for industrial separations.

Dalton's law, in turn, says the total pressure P is equal to the sum of the partial pressures, i.e.,

$$P = \sum_{n=1}^{N} p_n$$

Raoult's law says the partial pressure p_n of each component in a mixture of liquids is equal to the product of the mole fraction x_n and the saturation pressure $P_n^{sat}(T)$ of the pure component. That is,

$$p_n = x_n P_n^{sat}(T)$$

For an **ideal gas**, the partial pressure of an component in a mixture of gases is equal to the mole fraction y_n and total pressure P

$$p_n = y_n P$$

Subject to the assumptions of ideal liquid and gas mixtures, these three equations can be combined to provide a useful theory for vapor-liquid equilibrium and separations of ideal mixtures.

Vapor Pressure of Pure Components

The calculations in this notebook are for a representative system of two components, acetone and ethanol for which experimental data is available from the Dortmund Data Bank.

We start by creating two functions to estimate vapor pressure for the individual species using Antoine's equation.

In [ ]:

```
# Antoine's equations
A = 'acetone'
B = 'ethanol'

def PsatA(T):
    return 10**(7.02447 - 1161.0/(T + 224))

def PsatB(T):
    return 10**(8.04494 - 1554.3/(T + 222.65))
```

In particular, let's compute the saturation pressure at 32 °C can compare to experimental data.

In [6]:

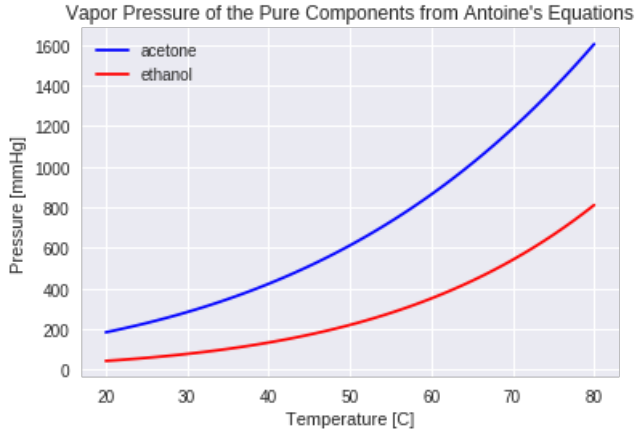
```
T = 32.0   # deg C

print('Saturation Pressure of', A, 'at', T, ' deg C =', round(PsatA(T),2), 'mmHg')
print('Saturation Pressure of', A, 'at', T, ' deg C =', round(PsatA(T)*101.325/760,2),
'kPa')

print()
print('Saturation Pressure of', B, 'at', T, ' deg C =', round(PsatB(T),2), 'mmHg')
print('Saturation Pressure of', B, 'at', T, ' deg C =', round(PsatB(T)*101.325/760,2),
'kPa')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-66e36a990288> in <module>()
      1 T = 32.0   # deg C
      2
----> 3 print('Saturation Pressure of', A, 'at', T, ' deg C =', round(PsatA(T),2),
'mmHg')
      4 print('Saturation Pressure of', A, 'at', T, ' deg C =', round(PsatA(T)*101.325/
760,2), 'kPa')
      5

NameError: name 'A' is not defined
```

Create plots for the vapor pressure of each component as a function of temperature.

In [4]:

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

# Plot pure component vapor pressures
T = np.linspace(20,80)

plt.plot(T,PsatA(T),'b')
plt.plot(T,PsatB(T),'r')

plt.ylabel('Pressure [mmHg]')
plt.xlabel('Temperature [C]')
plt.legend([A,B],loc='best')
plt.title("Vapor Pressure of the Pure Components from Antoine's Equations");
```



Things to Do / Things to Think About:

1. What is the more volatile species?
2. Using this chart, estimate the normal boiling points of acetone and ethanol.
3. In the cell below, compute the normal boiling points using the function `brentq` from `scipy.optimize` library. Compare your calculations to experimental data from a source that can trace back to the peer-reviewed literature.

In [5]:

```
from scipy.optimize import brentq

Patm = 760   # mmHg
Tlow = 20    # deg C
Thigh = 80   # deg C

TboilA = brentq(lambda T: PsatA(T) - Patm, Tlow, Thigh)
print('Normal boiling point of', A, '=', round(TboilA,2), 'deg C')
print('Normal boiling point of', A, '=', round(TboilA + 273.15,2), 'deg K')

print()
TboilB = brentq(lambda T: PsatB(T) - Patm, Tlow, Thigh)
print('Normal boiling point of', B, '=', round(TboilB,2), 'deg C')
print('Normal boiling point of', B, '=', round(TboilB + 273.15,2), 'deg K')
```

```
Normal boiling point of acetone = 56.19 deg C
Normal boiling point of acetone = 329.34 deg K

Normal boiling point of ethanol = 78.33 deg C
Normal boiling point of ethanol = 351.48 deg K
```

Boiling point data from the National Institute of Standards and Technology:

- NIST acetone
- NIST ethanol

Vapor Pressure of an Acetone/Ethanol Mixture at a Fixed Temperature

For an ideal gas, the total vapor pressure of the liquid is equal to the sum of partial pressures,

$$P = \sum_{n=1}^{N} x_n P_n^{sat}(T)$$

The is explained in further detail in the following video for a two component mixture.

In [6]:

```
from IPython.display import YouTubeVideo
YouTubeVideo('Adr9_2LnQdw')
```

Out[6]:

In this example we estimate the vapor pressure of an acetone/ethanol mixture using Raoult's law, assuming the vapor pressure of the component species are given by Antoine's equation. The estimate is compared to experimentally measured data.

For a binary mixture of A and B, the total vapor pressure is a function of T and x_A.

$$P = x_A P_A^{sat}(T) + (1 - x_A) P_B^{sat}(T)$$

The following cell shows a plot of the total vapor pressure P as a function of x_A for the case of an acetone (A) and ethanol (B) mixture. The component partial pressures

$$\begin{aligned} p_A &= x_A P_A^{sat}(T) \\ p_B &= (1 - x_A) P_B^{sat}(T) \end{aligned}$$

are also shown. First run the cell manually. The cell should run automatically when the slider is adjusted.

In [5]:

```
#@title Use the slider to adjust temperature (deg C) {run: "auto"}
T = 61 #@param {type:"slider", min:20, max:80, step:1}

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0,1)

plt.figure(figsize=(8,6))
plt.plot(x, [x*PsatA(T) + (1-x)*PsatB(T) for x in x],'b')
plt.plot(x, [x*PsatA(T) for x in x],'r--')
plt.plot(x, [(1-x)*PsatB(T) for x in x],'g--')
plt.plot(0, PsatB(T), 'g.', ms=20)
plt.plot(1, PsatA(T), 'r.', ms=20)

plt.xlim(-0.01,1.01)
plt.ylim(0, 1200)
plt.xlabel('$x_A$ = Mole fraction ' + A, fontsize = 14)
plt.ylabel('Vapor Pressure / mmHg', fontsize = 14)
plt.title('Raoult\'s Law: ' + A + ' / ' + B + ' at {:.1f} deg C'.format(T),
          fontsize = 16)
plt.legend(['Total Pressure: Raoult\'s Law','Vapor Pressure of Acetone',
            'Vapor Pressure of Ethanol']);
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-f49ffa510a60> in <module>()
      7
      8 plt.figure(figsize=(8,6))
----> 9 plt.plot(x, [x*PsatA(T) + (1-x)*PsatB(T) for x in x],'b')
     10 plt.plot(x, [x*PsatA(T) for x in x],'r--')
     11 plt.plot(x, [(1-x)*PsatB(T) for x in x],'g--')

<ipython-input-5-f49ffa510a60> in <listcomp>(.0)
      7
      8 plt.figure(figsize=(8,6))
----> 9 plt.plot(x, [x*PsatA(T) + (1-x)*PsatB(T) for x in x],'b')
     10 plt.plot(x, [x*PsatA(T) for x in x],'r--')
     11 plt.plot(x, [(1-x)*PsatB(T) for x in x],'g--')

NameError: name 'PsatA' is not defined
```

```
<matplotlib.figure.Figure at 0x7fbc7d7a0be0>
```

Things to Do / Things to Think About:

1. What is vapor pressure of a liquid mixture that is 50% ethanol at 32 deg C?
2. Note the point where the ethanol and acetone partial pressures intersect. What is the composition of the vapor phase at this point? What is the liquid phase composition? Why are they different?
3. Assuming you start with a cold liquid mixture that is 50% ethanol. By adjusting the temperature, estimate the point at which it starts boiling.

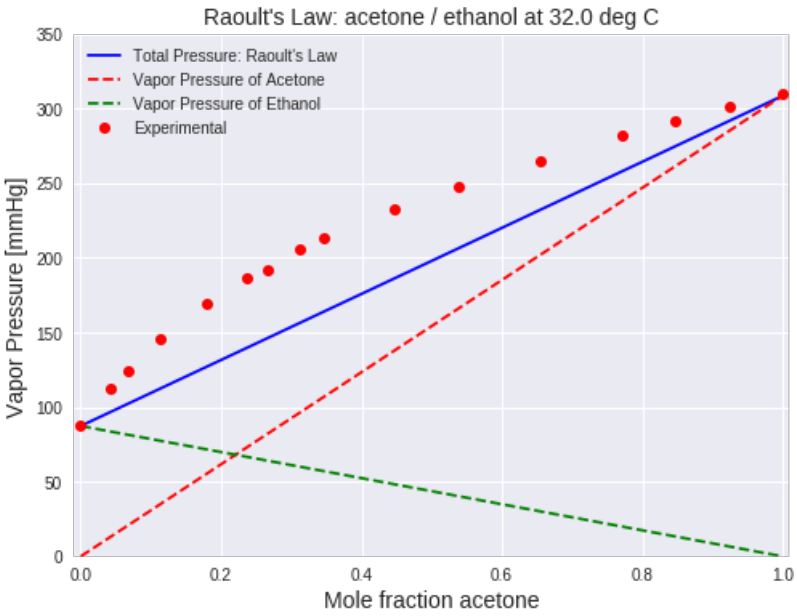
estimate the normal boiling point of a 50% liquid mixture of ethanol. Estimate the vapor phase composition at the normal boiling point.

Comparing Raoult's Law to Experimental Data

Experimental Data

In [8]:

```
T = 32  # deg C

plt.figure(figsize=(8,6))
plt.plot(x, [x*PsatA(T) + (1-x)*PsatB(T) for x in x],'b')
plt.plot(x, [x*PsatA(T) for x in x],'r--')
plt.plot(x, [(1-x)*PsatB(T) for x in x],'g--')

# Experimental data of (P,x) observations
Px = np.array([\
    [11.679, 0.00000],\
    [14.999, 0.04220],\
    [16.585, 0.06730],\
    [19.358, 0.11300],\
    [22.571, 0.17870],\
    [24.811, 0.23610],\
    [25.585, 0.26650],\
    [27.384, 0.31280],\
    [28.371, 0.34700],\
    [31.037, 0.44580],\
    [33.037, 0.53720],\
    [35.370, 0.65480],\
    [37.584, 0.77210],\
    [38.890, 0.84740],\
    [40.130, 0.92520],\
    [41.317, 1.00000]])

# Convert kPa to mmHg
P = Px.T[0]*760/101.3

# Convert K to C
T = 305.15 - 273.15

# Extract measured composition
x = Px.T[1]

# Overlay plot of experimental data and label$$
plt.plot(x,P,'ro')
plt.xlim(-0.01, 1.01)
plt.ylim(0, 350)
plt.xlabel('Mole fraction ' + A, fontsize = 14)
plt.ylabel('Vapor Pressure [mmHg]', fontsize = 14)
plt.title('Raoult\'s Law: ' + A + ' / ' + B + ' at {:.1f} deg C'.format(T),
          fontsize = 14)
plt.legend(['Total Pressure: Raoult\'s Law',
            'Vapor Pressure of Acetone',
            'Vapor Pressure of Ethanol',
            'Experimental']);
```

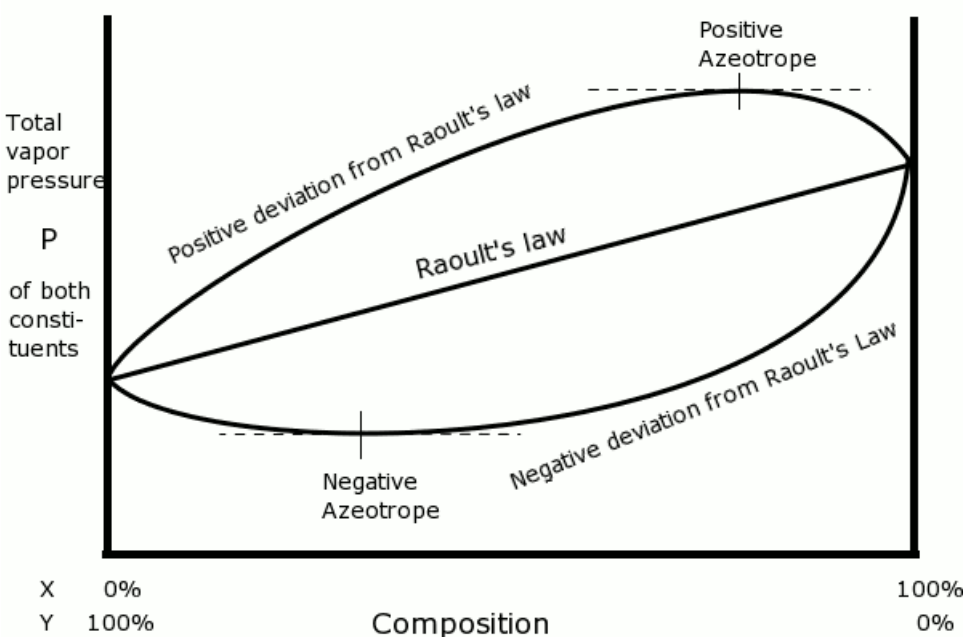


Things to Do / Things to Think About:

1. Does a real liquid mixture of ethanol and acetone boil at a higher or lower temperature than predicted by Raoult's law?

Deviations from Raoult's Law

Raoult's law works for ideal mixtures and ideal gases. Given a prediction from Raoult's law, the actual vapor pressure may be higher (**a positive deviation**) or lower (**a negative deviation**). If the deviations are sufficiently large then an **azeotrope** may form.



Things to Do / Things to Think About:

1. In this sketch, what is the more volatile species?
2. Sketch in the vapor phase partial pressures corresponding to Raoult's law. How do you expect these to change for a mixture with a negative deviation from Raoult's law?
3. Does a negative deviation azeotrope boil at a higher or lower temperature than a mixture that follows Raoult's law?

In [ ]:

7.5 Henry Law Constants

Henry's Law Calculator

Run both of the following cells before using the calculator.

In [ ]:

```
!pip install -q iapws
```

In [11]:

```
#@title Henry's Law Constant in Various Units { run: "auto", vertical-output: true }
Gas = "CO2" #@param ["Ar", "CH4", "C2H6", "CO", "CO2", "H2", "H2S", "He", "Kr", "N2",
"Ne", "O2", "SF6", "Xe"]
T = 25 #@param {type:"number"}

from iapws._iapws import _Henry as Henry

P_henry = Henry(T + 273.15, Gas, liquid="H2O")
print("Henry's Law Constant:")
print("    [MPa] =", round(P_henry,2))
print("    [bar] =", round(P_henry*10, 1))
print("    [atm] =", round(P_henry/0.101325, 1))
print("   [psia] =", round(P_henry*14.696/0.101325,0))
```

```
Henry's Law Constant:
    [MPa] = 165.64
    [bar] = 1656.4
    [atm] = 1634.8
   [psia] = 24025.0
```

In [ ]:

```
```

7.6 Binary Phase Diagrams for Ideal Mixtures

Summary

This notebook shows how to use Raoult's Law and Antoine's equations to calculate Pxy, Txy, and xy diagrams for binary mixtures. 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.

Introduction

For binary mixtures, Raoult's law provides a very useful equilibrium relationship between pressure, temperature, and composition of liquid and vapor phases. This relationship can be expressed as a set of equations, or graphically as

- **Pxy Diagram** relationship of pressure, liquid, and vapor composition at fixed temperature.
- **Txy Diagram** relationship of temperature, liquid, and vapor composition at fixed pressure.
- **xy Diagram** relationhip between liquid and vapor phase composition, usually plotted at fixed pressure.

The following video from LearnChemE gives a brief introduction.

In [57]:

```
from IPython.display import YouTubeVideo
YouTubeVideo('E_Vuz8cfbEo')
```

Out[57]:

Antoine's Equation

The calculations in this notebook use Antoine's equation to compute the saturation vapor pressure for a given temperature, and solves Antoine's equation for the saturation temperature for a given pressure.

For this purpose, we create a simple Python class to store data for a set of species, and to provide functions to compute saturation pressures and temperatures. Python class is a more advanced aspect of Python programming that has been deliberately avoided in most of the notebooks in this respository. In this particular instance, however, the use of a class significantly streamlines the notebook.

The thermo library is a complete implementation of a chemical property dataset in Python.

In [ ]:

```
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 the following cell we choose specific species to use as components `A` and `B` in subsequent calculations for thsi notebook.

In [59]:

```
A = acetone
B = ethanol

# report normal boiling points
for s in [A,B]:
  print('Normal boiling point of', s.name, ':', round(s.Tsat(760),1), 'deg C')
```

```
Normal boiling point of acetone : 56.2 deg C
Normal boiling point of ethanol : 78.3 deg C
```

Binary Mixtures

For an ideal **binary mixture** of A and B, the vapor phase pressure P is the sum of component partial pressures p_A and p_B

$$P = p_A + p_B$$

Raoult's law, in turn, says for ideal mixtures

$$p_A = x_A P_A^{sat}(T)$$
$$p_B = x_B P_B^{sat}(T)$$

Eliminate partial pressures and obtain an expression for total vapor pressure.

$$P = \underbrace{x_A P_A^{sat}(T)}_{p_A = y_A P} + \underbrace{x_B P_A^{sat}(T)}_{p_B = y_B P}$$

For binary mixtures, the substitutions $x_B = 1 - x_A$ and $y_B = 1 - y_A$ give an expression for total pressure as a function of composition x_A and temperature.

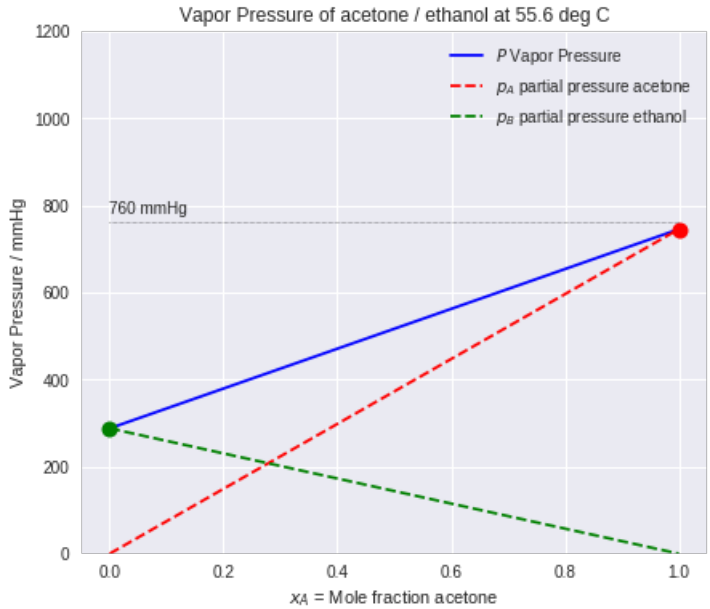
$$P = \underbrace{x_A P_A^{sat}(T)}_{p_A = y_A P} + \underbrace{(1 - x_A) P_A^{sat}(T)}_{p_B = (1 - y_B) P}$$

In the Raoult's Law notebook, we demonstrated this relationship by plotting P, p_A, and p_B as functions of x_A at a fixed temperature.

The next step is to select a temperature T and run the following cell. This cell will evaluate and display P, p_A and p_B functions of liquid phase mole fraction x_A.

In [60]:

```
#@title Raoult's Law { run: "auto", vertical-output: true, form-width: "500px"}
#@markdown Adjust the slider to set the liquid temperature.
T = 55.6 #@param {type:"slider", min:30, max:80, step:0.2}

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

# compute partial pressures and total pressure
xA = np.linspace(0,1)
pA = xA*A.Psat(T)
pB = (1-xA)*B.Psat(T)
Pv = pA + pB

# create plot
plt.figure(figsize=(7,6))
plt.plot(xA, Pv,'b')
plt.plot(xA, pA,'r--')
plt.plot(xA, pB,'g--')

# mark pure component saturation pressures
plt.plot(0, B.Psat(T), 'g.', ms=20)
plt.plot(1, A.Psat(T), 'r.', ms=20)

# annotate the plot
plt.plot([0,1],[760,760],'k:', lw=0.5)
plt.text(0, 780, '760 mmHg')
plt.ylim(0, 1200)
plt.xlabel('$x_A$ = Mole fraction ' + A.name)
plt.ylabel('Vapor Pressure / mmHg')
plt.title('Vapor Pressure of '+A.name+' / '+B.name+
          ' at {:.1f} deg C'.format(T))
plt.legend(['$P$ Vapor Pressure',
            '$p_A$ partial pressure ' + A.name,
            '$p_B$ partial pressure ' + B.name])
plt.show()
```



Exercises

1. Use the temperature slider to determine the approximate boiling point of pure acetone.
2. What is the approximate boiling point of an acetone/ethanol mixture that is 35 mole% acetone?
3. What is the corresponding vapor phase composition?

The dashed lines denote the partial presssures of A and B. This information can be used compute the vapor phase composition

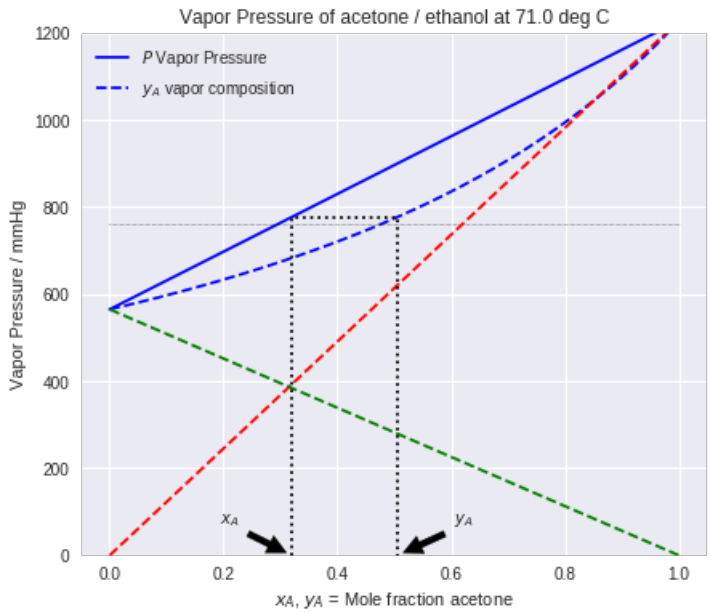
$$y_A = \frac{p_A}{p_A + p_B} = \frac{p_A}{P}$$

This information can be added to the plot by computing y_A and then plotting coordinate pairs (y_A, P).

To read this diagram, start with a value of the liquid phase composition, x_A, look up to find the vapor pressure, then look right to find the vapor composition in equilibrium with the liquid phase.

In [61]:

```
#@title Vapor Phase Composition { run: "auto", vertical-output: true }
T = 71 #@param {type:"slider", min:30, max:80, step:1}
xA = 0.32 #@param {type:"slider", min:0, max:1, step:0.01}
show_partial_pressures = True #@param {type:"boolean"}
show_tie_line = True #@param {type:"boolean"}

xA_save = xA

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

# compute partial pressures and total pressure
xA = np.linspace(0,1)
pA = xA*A.Psat(T)
pB = (1-xA)*B.Psat(T)
Pv = pA + pB
yA = pA/Pv

# create plot
plt.figure(figsize=(7,6))
plt.plot(xA, Pv, 'b')
plt.plot(yA, Pv, 'b--')

# annotate the plot
plt.plot([0,1], [760,760], 'k:', lw=0.5)
plt.ylim(0, 1200)
plt.xlabel('$x_A$, $y_A$ = Mole fraction ' + A.name)
plt.ylabel('Vapor Pressure / mmHg')
plt.title('Vapor Pressure of '+A.name+' / '+B.name+
          ' at {:.1f} deg C'.format(T))
plt.legend(['$P$ Vapor Pressure',
            '$y_A$ vapor composition'])

if show_partial_pressures:
    plt.plot(xA, pA, 'r--')
    plt.plot(xA, pB, 'g--')

# show how to use
if show_tie_line:
    xA = xA_save
    pA = xA*A.Psat(T)      # partial pressure A
    pB = (1-xA)*B.Psat(T) # partial pressure B
    Pv = pA + pB           # vapor pressure
    yA = pA/Pv             # vapor phase composition            )
    plt.plot([xA, xA, yA, yA], [0, Pv, Pv, 0], 'k:')
    dx = -40 if xA < yA else 50
    plt.annotate('$x_A$', xy=(xA,0), xycoords='data',
             xytext=(dx-5,20), textcoords='offset pixels',
             arrowprops=dict(facecolor='black', shrink=0.1))
    plt.annotate('$y_A$', xy=(yA,0), xycoords='data',
             xytext=(-dx-5,20), textcoords='offset pixels',
             arrowprops=dict(facecolor='black', shrink=0.1))

plt.show()
```



Exercises

1. You're in a warehouse and come across a sealed tank labeled 60% acetone in ethanol. It's 30 degrees C in the warehouse. What is the pressure in the tank?
2. Suppose you have a liquid acetone/ethanol mixture that is 50 mole% acetone. At a pressure of 1 atm, at what temperature does the mixture boil? What is the vapor phase composition?
3. A chemical process produces an acetone/ethanol vapor stream that is 50 mole% acetone. At what temperature does it begin to condense?

Pxy and Txy Diagrams for Binary Mixtures

For a **binary mixture** of A and B, Raoult's law reads

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

Substituting $x_B = 1 - x_A$

$$P = x_A P_A^{sat}(T) + (1 - x_A) P_A^{sat}(T)$$

Solving for x_A

$$x_A = \frac{P - P_B^{sat}(T)}{P_A^{sat}(T) - P_B^{sat}(T)}$$

This equation provides an alternative method of creating the diagrams shown above. fixing temperature T, the above equation is a linear function of P that goes through the points $(x_A, P) = (0, P_B^{sat}(T))$ and $(x_A, P) = (1, P_A^{sat}(T))$.

From Raolt's law we know

$$y_A P = x_A P_A^{sat}(T) \implies y_A = x_A \frac{P_A^{sat}(T)}{P}$$

Putting these all together

$$\begin{aligned} x_A &= \frac{P - P_B^{sat}(T)}{P_A^{sat}(T) - P_B^{sat}(T)} \\ y_A &= \frac{P - P_B^{sat}(T)}{P_A^{sat}(T) - P_B^{sat}(T)} \frac{P_A^{sat}(T)}{P} \end{aligned}$$

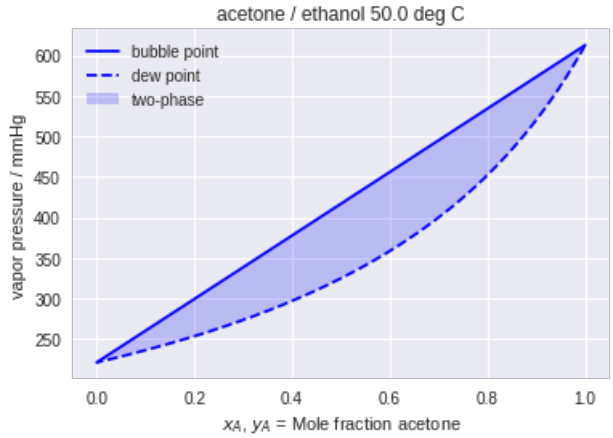
This gives the liquid phase and vapor phase compositions, x_A and y_A respectively, in terms pressure and temperature. This is very useful for the analysis of separation processes. From these equations we can construct three plots:

- **Pxy** Fix T. Plot x_A and y_A as functions of P in the range $P_B^{sat}(T)$ to $P_A^{sat}(T)$.
- **Txy** Fix P. Plot x_A and y_A as functions of T in the range $T_B^{sat}(P)$ to $T_A^{sat}(P)$.
- **xy** Fix either T or P. Plot y_A as a function of x_A for T in the range $T_B^{sat}(P)$ to $T_A^{sat}(P)$ or P in the range $P_B^{sat}(T)$ to $P_A^{sat}(T)$.

The next cells demonstrate each of these plots.

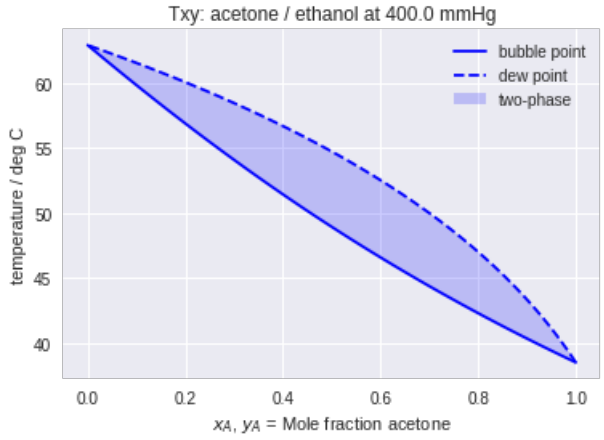
In [111]:

```
#@title Pxy Diagram { run: "auto", vertical-output: true }
T = 50 #@param {type:"slider", min:30, max:80, step:1}

import matplotlib.pyplot as plt
import numpy as np

def Pxy(A, B, T=0): # , z=0, P=760):
    Pp = np.linspace(A.Psat(T), B.Psat(T))
    xA = (Pp - B.Psat(T))/(A.Psat(T) - B.Psat(T))
    yA = xA*A.Psat(T)/Pp
    plt.plot(xA, Pp, 'b')
    plt.plot(yA, Pp, 'b--')
    plt.gca().fill_betweenx(Pp, xA, yA, facecolor='blue', alpha=0.2)
    plt.xlabel('$x_A$, $y_A$ = Mole fraction ' + A.name)
    plt.ylabel('vapor pressure / mmHg')
    plt.title(A.name + ' / ' + B.name + ' {:.1f} deg C'.format(T))
    plt.legend(['bubble point', 'dew point', 'two-phase'])

Pxy(A, B, T)
```



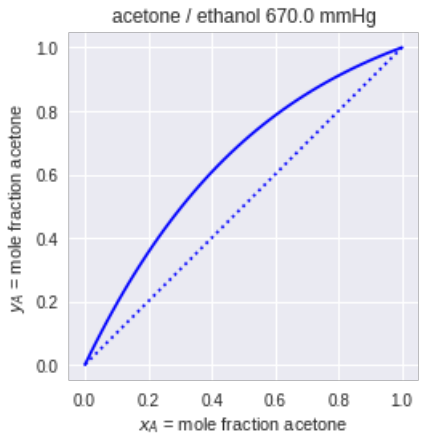
In [109]:

```
#@title Txy Diagram { run: "auto", vertical-output: true }
P = 400 #@param {type:"slider", min:200, max:1200, step:10}

import matplotlib.pyplot as plt
import numpy as np

def Txy(A, B, P=760):  #, z=0, T=0):
    Tp = np.linspace(A.Tsat(P), B.Tsat(P))
    xA = (P - B.Psat(Tp))/(A.Psat(Tp) - B.Psat(Tp))
    yA = xA*A.Psat(Tp)/P
    plt.plot(xA, Tp, 'b')
    plt.plot(yA, Tp, 'b--')
    plt.gca().fill_betweenx(Tp, xA, yA, facecolor='blue', alpha=0.2)
    plt.xlabel('$x_A$, $y_A$ = Mole fraction ' + A.name)
    plt.ylabel('temperature / deg C')
    plt.title('Txy: '+A.name+' / '+B.name+' at {:.1f} mmHg'.format(P))
    plt.legend(['bubble point', 'dew point', 'two-phase'])

Txy(A, B, P)
```



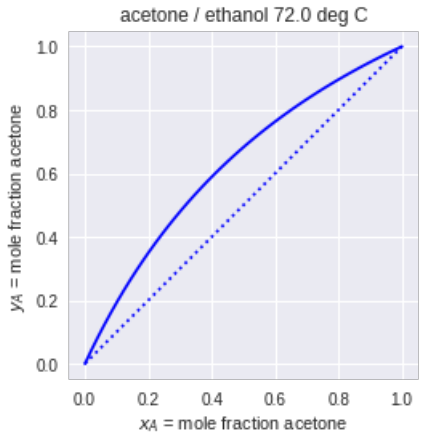
In [122]:

```
#@title xy Diagram for a given pressure { run: "auto", vertical-output: true }
P = 670 #@param {type:"slider", min:200, max:1200, step:10}

import matplotlib.pyplot as plt
import numpy as np

def xy(A, B, P=None, T=None):  #, z=0, T=0):
  title = A.name + ' / ' + B.name
  if P is not None:
    T = np.linspace(A.Tsat(P), B.Tsat(P))
    title = title + ' {:.1f} mmHg'.format(P)
  elif T is not None:
    P = np.linspace(A.Psat(T), B.Psat(T))
    title = title + ' {:.1f} deg C'.format(T)
  xA = (P - B.Psat(T))/(A.Psat(T) - B.Psat(T))
  yA = xA*A.Psat(T)/P
  plt.plot(xA, yA, 'b')
  plt.plot([0,1],[0,1],'b:')
  plt.gca().set_aspect('equal','box')
  plt.xlabel('$x_A$ = mole fraction ' + A.name)
  plt.ylabel('$y_A$ = mole fraction ' + A.name)
  plt.title(title)

xy(A, B, P=P)
```



In [123]:

```
#@title xy Diagram for a given temperature { run: "auto", vertical-output: true }
T = 72 #@param {type:"slider", min:20, max:80, step:1}


xy(A, B, T=T)
```



Exercises

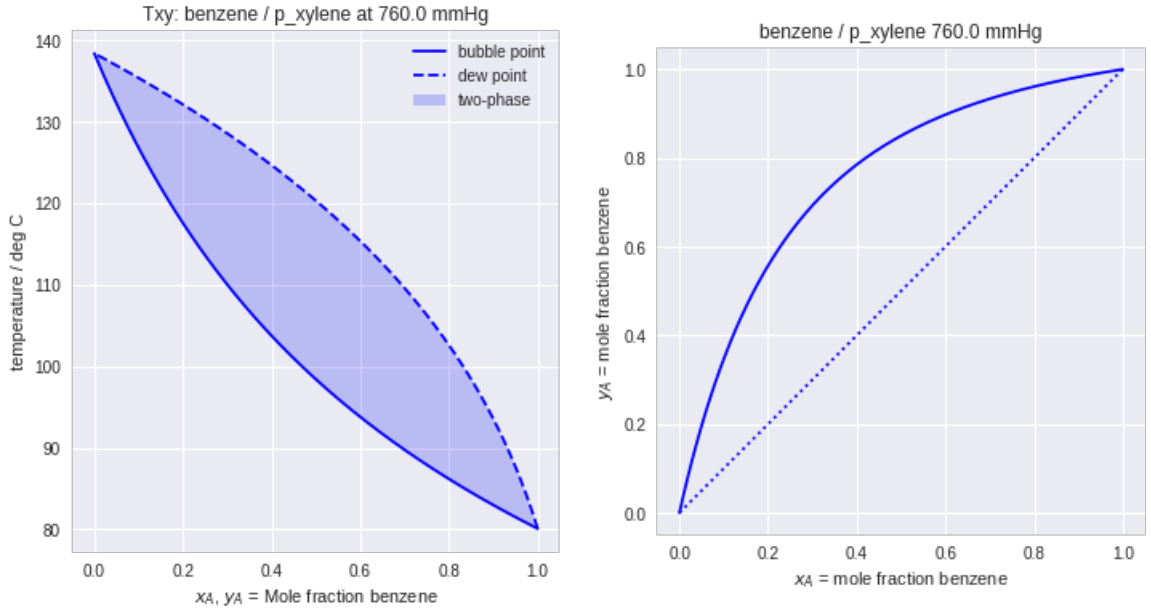
1. Why are the two curves labeled the 'dew point' and 'bubble point'?
2. Why is the region between the bubble and dew points called the two-phase region? Can a single phase actually exist that with a composition corresponding to a point between the two curves?
3. Where are the liquid and vapor phases located on the Pxy diagram? On the Txy diagram?
4. Use the above charts to answer the questions appearing earlier in this notebook. Which chart (or charts) makes it most convenient to answer each question?

Comparison to Experimental Data: Benzene/p-Xylene

The following cells show ideal mixture calculations for the benzene/p-xylene system which can be compared to experimental data available here.

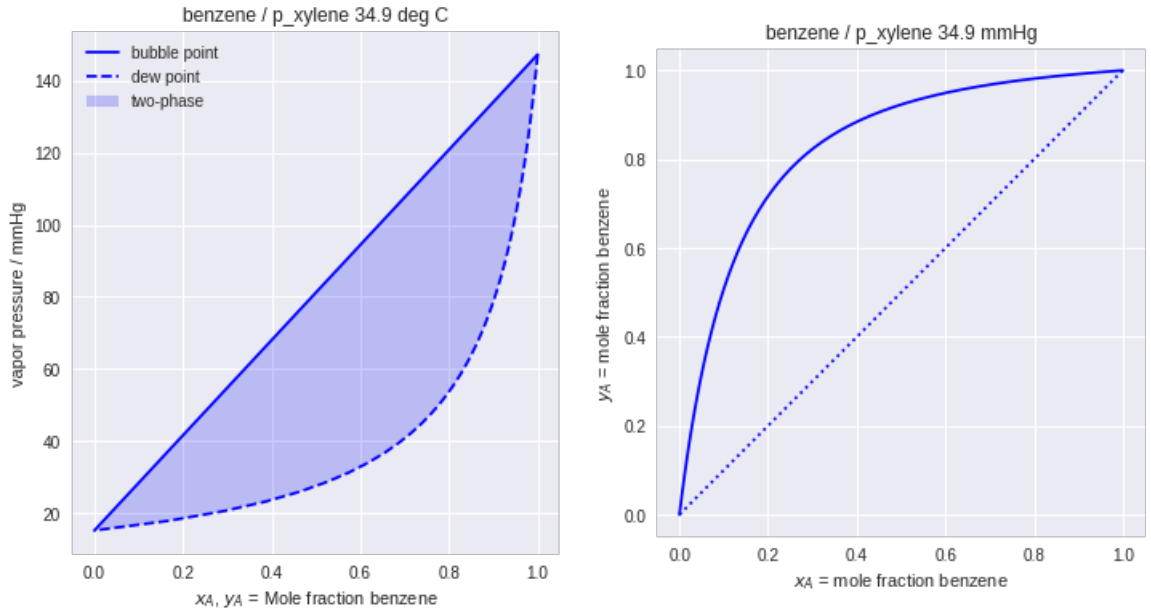
In [124]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
Txy(benzene, p_xylene, 760)
plt.subplot(1,2,2)
xy(benzene, p_xylene, P=760)
```



In [125]:

```
plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
Pxy(benzene, p_xylene, T=308-273.15)
plt.subplot(1,2,2)
xy(benzene, p_xylene, P=308-273.15)
```



Things to Do / Things to Think About

1. Repeat these calculations for a different example from the Dortmund Data Bank.

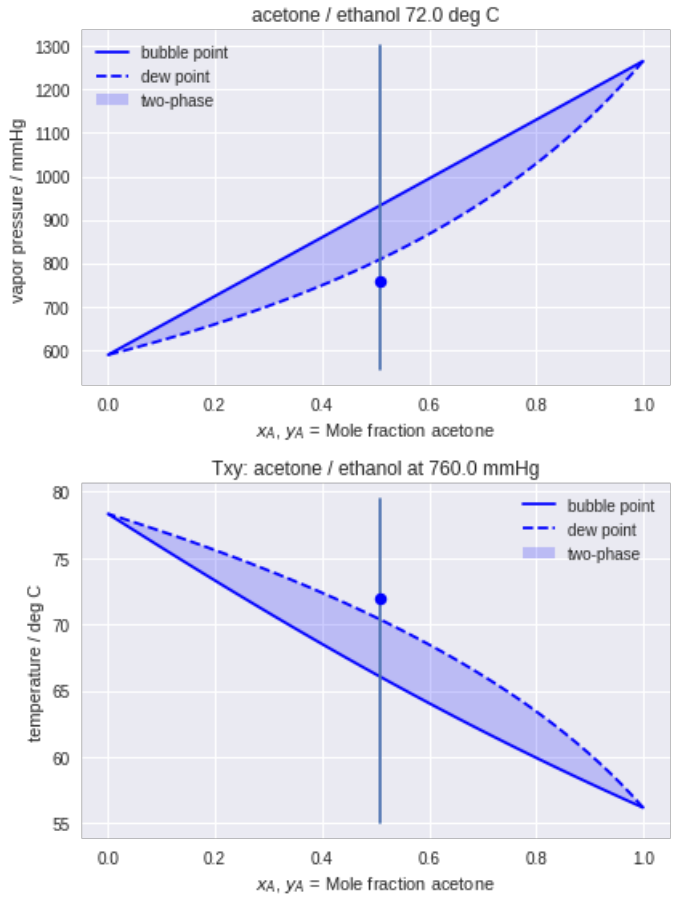
Flash

As a thought experiment, imagine you have a binary liquid mixture that with a composition of 50 mole% A, where A is the more volatile species. You heat the mixture in a sealed apparatus capable of maintaining constant pressure.

1. You gradually heat the mixture until the first bubble forms. Where is the point located on the Txy diagram? What is the composition of the bubble? What is the overall composition in the sealed vessel?
2. You continue heating the mixture until half of the mixture is evenly split between the liquid and vapor phases. By evenly split, we mean the moles of vapor equals the moles of liquid. What are the liquid and vapor phase compositions now? What is the overall composition?
3. You continue heating until the last drop of liquid. What are the compositions now?

In [126]:

```
#@title PT Flash { run: "auto", vertical-output: true }
P = 760 #@param {type:"slider", min:200, max:1200, step:10}
T = 72 #@param {type:"slider", min:30, max:140, step:1}
z = 0.51 #@param {type:"slider", min:0, max:1, step:0.01}

import matplotlib.pyplot as plt
import numpy as np

def PTflash(A, B, P, T, z):
    xA = (P - B.Psat(T))/(A.Psat(T) - B.Psat(T))
    yA = xA*A.Psat(T)/P

    plt.figure(figsize=(6,8))
    plt.subplot(2,1,1)
    Pxy(A, B, T)
    plt.plot([z ,z], plt.ylim())
    if xA < z < yA:
        plt.plot([xA, yA], [P, P], 'b-', marker='.', ms=15)
    else:
        plt.plot(z, P, 'b.', ms=15)

    plt.subplot(2,1,2)
    Txy(A, B, P)
    plt.plot([z ,z], plt.ylim())
    if xA < z < yA:
        plt.plot([xA, yA], [T, T], 'b-', marker='.', ms=15)
    else:
        plt.plot(z, T, 'b.', ms=15)
    plt.tight_layout()

PTflash(A, B, P, T, z)
```



Exercises

1. Modify this notebook to create Txy and xy diagrams for an acetaldehyde/ethanol mixture. Create an x-y diagram, and compare to the experimental data avaiable here: S. G. D'Avila and R. S. F. Silva, "Isothermal vapor-liquid equilibrium data by total pressure method. Systems acetaldehyde-ethanol, acetaldehyde-water, and ethanol-water," Journal of Chemical & Engineering Data, vol. 15 (3), 421-424, 1970.

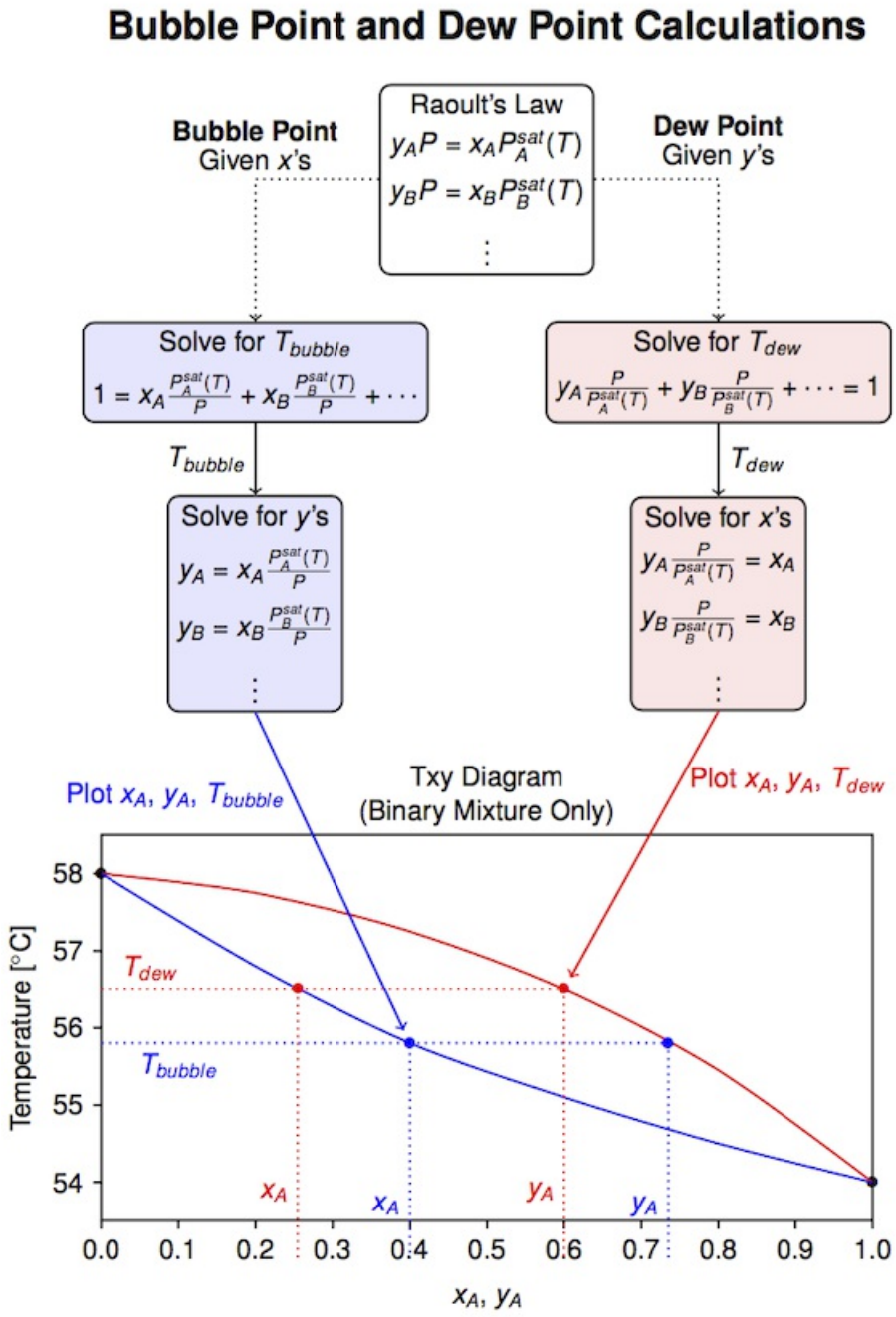
In [ ]:

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 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

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

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

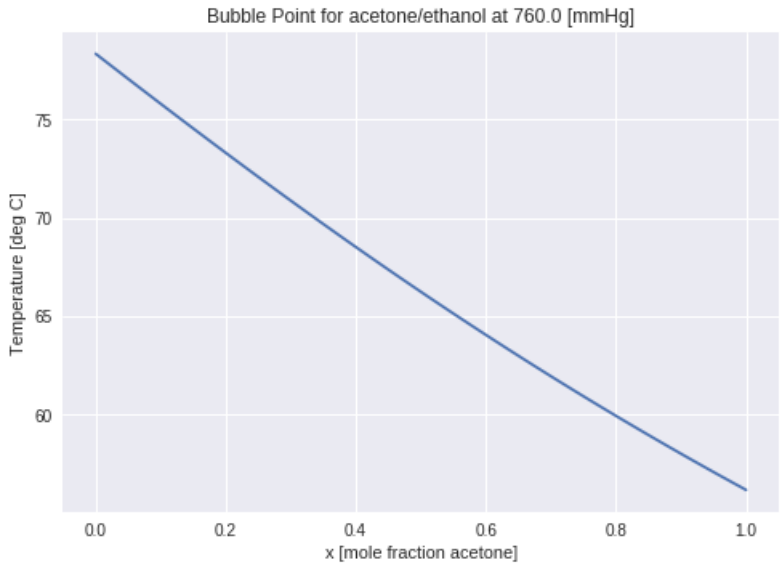
It's a relatively simply matter to encapsulate the bubble point calculation into a function that, given the liquid phase mole fraction for an ideal binary mixture, uses a root-finding procedure to return the bubble point temperature.

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]:

```
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));
```

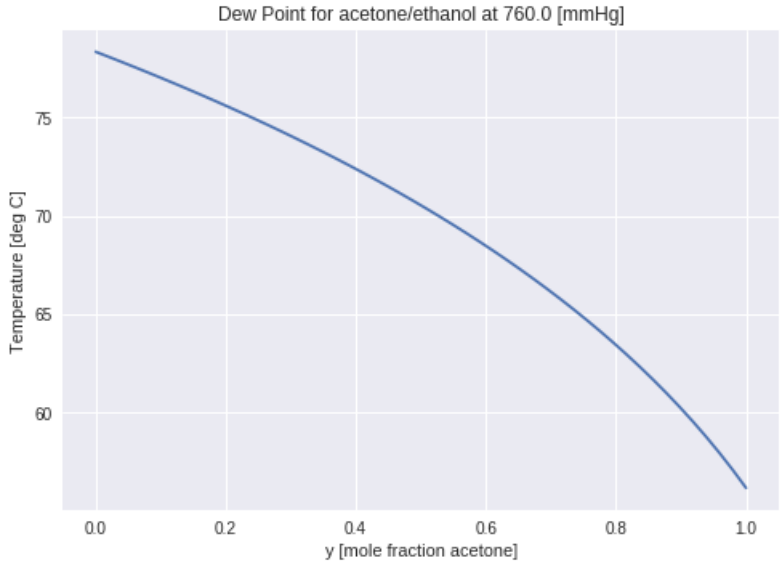
```
Dew point temperature = 68.520 [deg C]
Dew point composition = 0.400, 0.600
```

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 [ ]:

```

```

7.8 Bubble and Dew Points for Multicomponent 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.

Bubble and Dew Point Equations for Ideal Mixtures

Initialize the IPython workspace with with default settings for plots.

In [ ]:

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

In [ ]:

```
from IPython.display import YouTubeVideo
YouTubeVideo('theq1Go858E')
```

Bubble and dew point calculations for ideal mixtures are all about solving a fixed set of equations. If we have N chemical species, and refer to the liquid phase mole fractions as $x_1, \ldots, x_N$ and the vapor phase mole fractions as $y_1, \ldots, y_N$, then two of these equations are

$$x_1 + x_2 + \cdots + x_N = 1$$

and

$$y_1 + y_2 + \cdots + y_N = 1$$

The remaining equations come from Raoult's law. For each of the $n = 1, 2, \ldots, N$ species we have an equation

$$y_n P = x_n P_n^{sat}(T)$$

where $P_n^{sat}(T)$ is determined from experimental data or from a correlation such as Antoine's equation. This gives us a total $N + 2$ equations.

The unknown variables are the N values of x_n, the N values of y_n, plus temperature T and pressure P -- a total $2N + 2$ variables. With this as context, we can identify two types of problems.

Bubble Point Equations

If the composition of the liquid phase is known, then the equilibrium equations can be solved for the unknown vapor phase composition

$$y_n = x_n \frac{P_n^{sat}(T)}{P}$$

Substituting these values into the equation $y_1 + y_2 + \cdots + y_N = 1$ gives an equation.

$$x_1 \frac{P_1^{sat}(T)}{P} + \cdots + x_N \frac{P_N^{sat}(T)}{P} - 1 = 0$$

If P is known, then the equilibrium value of T is a root to this equation that can be found using standard root-finding functions in the Python or Matlab libraries.

If T is known, the solution for P is simply

$$P = x_1 P_1^{sat}(T) + x_2 P_2^{sat}(T) + \cdots + x_N P_N^{sat}(T)$$

Once both T and P are known, the vapor phase composition can be computed by substituting those values back into the first equation.

Dew Point Equations

If the composition of the vapor phase is known, then the equilibrium equations can be solved for the unknown vapor liquid phase composition

$$x_n = y_n \frac{P}{P_n^{sat}(T)}$$

Substituting these values into the equation $x_1 + x_2 + \cdots + x_N = 1$ gives an equation

$$y_1 \frac{P}{P_1^{sat}(T)} + \cdots + y_N \frac{P}{P_N^{sat}(T)} - 1 = 0$$

If P is known, then the equilibrium value of T is a root to this equation that can be found using standard root-finding functions in the Python or Matlab libraries.

If T is known, the solution for P is

$$\frac{1}{P} = \frac{y_1}{P_1^{sat}(T)} + \frac{y_2}{P_2^{sat}(T)} + \cdots + \frac{y_N}{P_N^{sat}(T)}$$

Once both T and P are known, the liquid phase composition can be computed by substituting those values back into the first equation.

Multicomponent Mixtures

In [ ]:

```
Psat = dict()
Psat['acetone'] = lambda T: 10**(7.02447 - 1161.0/(224 + T))
Psat['benzene'] = lambda T: 10**(6.89272 - 1203.531/(219.888 + T))
Psat['ethanol'] = lambda T: 10**(8.04494 - 1554.3/(222.65 + T))
Psat['toluene'] = lambda T: 10**(6.95805 - 1346.773/(219.693 + T))
```

For the multicomponent case we will use Python dictionaries to store compositions of the liquid phases. Bubble point functions.

Here we use the `fsolve` function from the scipy.optimize library to return the root of this 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 [ ]:

```
from scipy.optimize import fsolve
def Tbub(species,x):
    return fsolve(lambda T : sum([x[s]*Psat[s](T)/P for s in species]) - 1.0,60)[0]

def ybub(species,x):
    return {s: x[s]*Psat[s](Tbub(species,x))/P for s in species}
```

Dew point functions

In [ ]:

```
def Tdew(species,y):
    return fsolve(lambda T : sum([y[s]*P/Psat[s](T) for s in species]) - 1.0,60)[0]

def xdew(species,y):
    return {s: y[s]*P/Psat[s](Tdew(species,y)) for s in species}
```

Demonstration

In [ ]:

```
species = ['acetone','benzene','toluene']
z = dict()

P = 0.8*760
z['acetone'] = 0.1
z['benzene'] = 0.3
z['toluene'] = 0.6

print("\nBubble Point Calculations")

x = z
T = Tbub(species,x)
y = ybub(species,x)

print("Temperature = {:5.2f} [deg C]".format(T))
print("Pressure    = {:7.2f} [mmHg]".format(P))
print("  Composition    x[s]   y[s]")
for s in species:
    print("    {:10s} {:6.3f} {:6.3f}".format(s,x[s],y[s]))

print("\nDew Point Calculations")

y = z
T = Tdew(species,y)
x = xdew(species,y)

print("Temperature = {:5.2f} [deg C]".format(T))
print("Pressure    = {:7.2f} [mmHg]".format(P))
print("  Composition    x[s]   y[s]")
for s in species:
    print("    {:10s} {:6.3f} {:6.3f}".format(s,x[s],y[s]))
```

In [ ]:

```
```

7.9 Isothermal Flash and the Rachford-Rice Equation

Summary

This Jupyter notebook illustrates the use of the Rachford-Rice equation solve the material balances for an isothermal flash of an ideal mixture. The video is used with permission from learnCheme.com, a project at the University of Colorado funded by the National Science Foundation and the Shell Corporation.

Derivation of the Rachford-Rice Equation

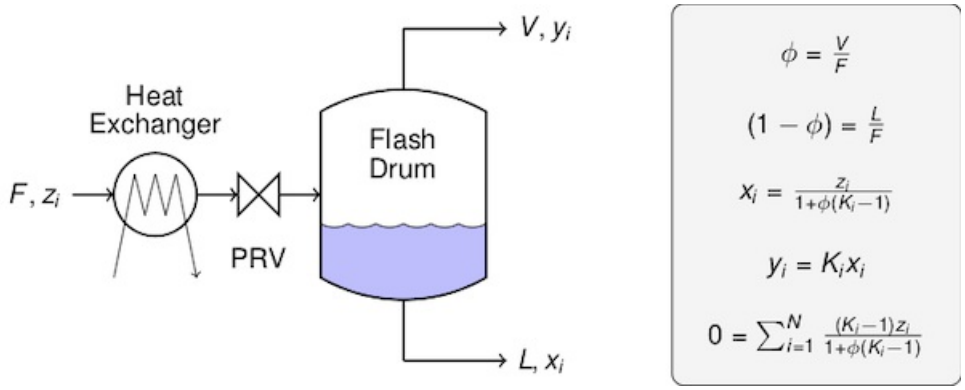
The derivation of the Rachford-Rice equation is a relatively straightford application of component material balances and Raoult's law for an ideal solution.

In [1]:

```
from IPython.display import YouTubeVideo
YouTubeVideo("ACxOiXWq1SQ",560,315,rel=0)
```

Out[1]:

The quantities, definitions, and equations are summarized in the following figure.



To sketch the derivation, we begin with the overall constraint on the liquid phase and vapor phase mole fractions $x_1 + x_2 + \cdots + x_N = 1$ and $y_1 + y_2 + \cdots + y_N = 1$. Subtracting the first from the second we find

$$\sum_{n=1}^{N}(y_n - x_n) = 0$$

This doesn't look like much, but it turns out to be the essential trick in the development.

Next we need expressions for y_n and x_n to substitute into terms in the sum. We get these by solving the component material balance and equilibrium equations for y_n and x_n. For each species we write a material balance

$$Lx_n + Vy_n = Fz_n$$

Dividing through by the feedrate we get a parameter $\phi = \frac{V}{L}$ denoting the fraction of the feedstream that leaves the flash unit in the vapor stream, the remaining fraction $1 - \phi$ leaving in the liquid stream. With this notation the material balance becomes

$$(1-\phi)x_n + \phi y_n = z_n$$

for each species.

The second equation is

$$y_n = K_n x_n$$

where the 'K-factor' for an ideal solution is given by Raoult's law

$$K_n = \frac{P_n^{sat}(T)}{P}$$

The K-factor depends on the operating pressure and temperature of the flash unit. Solving the material balance and equilibrium equations gives

$$x_n = \frac{z_n}{1+\phi(K_n-1)}$$
$$y_n = \frac{K_n z_n}{1+\phi(K_n-1)}$$

so that the difference $y_n - x_n$ is given by

$$y_n - x_n = \frac{(K_n-1)z_n}{1+\phi(K_n-1)}$$

Substitution leads to the Rachford-Rice equation

$$\sum_{n=1}^{N}\frac{(K_n-1)z_n}{1+\phi(K_n-1)} = 0$$

This equation can be used to solve a variety of vapor-liquid equilibrium problems as outline in the following table.

Problem Classification

Problem Type	z_i's	T	P	ϕ	Action
Bubble Point	✓	unknown	✓	0	Set $x_i = z_i$. Solve for T and y_i's
Bubble Point	✓	✓	unknown	0	Set $x_i = z_i$. Solve for P and y_i's
Dew Point	✓	unknown	✓	1	Set $y_i = z_i$. Solve for T and x_i's
Dew Point	✓	✓	unknown	1	Set $y_i = z_i$. Solve for P and x_i's
Isothermal Flash	✓	✓	✓	unknown	Solve for ϕ, x_i's, and y_i's

Isothermal Flash of a Binary Mixture

Problem specifications

In [1]:

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

In [2]:

```
A = 'acetone'
B = 'ethanol'

P = 760
T = 65

z = dict()
z[A] = 0.6
z[B] = 1 - z[A]
```

Compute the K-factors for the given operating conditions

In [3]:

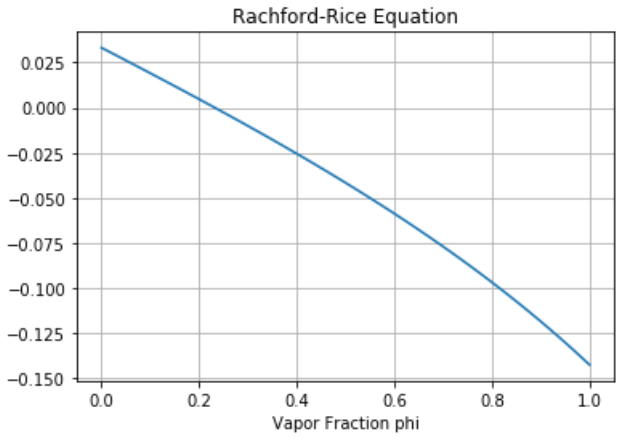
```
# Antoine's equations. T [deg C], P [mmHg]
Psat = dict()
Psat[A] = lambda T: 10**(7.02447 - 1161.0/(224 + T))
Psat[B] = lambda T: 10**(8.04494 - 1554.3/(222.65 + T))

# Compute K-factors
K = dict()
K[A] = Psat[A](T)/P
K[B] = Psat[B](T)/P

print("Pressure     {:6.2f} [mmHg]".format(P))
print("Temperature  {:6.2f} [deg C]".format(T))
print("K-factors:")
for n in Psat:
    print("   {:s}  {:7.3f}".format(n,K[n]))
```

```
Pressure     760.00 [mmHg]
Temperature   65.00 [deg C]
K-factors:
   acetone    1.338
   ethanol    0.576
```

Rachford-Rice equation

In [4]:

```
def RR(phi):
    return (K[A]-1)*z[A]/(1 + phi*(K[A]-1)) + (K[B]-1)*z[B]/(1 + phi*(K[B]-1))

phi = np.linspace(0,1)
plt.plot(phi,[RR(phi) for phi in phi])
plt.xlabel('Vapor Fraction phi')
plt.title('Rachford-Rice Equation')
plt.grid();
```



Finding roots of the Rachford-Rice equation

In [5]:

```
from scipy.optimize import brentq

phi = brentq(RR,0,1)

print("Vapor Fraction  {:6.4f}".format(phi))
print("Liquid Fraction {:6.4f}".format(1-phi))
```

```
Vapor Fraction  0.2317
Liquid Fraction 0.7683
```

Compositions

In [6]:

```
x = dict()
y = dict()

print("Component    z[n]    x[n]    y[n]")

for n in [A,B]:
    x[n] = z[n]/(1 + phi*(K[n]-1))
    y[n] = K[n]*x[n]
    print("{:10s} {:6.4n}  {:6.4f}  {:6.4f}".format(n,z[n],x[n],y[n]))
```

```
Component    z[n]    x[n]    y[n]
acetone       0.6  0.5565  0.7444
ethanol       0.4  0.4435  0.2556
```

Multicomponent Mixtures

In [7]:

```
P = 760
T = 65

z = dict()
z['acetone'] = 0.6
z['benzene'] = 0.01
z['toluene'] = 0.01
z['ethanol'] = 1 - sum(z.values())
```

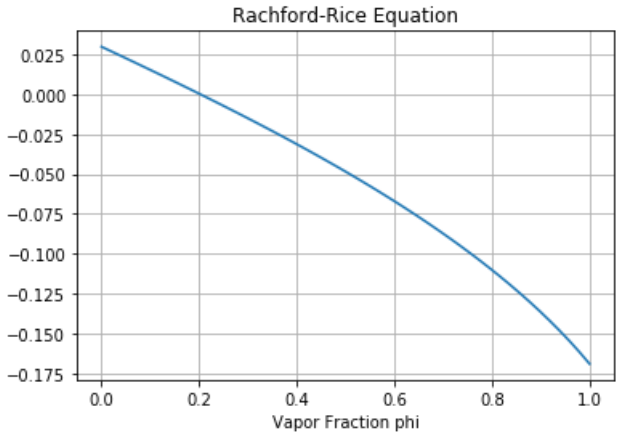
In [8]:

```
Psat = dict()
Psat['acetone'] = lambda T: 10**(7.02447 - 1161.0/(224 + T))
Psat['benzene'] = lambda T: 10**(6.89272 - 1203.531/(219.888 + T))
Psat['ethanol'] = lambda T: 10**(8.04494 - 1554.3/(222.65 + T))
Psat['toluene'] = lambda T: 10**(6.95805 - 1346.773/(219.693 + T))

K = {n : lambda P,T,n=n: Psat[n](T)/P for n in Psat}

print("Pressure     {:6.2f} [mmHg]".format(P))
print("Temperature  {:6.2f} [deg C]".format(T))
print("K-factors:")
for n in K:
    print("   {:s}  {:7.3f}".format(n,K[n](P,T)))
```

```
Pressure     760.00 [mmHg]
Temperature   65.00 [deg C]
K-factors:
   acetone    1.338
   benzene    0.613
   ethanol    0.576
   toluene    0.222
```

In [9]:

```
def RR(phi):
    return sum([(K[n](P,T)-1)*z[n]/(1 + phi*(K[n](P,T)-1)) for n in K.keys()])

phi = np.linspace(0,1)
plt.plot(phi,[RR(phi) for phi in phi])
plt.xlabel('Vapor Fraction phi')
plt.title('Rachford-Rice Equation')
plt.grid();
```



In [10]:

```
from scipy.optimize import brentq

phi = brentq(RR,0,1)

print("Vapor Fraction  {:6.4f}".format(phi))
print("Liquid Fraction {:6.4f}".format(1-phi))
```

```
Vapor Fraction  0.2033
Liquid Fraction 0.7967
```

In [11]:

```
x = {n: z[n]/(1 + phi*(K[n](P,T)-1)) for n in z}
y = {n: K[n](P,T)*z[n]/(1 + phi*(K[n](P,T)-1)) for n in z}

print("Component    z[n]    x[n]    y[n]")

for n in z.keys():
    print("{:10s} {:6.4f}  {:6.4f}  {:6.4f}".format(n,z[n],x[n],y[n]))
```

```
Component    z[n]    x[n]    y[n]
acetone    0.6000  0.5615  0.7511
benzene    0.0100  0.0109  0.0067
toluene    0.0100  0.0119  0.0026
ethanol    0.3800  0.4158  0.2396
```

Experiments suggest the bursting pressure of a 2 liter soda bottle is 150 psig.

Exercises

Design of a Carbonated Beverage

The purpose of carbonating beverages is to provide a positive pressure inside the package to keep out oxygen and other potential contaminants. The burst pressure of 2 liter soda bottles has been measured to be 150 psig (approx. 10 atm). For safety, suppose you want the bottle pressure to be no more than 6 atm gauge on a hot summer day in Arizona (say 50 °C,) and yet have at least 0.5 atm of positive gauge pressure at 0 °C. Assuming your beverage is a mixture of CO_2 and water, is it possible to meet this specification? What concentration (measured in g of CO_2 per g of water) would you recommend?

In [ ]:

```
```

7.10 Binary Distillation with McCabe-Thiele

Summary

This notebook illustrates the the design of binary distillation columns using the McCabe-Thiele graphical technique.

Problem Statement

A chemicals plant receives a 500 kg-mol/hour stream containing 60 mol% n-heptane, 40 mol% n-octane from a nearby refinery. The stream is available as a saturated liquid at 2 atmospheres. The plant operator needs to separate the mixture such that one product stream is 95 mol % pure n-octane, and that 95% of the n-octane is recovered.

Distillation

by User:Luigi Chiesa. Licensed under CC BY 3.0 via Wikimedia Commons.

Distillation is a workhorse for the large-scale separation of homogeneous volatile mixtures, such as the n-heptane/n-octane mixture described in the problem. Distillation exploits the difference in volatility among components of the mixture.

Txy Diagram

Can the desired product purity be accomplished in a single flash drum? Show why or why not using a Txy diagram.

Chemical Property Data

We begin the analysis by gathering Antoine equation data to create functions that compute the saturation pressure for n-heptane and n-octane. This data comes from Appendix B of the Murphy textbook with units of temperature in °C and pressure in mmHg.

In [17]:

```
class species(object):
    pass
from scipy.optimize import fsolve


A = species()
A.name = 'n-heptane'
A.Psat = lambda T: 10**(6.89677 -  1264.90/(T +  216.54))
A.Tsat = lambda P: fsolve(lambda T: A.Psat(T) - P,50)[0]


B.name = 'n-octane)'
B.Psat =

A.Psat(50)
A.Tsat(141)
```

Out[17]:

```
49.89210994784813
```

In [10]:

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

Psat = dict()
Psat['n-heptane'] = lambda T: 10**(6.89677 -  1264.90/(T +  216.54))
Psat['n-octane'] = lambda T: 10**(6.91868 - 1351.99/(T + 209.15))
```

The saturation temperatures (i.e, boiling points) can be computed by solving Antoine's equation for temperature. Next we create functions for each species for which we have specified a Psat function. This is more general than is really needed for the problem, but might be useful in other situations.

In [3]:

```
from scipy.optimize import fsolve

Tsat = dict()
for s in Psat.keys():
    Tsat[s] = lambda P, s=s: fsolve(lambda T: Psat[s](T)-P,50)[0]
```

Construct Txy Diagram

Next we compute the lower and upper bound on temperatures for the Txy diagram.

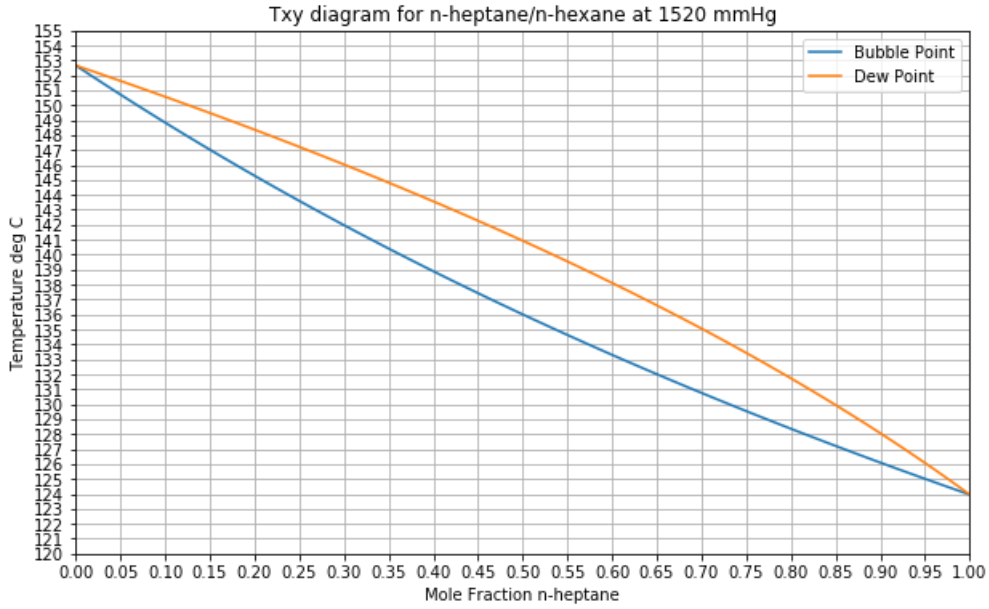
In [4]:

```
A = 'n-heptane'
B = 'n-octane'
P = 2*760;

print("{:12s}  {:.1f} deg C".format(A,Tsat[A](P)))
print("{:12s}  {:.1f} deg C".format(B,Tsat[B](P)))

T = np.linspace(Tsat[A](P),Tsat[B](P))
```

```
n-heptane     124.0 deg C
n-octane      152.7 deg C
```

In [5]:

```
x = lambda T: (P-Psat[B](T))/(Psat[A](T)-Psat[B](T))
y = lambda T: x(T)*Psat[A](T)/P

plt.figure(figsize=(10,6))

plt.plot([x(T) for T in T],T,[y(T) for T in T],T)
plt.xlabel('Mole Fraction n-heptane')
plt.ylabel('Temperature deg C')
plt.title('Txy diagram for n-heptane/n-hexane at 1520 mmHg')
plt.legend(['Bubble Point','Dew Point'])
plt.grid()

plt.ylim(120,155)
plt.xlim(0,1)
plt.xticks(np.linspace(0,1.0,21))
plt.yticks(np.linspace(120,155,36));
```



Analysis

The problem asked if the purity specification for the n-octane stream can be met in a single stage flash drum. The feed concentration of n-heptane is 60 mol%. The desired n-octane product would have an n-heptane concentration of 5 mol%. For the given feed, we see the minimum possible concentration of n-heptane would be at the dew point of approximate 138 °C, yielding a liquid phase composition of about 43 mol% n-heptane, and 57 mol% n-octane. This is far short of the stream specification.

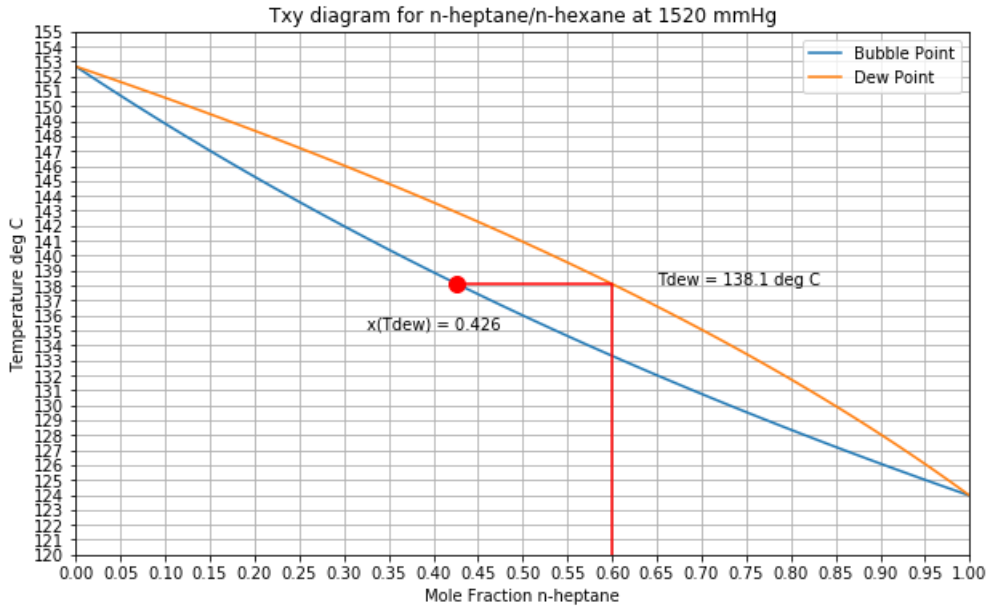
The following cell provides supporting calculations.

In [6]:

```
x = lambda T: (P-Psat[B](T))/(Psat[A](T)-Psat[B](T))
y = lambda T: x(T)*Psat[A](T)/P

plt.figure(figsize=(10,6))

plt.plot([x(T) for T in T],T,[y(T) for T in T],T)
plt.xlabel('Mole Fraction n-heptane')
plt.ylabel('Temperature deg C')
plt.title('Txy diagram for n-heptane/n-hexane at 1520 mmHg')
plt.legend(['Bubble Point','Dew Point'])
plt.grid()

plt.ylim(120,155)
plt.xlim(0,1)
plt.xticks(np.linspace(0,1.0,21))
plt.yticks(np.linspace(120,155,36));

Tdew = fsolve(lambda T: y(T)-0.60, 138)
ax = plt.axis()
plt.plot([0.6,0.6,x(Tdew)],[ax[2],Tdew,Tdew],'r-')
plt.plot(x(Tdew),Tdew,'ro',ms=10)
plt.text(0.6+0.05,Tdew,'Tdew = {:5.1f} deg C'.format(float(Tdew)))
plt.text(x(Tdew)-.1,Tdew-3,'x(Tdew) = {:5.3f}'.format(float(x(Tdew))))

#plt.savefig('Txy Diagram Soln.png')
```

Out[6]:

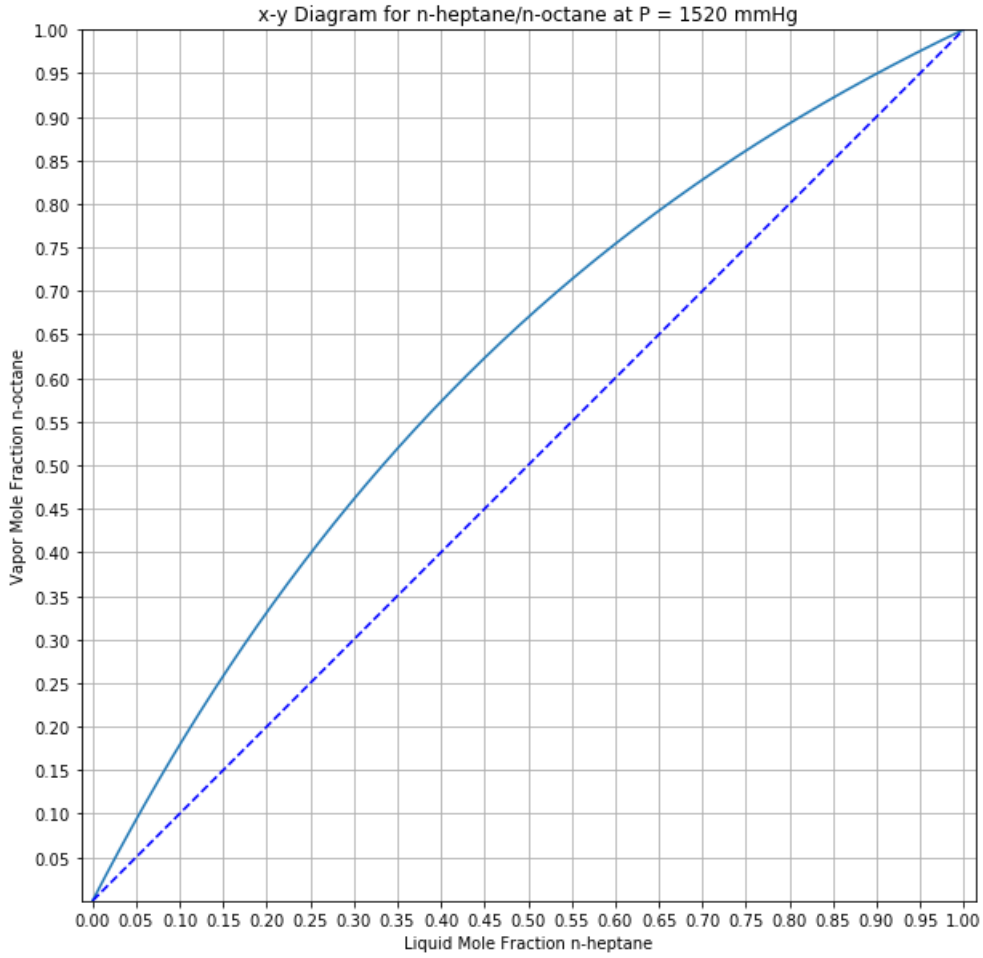
```
<matplotlib.text.Text at 0x111c3b940>
```



x-y Data

In [7]:

```
plt.figure(figsize=(10,10))
plt.plot([x(T) for T in T],[y(T) for T in T])

plt.plot([0,1],[0,1],'b--')
plt.axis('equal')
plt.xticks(np.linspace(0,1.0,21))
plt.yticks(np.linspace(0.05,1.0,20))

plt.title('x-y Diagram for {:s}/{:s} at P = {:.0f} mmHg'.format(A,B,P))
plt.xlabel('Liquid Mole Fraction {:s}'.format(A))
plt.ylabel('Vapor Mole Fraction {:s}'.format(B))

plt.xlim(0,1)
plt.ylim(0,1)
plt.grid();

#plt.savefig('xy Diagram.png')
```



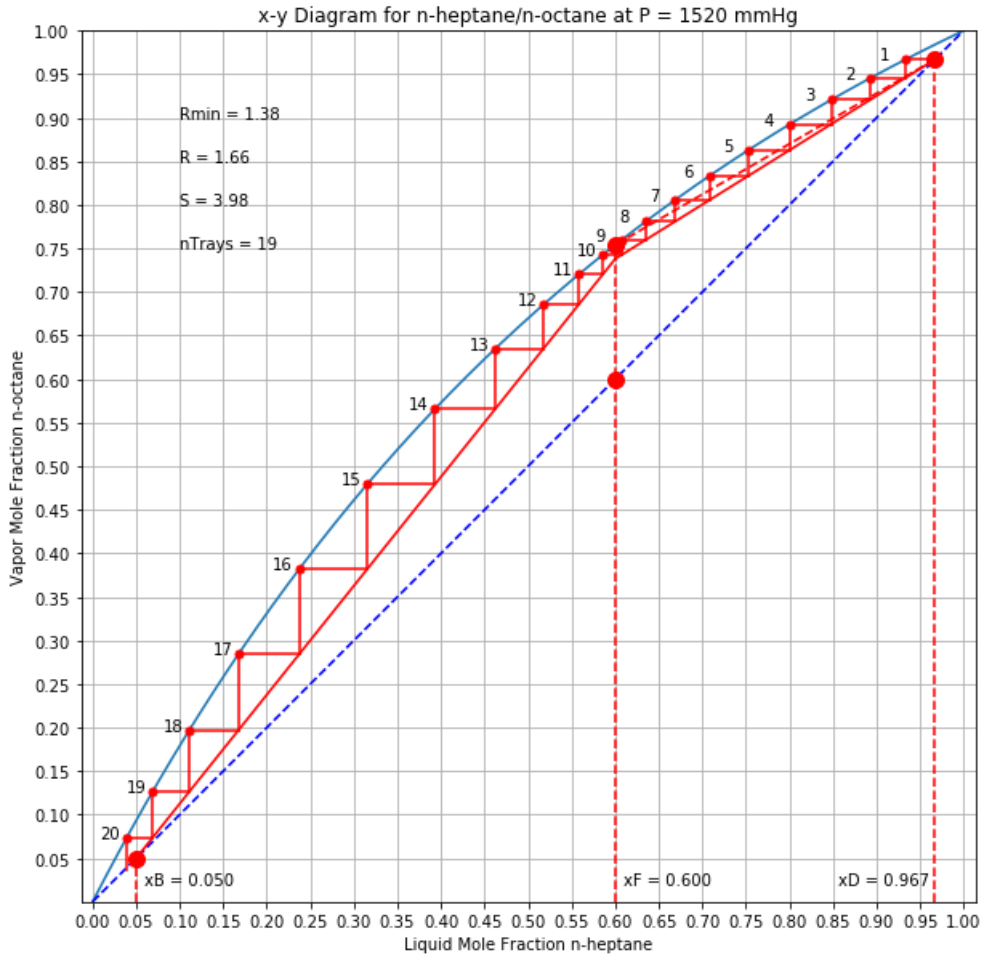
In [8]:

```
plt.figure(figsize=(10,10))
plt.plot([x(T) for T in T],[y(T) for T in T])

plt.plot([0,1],[0,1],'b--')
plt.axis('equal')
plt.xticks(np.linspace(0,1.0,21))
```

```
plt.yticks(np.linspace(0.05,1.0,20))

plt.title('x-y Diagram for {:s}/{:s} at P = {:.0f} mmHg'.format(A,B,P))
plt.xlabel('Liquid Mole Fraction {:s}'.format(A))
plt.ylabel('Vapor Mole Fraction {:s}'.format(B))

plt.xlim(0,1)
plt.ylim(0,1)
plt.grid();

xB = 0.05
xF = 0.60
xD = 0.96666

Tbub = fsolve(lambda T:x(T) - xF,100)
yF = y(Tbub)

plt.plot([xB,xB],[0,xB],'r--')
plt.plot(xB,xB,'ro',ms=10)
plt.text(xB+0.01,0.02,'xB = {:0.3f}'.format(float(xB)))

plt.plot([xF,xF,xF],[0,xF,yF],'r--')
plt.plot([xF,xF],[xF,yF],'ro',ms=10)
plt.text(xF+0.01,0.02,'xF = {:0.3f}'.format(float(xF)))

plt.plot([xD,xD],[0,xD],'r--')
plt.plot(xD,xD,'ro',ms=10)
plt.text(xD-0.11,0.02,'xD = {:0.3f}'.format(float(xD)))

plt.plot([xD,xF],[xD,yF],'r--')
Eslope = (xD-yF)/(xD-xF)
Rmin = Eslope/(1-Eslope)
R = 1.2*Rmin

zF = xD-R*(xD-xF)/(R+1)
plt.plot([xD,xF],[xD,zF],'r-')

Sslope = (zF-xB)/(xF-xB)
S = 1/(Sslope-1)
plt.plot([xB,xF],[xB,xB + (S+1)*(xF-xB)/S],'r-')

xP = xD
yP = xD

nTray = 0

while xP > xB:
    nTray += 1
    Tdew = fsolve(lambda T:y(T) - yP, 100)
    xQ = xP
    xP = x(Tdew)
    plt.plot([xQ,xP],[yP,yP],'r')
    plt.plot(xP,yP,'ro',ms=5)
    plt.text(xP-0.03,yP,nTray)

    yQ = yP
    yP = min([xD - (R/(R+1))*(xD-xP),xB + ((S+1)/S)*(xP-xB)])
    plt.plot([xP,xP],[yQ,yP],'r')

nTray -= 1

plt.text(0.1,0.90,'Rmin = {:0.2f}'.format(float(Rmin)))
plt.text(0.1,0.85,'R = {:0.2f}'.format(float(R)))
plt.text(0.1,0.80,'S = {:0.2f}'.format(float(S)))
plt.text(0.1,0.75,'nTrays = {:d}'.format(int(nTray)))

#plt.savefig('xy Diagram Soln.png')
```

Out[8]:

```
<matplotlib.text.Text at 0x112554f98>
```

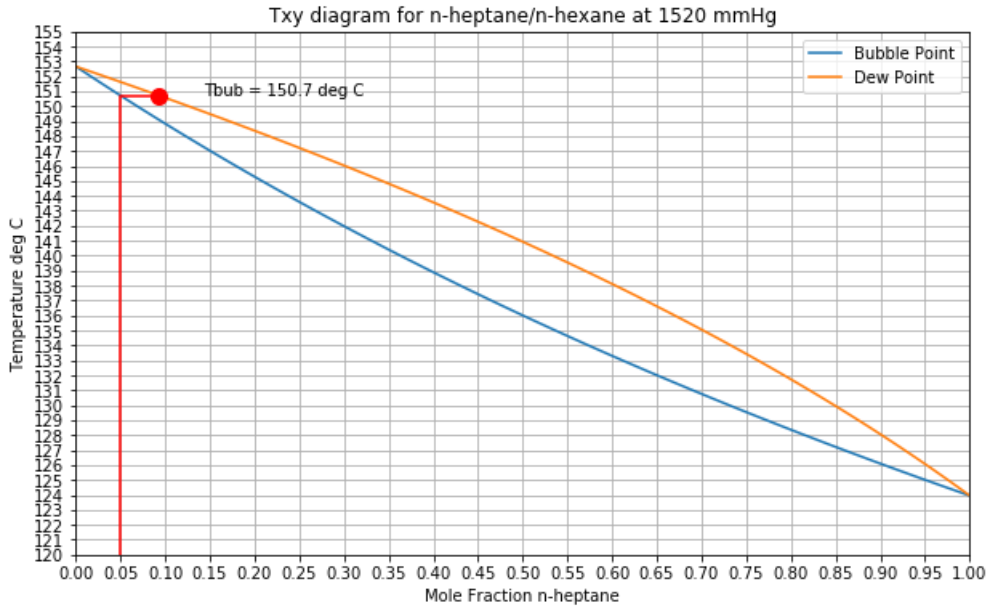


In [9]:

```
x = lambda T: (P-Psat[B](T))/(Psat[A](T)-Psat[B](T))
y = lambda T: x(T)*Psat[A](T)/P

plt.figure(figsize=(10,6))

plt.plot([x(T) for T in T],T,[y(T) for T in T],T)
plt.xlabel('Mole Fraction n-heptane')
plt.ylabel('Temperature deg C')
plt.title('Txy diagram for n-heptane/n-hexane at 1520 mmHg')
plt.legend(['Bubble Point','Dew Point'])
plt.grid()

plt.ylim(120,155)
plt.xlim(0,1)
plt.xticks(np.linspace(0,1.0,21))
plt.yticks(np.linspace(120,155,36));

Tbub = fsolve(lambda T: x(T)-xB, 138)
ax = plt.axis()
plt.plot([xB,xB,y(Tbub)],[ax[2],Tbub,Tbub],'r-')
plt.plot(y(Tbub),Tbub,'ro',ms=10)
plt.text(y(Tbub)+0.05,Tbub,'Tbub = {:5.1f} deg C'.format(float(Tbub)))

#plt.savefig('Txy Diagram Reboiler.png')
```

Out[9]:

```
<matplotlib.text.Text at 0x11165b390>
```



In [ ]:

Chapter 8. Energy Balances

8.1 Energy Balances on a Classroom

Heat Requirement

Classroom air quality standards typically specify between 4 and 12 air changes per hour. Consider a classroom with dimensions of 10 by 8 meters, and height of 3 meters. Outdoor air temperature is 0°C, and local campus standards require conditioning air to 25°C and supplying 6 air changes per hour. How much heat is required?

Solution

A generic energy balance can be written for a system with one inlet flow and one outlet flow as

$$\dot{m}_{out}\left(\hat{H}_{out}+\hat{E}_{K,out}+\hat{E}_{P,out}\right)=\dot{m}_{in}\left(\hat{H}_{in}+\hat{E}_{K,in}+\hat{E}_{P,in}\right)+\dot{Q}+\dot{W}_{shaft}$$

In this case, the changes in enthalpy will be significant larger than changes in kinetic or potential energy, and there is no substantial shaft work being done.The energy balance thereby reduces to

$$\dot{m}_{out}\hat{H}_{out}=\dot{m}_{in}\hat{H}_{in}+\dot{Q}$$

The mass inflow is equal to the mass outflow at steady. Solving for the heat requirement

$$\dot{Q}=\dot{m}\left(\hat{H}_{out}-\hat{H}_{in}\right)$$

Table B.8 of Felder, et al., presents data for the molar specific enthalpies of selected gases in units of kJ/gmol. For that purpose, we recast the heat balance in molar units

$$\dot{Q}=\dot{n}\left(\hat{H}_{out}-\hat{H}_{in}\right)$$

We can now do the necessary calculations.

In [ ]:

```
# volumetric flowrate at 25 C

Vdot = 6 * 10 * 8 * 3.  # cubic meters/hour
R = 0.08206             # liter-atm/gmol-K
P = 1.                  # atm
T = 25 + 273.15         # K
ndot = P*Vdot*1000/(R*T)     # gmol/hr

print("molar flow [gmol/hr] =", ndot)

Hin = -0.72             # kJ/gmol
Hout = 0.               # kJ/gmol

Qdot = ndot*(Hout - Hin)

print("heat required [kJ/hr] =", Qdot)
print("heat required [watts] =", Qdot*1000/3600)
print("heat required [BTU/hr] =", Qdot/1.055)
```

```
molar flow [gmol/hr] = 58856.73490056012
heat required [kJ/hr] = 42376.849128403286
heat required [watts] = 11771.346980112023
heat required [BTU/hr] = 40167.629505595534
```

Steam Requirement

The central campus utilities distribute steam to campus buildings. The steam is specified at 70 psig. How much steam is required to heat this classroom?

Solution

In this case we perform an energy balance on the steam condenser providing heat to the incoming air stream to the classroom. If the required heat is $\dot{Q}_{heat}$, then the condenser heat balance is

$$\dot{m}_{steam}\left(\hat{H}_{steam,out} - \hat{H}_{steam,in}\right) = -\dot{Q}_{heat}$$

(Be sure you understand the why the sign of $\dot{Q}_{heat}$ is negative.) Solving for the steam requirement

$$\dot{m}_{steam} = \frac{\dot{Q}_{heat}}{\hat{H}_{steam,in} - \hat{H}_{steam,out}}$$

We will assume the inlet and outlet steam are the saturated vapor and liquid, respectively, at 70 psig. Data for saturated steam can be found in Table B.6 of Felder, et al.

In [ ]:

```
Psteam = (70 + 14.696)/14.696 * 101325/100000.  # bar
print("steam pressure [bar absolute] =", Psteam)
```

```
steam pressure [bar absolute] = 5.839563282525857
```

After finding the enthalpies of saturated liquid and vapor at the indicated pressure:

In [ ]:

```
H_vapor = 2755.5    # kJ/kg
H_water = 670.4     # kJ/kg

m_dot = Qdot/(H_vapor - H_water)
print("steam flow [kg/hr] =", m_dot)
```

```
steam flow [kg/hr] = 20.32365312378461
```

Pipe Sizing

Piping for the distribution of saturated steam generally provide for a flow velocity between 25 and 35 m/s. What is the interior diameter of the piping needed to serve the classroom?

Solution

The volumetric flowrate of steam flowing in a pipe with diameter d at velocity v is

$$\dot{V} = Av = \left(\frac{1}{4}\pi d^2\right)v$$

For steam with a specific volume $\hat{V}$ the volumetric flow is $\dot{V} = \dot{m}\hat{V}$. Solving for d

$$d = 2\sqrt{\frac{\dot{m}\hat{V}}{\pi v}}$$

The specific volume is found in Table B.4 of Felder, et al.

In [ ]:

```
Vhat = 0.315      # m3/kg
v = 30            # m/s
pi = 3.1416
d = 2*(m_dot*Vhat/pi/v/3600)**0.5   # m

print("needed pipe diameter [cm] =", 100*d)
```

```
needed pipe diameter [cm] = 0.8687580416317904
```

In [ ]:

```

```

8.2 Water and Steam Calculator

In [ ]:

```
!pip install iapws -q
```

Properties of Steam

Execute the following the cell to estimate the properties of water at specified temperature and pressure.

In [ ]:

```
#@title Properties of Water { run: "auto", vertical-output: true }
#@markdown Enter pressure [bar]
P = 5.0 #@param {type:"number"}
#@markdown Enter temperature [deg C]
T = 200 #@param {type:"number"}

from iapws import IAPWS97

water = IAPWS97(P = P/10, T = T + 273.15)
sat_liq = IAPWS97(P = P/10, x = 0)

if sat_liq.T < T + 273.15:
  print('Superheated Vapor:')
elif sat_liq.T > T + 273.15:
  print('Supercooled Liquid:')

print("    h [kJ/kg]   =", round(water.h,1))
print("    u [kJ/kg]   =", round(water.u,1))
print("    v [m3/kg]   =", round(water.v,5))
print("    s [kJ/kg/K] =", round(water.s,3))
print()
print("At this pressure Tsat [deg C] =", round(sat_liq.T - 273.15, 2))
```

```
Superheated Vapor:
    h [kJ/kg]   = 2855.9
    u [kJ/kg]   = 2643.4
    v [m3/kg]   = 0.42503
    s [kJ/kg/K] = 7.061

At this pressure Tsat [deg C] = 151.84
```

Saturation Conditions at a Specified Pressure

Execute the following cell to estimate the properties of saturated liquid water and water vapor at a specified pressure.

In [ ]:

```
#@title Water Liquid/Vapor Saturation (Pressure) { run: "auto", vertical-output: true
}
#@markdown Enter pressure [bar]
P = 2.44 #@param {type:"number"}

from iapws import IAPWS97

sat_liq = IAPWS97(P = P/10, x = 0)
sat_vap = IAPWS97(P = P/10, x = 1)

print("T [deg C]   =", round(sat_liq.T - 273.15, 2))
print("\nSaturated Liquid:")
print("    h [kJ/kg]   =", round(sat_liq.h,1))
print("    u [kJ/kg]   =", round(sat_liq.u,1))
print("    v [m3/kg]   =", round(sat_liq.v,5))
print("    s [kJ/kg/K] =", round(sat_liq.s,3))

print("\nSaturated Vapor:")
print("    h [kJ/kg]   =", round(sat_vap.h,1))
print("    u [kJ/kg]   =", round(sat_vap.u,1))
print("    v [m3/kg]   =", round(sat_vap.v,5))
print("    s [kJ/kg/K] =", round(sat_vap.s,3))
```

```
T [deg C]   = 126.62

Saturated Liquid:
    h [kJ/kg]   = 531.9
    u [kJ/kg]   = 531.7
    v [m3/kg]   = 0.00107
    s [kJ/kg/K] = 1.599

Saturated Vapor:
    h [kJ/kg]   = 2715.4
    u [kJ/kg]   = 2536.0
    v [m3/kg]   = 0.73524
    s [kJ/kg/K] = 7.061
```

Saturation Conditions at a Specified Temperature

Execute the following cell to estimate the properties of saturated liquid water and water vapor at a specified temperature.

In [ ]:

```
#@title Water Liquid/Vapor Saturation (Temperature) { run: "auto", vertical-output: tr
ue }
#@markdown Enter temperature [deg C]
T = 81.3 #@param {type:"number"}

from iapws import IAPWS97

sat_liq = IAPWS97(T = T+273.15, x = 0)
sat_vap = IAPWS97(T = T+273.15, x = 1)

print("P [bar]   =", round(10*sat_liq.P, 5))
print("\nSaturated Liquid:")
print("    h [kJ/kg]   =", round(sat_liq.h,1))
print("    u [kJ/kg]   =", round(sat_liq.u,1))
print("    v [m3/kg]   =", round(sat_liq.v,5))
print("    s [kJ/kg/K] =", round(sat_liq.s,3))

print("\nSaturated Vapor:")
print("    h [kJ/kg]   =", round(sat_vap.h,1))
print("    u [kJ/kg]   =", round(sat_vap.u,1))
print("    v [m3/kg]   =", round(sat_vap.v,5))
print("    s [kJ/kg/K] =", round(sat_vap.s,3))
```

```
P [bar]   = 0.49966

Saturated Liquid:
    h [kJ/kg]   = 340.4
    u [kJ/kg]   = 340.4
    v [m3/kg]   = 0.00103
    s [kJ/kg/K] = 1.091

Saturated Vapor:
    h [kJ/kg]   = 2645.2
    u [kJ/kg]   = 2483.2
    v [m3/kg]   = 3.24219
    s [kJ/kg/K] = 7.593
```

In [ ]:

8.3 Basic Energy Computations

Computing Enthalpy and Internal Energy Changes for Common Situations

Internal energy (U) and enthalpy ($H = U + PV$) are thermodynamic state variables. We can use this property to compute changes in internal energy or enthalpy due to changes in pressure, temperature, phase, composition, and mixing/solution. The following table presents basic formulas for these calculations.

Change in	$\Delta\hat{H} = \Delta\hat{U} + P\Delta\hat{V}$	$\Delta\hat{U}$	**Comments**
Pressure	~ 0 (gas) ~$\hat{V}\Delta P$ (solid or liquid)	~ 0	Generally neglected except for large pressure changes.
Temperature	$\int_{T_1}^{T_2} C_p(T)dT$ $\approx \bar{C}_p(T_2 - T_1)$	$\int_{T_1}^{T_2} C_v(T)dT$ $\approx \bar{C}_v(T_2 - T_1)$	Expressions available for $C_p(T)$ $C_p \approx C_v$ (gases) $+R$ $C_p \approx C_v$ (liquids and solids)
Phase	$\Delta\hat{H}_{vap}$ (liquid to vapor) $\Delta\hat{H}_m$ (solid to liquid)	$\Delta\hat{U}_{vap} \approx \Delta\hat{H}_{vap} - RT_b$ $\Delta\hat{U}_m \approx \Delta\hat{H}_m$	
Composition due to Reaction	$\Delta\hat{H}_r^\circ = \sum_i \nu_i \Delta\hat{H}_{f,i}^\circ$ $\Delta\hat{H}_r^\circ = -\sum_i \nu_i \Delta\hat{H}_{c,i}^\circ$	$\Delta\hat{U}_r \approx \Delta\hat{H}_r - RT\Delta n_r$ $\Delta\hat{U}_r \approx \Delta\hat{H}_r$ (solid or liquid)	Δn_r is the cange in moles due to reaction Standard conditions are 25°C and 1 atm. Be sure all data uses same standard conditions.
Composition due to Mixing/Sol'n	$\Delta\hat{H}_{soln}$ $\Delta\hat{H}_{mix}$	$\Delta\hat{U}_{soln} \approx \Delta\hat{H}_{soln}$ $\Delta\hat{U}_{mix} \approx \Delta\hat{H}_{mix}$	Important for non-ideal mixtures. Typical units are per mole of solute, not solution.

Examples

Pumping a Fluid

For a particular fire-fighting situation, it is determined that 1,250 gpm is required. The fire hydrant will supply sufficient water at a pressure of 35 psig. A pressure of 180 psig is needed to reach the top of the 212 foot bulding. What size engine (in Hp) is required to power the fire pump?

In [1]:

```
Vdot = 1250/264.172/60            # flow in m**3/s
dP = (180 - 35)*101325/14.696     # pressure change in pascals (N/m**2)

P = Vdot*dP                       # power in N-m/sec = watts
print("fire pump requirement [watts] =", P)
print("fire pump requirement [hp] =", P/746)
```

```
fire pump requirement [watts] = 78841.96681903958
fire pump requirement [hp] = 105.68628259924876
```

Exercises

Vaporization of Phenol

Solid phenol at 25°C and 1 atm is converted to phenol vapor at 300°C and 3 atm. How much heat will be required?

In [ ]:

8.4 Energy Balances for a Steam Turbine

Summary

The notebook presents shows how to set up and solve the energy balances for a steam turbine flowsheet. The example demonstrates the use of steam tables.

Problem Statement

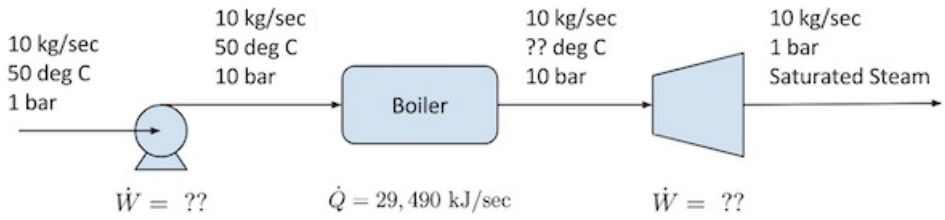
10 kg/sec of water at 1 bar absolute pressure and 50 °C is pumped isothermally to a pressure of 10 bar absolute, then fed to a boiler. The net heat input to the boiler is 29,490 kJ/sec. The boiler produces steam at 10 bar absolute which then enters a turbine. The exhaust leaving the turbine is saturated steam at 1 bar absolute.

1. What is the pump work?
2. What is the temperature of the steam produced by the boiler?
3. What is the turbine work produced?
4. What is the thermal efficiency (i.e., net work out/net heat in) for this system?
5. Locate the outlet conditions of the boiler and of the steam turbine on the attached T-S diagram. Do the specifications satisfy the second law of thermodynamics?

Steam tables are provided. If needed, you can assume the heat capacity of liquid water is 4.2 kJ/kg-C and for steam is 1.9 kJ/kg-C.

Solution

The first step in the analysis is to draw a flowsheet showing what is known and what is unknown.



The problem requires an energy balance for each process unit. The generic energy balance reads

$$\frac{dE_{sys}}{dt} = \dot{m}_{in}\hat{H}_{in} - \dot{m}_{out}\hat{H}_{out} + \dot{W} + \dot{Q}$$

where each term denotes a rate at which energy is transferred or work is done with units of power. At steady state the rate of change of system energy is zero, and mass inflow equal to outflow, $\dot{m}_{in} = \dot{m}_{out}$. The energy balance for each unit can be written as

$$0 = \dot{m}(\hat{H}_{in} - \hat{H}_{out}) + \dot{W} + \dot{Q}$$

This energy balance will be applied to each of three units in the flowsheet.

Part a. What is the pump work?

The pump is operated by a motor which provides work but no substantial amount of heat. The energy balance reduces to

$$0 = \dot{m}(\hat{H}_{in} - \hat{H}_{out}) + \dot{W}_{pump}$$

then solving for the pump work

$$\dot{W}_{pump} = \dot{m}(\hat{H}_{in} - \hat{H}_{out})$$

Substituting for enthalpy $\hat{H} = \hat{U} + P\hat{V}$,

$$\dot{W}_{pump} = \dot{m}([\hat{U}_{out} + P_{out}\hat{V}_{out}] - [\hat{U}_{in} + P_{in}\hat{V}_{in}])$$

Rearranging,

$$\dot{W}_{pump} = \dot{m}(\hat{U}_{out} - \hat{U}_{in} + P_{out}\hat{V}_{out} - P_{in}\hat{V}_{in})$$

The internal energy of liquid water is, to a good approximation, a function of temperature alone, so $\hat{U}_{out} \approx \hat{U}_{in}$. Liquid water is also very nearly incompressible which implies the specific volume is constant $\hat{V}_{out} \approx \hat{V}_{in}$. So to a good approximation

$$\dot{W}_{pump} \approx \dot{m}V(Pout - Pin)$$

The specific volume of water is 0.001 m^3 per kg. Expressing pressure in Pascals gives a straightforward calculation

$$\begin{aligned}\dot{W}_{pump} = 10\frac{kg}{sec} \times 0.001\frac{m^3}{kg}(10.0 \times 10^5\,\frac{N}{m^2} - 1.0 \times 10^5\,\frac{N}{m^2}) \\ = 9,000N - m \quad / \quad sec \\ = 9kilowatts\end{aligned}$$

At 746 watts per horsepower this corresponds to a work requirement of about 12 hp. In practice, a larger motor would be required to accommodate pump inefficiencies and other operational factors.

Part b. What is the temperature of the steam produced by the boiler?

The boiler has a large input of thermal energy, and the amount of mechanical work is negligible. The energy balance is then

$$\dot{Q}_{pump} = \dot{m}(\hat{H}_{out} - \hat{H}_{in})$$

The water flowing through the boiler begins at 50 °C and 10 bar, is heated to saturation temperature which point it vaporizes, and the resulting steam heated to the exit temperature. The change in specific enthalpy is given as the sum of these three processes

$$\hat{H}_{out} - \hat{H}_{in} = \Delta\hat{H}_{50^\circ C \to T_{sat}} + \Delta\hat{H}_v + \Delta\hat{H}_{T_{sat} \to T_{exit}}$$

The steam table provides the essential information on saturation temperature and enthalpy of vaporization at 10 bar.

P, bar (T^{sat}, °C)		Sat'd liquid	Sat'd vapor
0.006116	$\hat{H}$	0.00	2500.9
(0.01)	$\hat{U}$	0.00	2374.9
	$\hat{V}$	0.00100	206.55
0.1	$\hat{H}$	191.81	2583.9
(45.806)	$\hat{U}$	191.80	2437.2
	$\hat{V}$	0.00101	14.670
1.0	$\hat{H}$	417.50	2674.9
(99.606)	$\hat{U}$	417.40	2505.6
	$\hat{V}$	0.00104	1.6939
5.0	$\hat{H}$	640.09	2748.1

(151.85)	U	639.54	2560.7
	$\hat{V}$	0.00109	0.37481
10.0	$\hat{H}$	762.52	2777.1
(179.88)	$\hat{U}$	761.39	2582.7
	$\hat{V}$	0.00113	0.1944

Calculating each of the three terms

$$\begin{aligned}\Delta\hat{H}_{50^\circ C \to T_{sat}} &= C_{p,liq}(T_{sat} - 50) \\ &= 4.2\frac{kJ}{kg^\circ C}(179.88^\circ C - 50^\circ C) \\ &= 545.5\frac{kJ}{kg}\end{aligned}$$

$$\Delta\hat{H}_v = 2777.1\frac{kJ}{kg} - 762.52\frac{kJ}{kg} = 2,014.6\frac{kJ}{kg}$$

$$\Delta\hat{H}_{T_{sat} \to T_{exit}} = 1.9\frac{kJ}{kg^\circ C}(T_{exit} - 179.88^\circ C)$$

Rearranging the energy balance

$$\hat{H}_{out} - \hat{H}_{in} = \frac{\dot{Q}}{\dot{m}}$$

Substituting for $\hat{H}_{out} - \hat{H}_{in}$

$$545.5\frac{kJ}{kg} + 2,014.6\frac{kJ}{kg} + 1.9\frac{kJ}{kg \circ C}(T_{exit} - 179.88^\circ C) = \frac{29,490\frac{kJ}{sec}}{10\frac{kg}{sec}}$$

Solving for the unknown exit temperature

$$\begin{aligned}T_{exit} = 179.88^\circ C + &\frac{\frac{29,490\frac{kJ}{sec}}{10\frac{kg}{sec}} - 545.5\frac{kJ}{kg} - 2,014.6\frac{kJ}{kg}}{1.9\frac{kJ}{kg^\circ C}} \\ &= 179.88^\circ C + 204.7^\circ C \\ &= 385\$^\circ \quad \$ \quad C\end{aligned}$$

Part c. How much turbine work is produced?

Because the turbine exchanges very little thermal energy with the surroundings, energy balance for the turbine is

$$0 = \dot{m}(\hat{H}_{out} - \hat{H}_{in}) - \dot{W}_{turbine}$$

Note that the sign of $\dot{W}_{turbine}$ has been changed to correspond to the production of work by the turbine rather the usual convention where work is done on the system. Solving for $\dot{W}_{turbine}$

$$\dot{W}_{turbine} = \dot{m}(\hat{H}_{out} - \hat{H}_{in})$$

The turbine outlet consists of saturated water vapor at 1 bar. From the steam tables

$$\hat{H}_{out} = 2,674.9 \frac{kJ}{kg}$$

The inlet condition is superheated vapor at 10 bar and 385°C. The steam table does not explicitly include this condition, so the best we can do is interpolate

$$\begin{aligned}\hat{H}_{in} &= \hat{H}(10\ bar, 350^{\circ}C) + 35^{\circ}C \times \frac{\hat{H}(10\ bar, 400^{\circ}C) - \hat{H}(10\ bar, 350^{\circ}C)}{50^{\circ}C} \\ &= 3,158.2 + 35 \times \frac{3,264.5 - 3,158.2}{50} \\ &= 3,233.3 \frac{kJ}{kg}\end{aligned}$$

Calculating

$$\begin{aligned}\dot{W}_{turbine} &= 10 \frac{kg}{sec} \times (3,233.3 \frac{kJ}{kg} - 2,674.9 \frac{kJ}{kg}) \\ &= 5,584.1 \frac{kJ}{sec} \\ &= 5.58 megawatts\end{aligned}$$

Part d. What is the thermal efficiency, i.e., net work out/net heat in, for this system?

The thermal efficiency is given by

$$\epsilon_{thermal} = \frac{5,584.1 \frac{kJ}{sec} - 9 \frac{kJ}{sec}}{24.49 \frac{kJ}{sec}} = 0.228 = 22.8$$

This is relatively low compared to modern power generation systems which exhibit efficiencies of 40% or higher. The main limitation in this example is the relatively modest steam temperature exiting the boiler, and the absence of steam regenerators commonly used in commercial power generation.

In [ ]:

8.5 Humidity and Psychrometrics

Summary

Psychrometrics is the study of the physical and thermodynamic properties of air and water vapor mixtures. Psychrometrics has a wide range of drying, humidification, weather, and environmental applications. The same principles, however, apply to any mixture of a condensable vapor mixed with non-condensable gase.

Definitions of Humidity

The familiar definitions of mass and mole fraction, of course, apply to a mixture of a vapor and condensable gas. But there are also additional methods for expressing composition and the relative amount of the condensable component.

For the definitions below, we will use A to represent the non-condensable component (typically air), and W to denote the condensable component (typically water).

Relative Humidity

Relative humidity (symbol RH or ϕ) is the ratio of the partial pressure of water vapor to the equilibrium saturation pressure.

$$\text{Relative Humidity} = RH = \phi = \frac{p_W}{p_W^{sat}(T)}$$

Relative humidity depends on temperature and, for mixtures of fixed composition, pressure. It is commonly expressed as a percentage.

Absolute Humidity

Absolute humidity (symbol AH) is the total mass of water vapor per unit volume of humid air.

$$\text{Absolute Humidity} = AH = \frac{m_W}{V}$$

Typical units are kg/m^3 or g/m^3. Absolute humidity is useful for expressing how much water is present in a given volume at a particular pressure and temperature. However, because volume varies with temperature and pressure, volumetric measures are generally not used directly in mass and energy balances.

Assuming the ideal gas law holds, the mass of water in a given volume is

$$AH = \frac{m_W}{V} \approx \frac{M_W p_W}{RT} \approx \frac{M_W y_W P}{RT}$$

where M_W is the molar mass of water, p_W is the partial pressure, and y_W is mole fraction. For a given mixture, increasing pressure compresses the mixture into a smaller volume and therefore increases absolute humidity. Conversely, increasing temperature expands the volume and therefore descreases absolute humidity.

Specific Humidity

Specific Humidity (symbol SH) is the mass of water divided by total mass of moist air.

$$\text{Specific Humidity} = SH = \frac{m_W}{m_A + m_W}$$

This is equivalent to mass fraction. While this would work for a variety of applications, the resulting calculations are more complicated than necessay for the majority of cases where the mass flow of dry air is often constant.

Mixing Ratio (or Moisture Content, or Humidity Ratio)

Mixing Ratio (or Moisture Content) (symbol w) is ratio of the mass of water vapor to the mass of dry air.

$$\text{Mixing Ratio} = w = \frac{m_W}{m_A}$$

Typical units are kg of water per kg of dry air. (Older literature will sometimes show units of grains of water per pound of dry air. There are 7000 grains per pound.)

Mixing ratio is the most commonly encountered measure of humidity because of it's computational utility in humidifciation and drying applications.

From the ideal gas law

$$m_W = \frac{M_W p_W v}{RT}$$

and

$$m_A = \frac{M_A p_A v}{RT} = \frac{M_A (P - p_W) v}{RT}$$

Taking the ratio

$$w = \frac{m_W}{m_A} = \frac{M_W}{M_A} \frac{p_W}{P - p_W} = \frac{\omega p_W}{P - p_W}$$

where $\omega = \frac{M_W}{M_A} = 0.622$ is the ratio of molar masses. At saturation the mixing ratio is

$$w_{sat}(T, P) = \frac{\omega p_W^{sat}(T)}{P - p_W^{sat}(T)}$$

For a relative humidity ϕ, the mixing ratio is

$$w(T, P, \phi) = \frac{\omega \phi p_W^{sat}(T)}{P - \phi p_W^{sat}(T)}$$

This expression demonstrates that the mixing ratio depends on temperature, pressure, and relative humidity.

Dew Point

The dew point (symbol T_{dew}) is the temperature at which the first drop of dew is formed when air is cooled at constant pressure. It is an indirect measure of moisture content.

Data Sources

Molar Masses of Air and Water

In [2]:

```
Ma = 0.78*(2*14.00675) + 0.21*(2*15.994) + 0.01*39.948
print("Molar Mass of Air = {0:0.2f}".format(Ma))

Mw = 2*1.00794 + 15.9994
print("Molar Mass of Water = {0:0.2f}".format(Mw))

print("Ratio Mw/Ma = {0:0.4f}".format(Mw/Ma))
```

```
Molar Mass of Air = 28.97
Molar Mass of Water = 18.02
Ratio Mw/Ma = 0.6219
```

Vapor Pressure of Water

There are a number of widely used correlations for the vapor pressure of water, among them Antoine's equation, Goff-Gratch equation, and the Arden Buck equation. The following cell implements correlations recommended by Wagner and Pruß for the IAPWS 1995 Steam Tables.

For the range 0 °C to 373 °C, the vapor pressure of liquid water is given by

$$\ln \frac{p_w^{sat}}{p_c} = \frac{T_c}{T}\left(a_1\vartheta + a_2\vartheta^{1.5} + a_3\vartheta^3 + a_4\vartheta^{3.5} + a_5\vartheta^4 + a_6\vartheta^{7.5}\right)$$

where

$$\vartheta = 1 - \frac{T}{T_c}$$

and $T_c = 647.096\,K$ and $P_c = 220.640$ bar are the temperature and pressure at the critical point. For the range -100 °C to 0.01 °C, the vapor pressure of ice is calculated using

$$\ln \frac{p_w^{sat}}{p_n} = b_0(1-\theta^{-1.5}) + b_1(1-\theta^{-1.25})$$

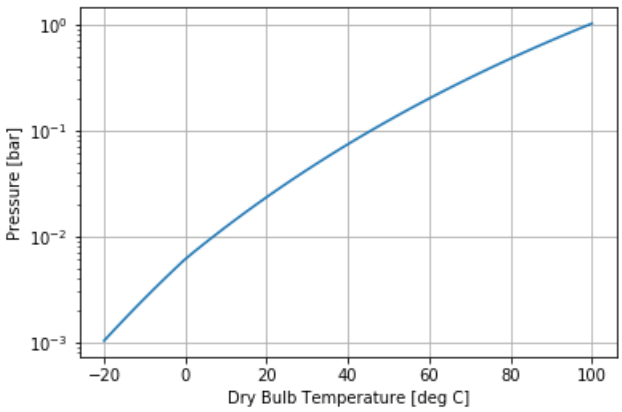
where

$$\theta = \frac{T}{T_n}$$

and $T_n = 0.01^\circ$ C and $p_n = 0.00611657\,bar$ are the triple point temperature and pressure. These correlations assume temperature in Kelvin, and returns pressure in the same units at p_c and p_n.

In [3]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np


def pSatW_(T):
    """Calculates the saturation pressure of water.

    Args:
        T: Temperature in degrees Celsius between -100C and 373C.

    Returns:
        The saturation pressure in bars.

    """
    Tc = 647.096          # critcal temperature in K
    Pc = 220.640          # critical pressure of water in bar
    a = np.array([0.0, -7.85951783, 1.84408259, -11.7866497, 22.6807411, -15.9618719, 1
.80122502])

    Tn = 273.16           # triple point temperature in K
    Pn = 0.00611657       # triple point pressure in bar
    b = np.array([-13.928169, 34.707823])

    if T >= 0.01 and T <= 373.0:
        phi = 1.0 - (T + 273.15)/Tc
        return Pc*np.exp((Tc/(T + 273.15))*(a[1]*phi + a[2]*phi**1.5
            + a[3]*phi**3.0 + a[4]*phi**3.5 + a[5]*phi**4.0 + a[6]*phi**7.5))
    elif T < 0.01 and T >= -100.0:
        theta = (T+273.15)/Tn
        return Pn*np.exp(b[0]*(1 - theta**-1.5) + b[1]*(1 - theta**-1.25))
    else:
        return float('nan')
        #raise ValueError('Temperture {0:0.2f} outside of valid range for pSatW.'.forma
t(T))

# extend function to nparray
pSatW = np.vectorize(pSatW_)


T = np.linspace(-20.0, 100.0)
plt.semilogy(T,pSatW(T))
plt.xlabel('Dry Bulb Temperature [deg C]')
plt.ylabel('Pressure [bar]')
plt.grid()
```



Steam Tables

The International Association for the Properties of Water and Steam (IAPWS) produces highly authoritative data sets for the properties of water and steam. Their 'recommended formulations' are the foundation for the steam tables most widely used by engineers.

iapws is an implementation of IAPWS-IF97 formulation for the properties of water and steam. The following cells demonstrate the installation of iapws, and use iapws to calculate properties needed for psychrometry.

Installation

In [4]:

```
# shell escape command to install iapws
!pip install iapws

# sample usage of iapws to demonstrate successful installation
from iapws import IAPWS97

# units are MPa, K, kJ/kg

sat_steam=IAPWS97(P=1,x=1)                #saturated steam with known P
sat_liquid=IAPWS97(T=370, x=0)            #saturated liquid with known T
steam=IAPWS97(P=2.5, T=500)               #steam with known P and T
print(sat_steam.h, sat_liquid.h, steam.h) #calculated enthalpies
```

```
Requirement already satisfied: iapws in /anaconda3/lib/python3.7/site-packages (1.4)
Requirement already satisfied: scipy in /anaconda3/lib/python3.7/site-packages (from ia
pws) (1.2.1)
2777.1195376846617 405.81466030352686 2811.704862899988
```

Heat of Vaporization

In [5]:

```
def Hvap(T):
    sat_vapor = IAPWS97(T = T + 273.15, x=1)
    sat_liquid = IAPWS97(T = T + 273.15, x=0)
    return sat_vapor.h - sat_liquid.h

print("Heat of Vaporization at 0 deg C = {0:0.2f} kJ/kg".format(Hvap(0.0)))
```

```
Heat of Vaporization at 0 deg C = 2500.93 kJ/kg
```

Heat Capacities

In [6]:

```
T = 25.0

def Cpvap(T):
    return IAPWS97(T = T + 273.15, x=1).cp

def Cpliq(T):
    return IAPWS97(T = T + 273.15, x=0).cp

sat_vapor = IAPWS97(T = T + 273.15, x=1)
print("Heat Capacity of Water Vapor at {0:0.1f}C = {1:0.3f} kJ/kg/K".format(T,Cpvap(25)
))

sat_liq = IAPWS97(T = T + 273.15, x=0)
print("Heat Capacity of Water Liquid at {0:0.1f}C = {1:0.3f} kJ/kg/K".format(T,Cpliq(25
)))
```

```
Heat Capacity of Water Vapor at 25.0C = 1.912 kJ/kg/K
Heat Capacity of Water Liquid at 25.0C = 4.182 kJ/kg/K
```

Thermodynamic Modeling

Specific Volume

For the purposes of psychrometric modeling, the specific volume (symbol v) is defined as the volume of humid air per kilogram of dry air.

Given 1 kilograms of dry air containing w kilograms of water vapor, from the ideal gas law the specific volume is

$$v = (\frac{1}{M_a} + \frac{w}{M_w})\frac{RT}{P}$$

Contours of constant specific volume are typically plotted on the psychrometric chart. For this purpose we need a relationship of w to the dry bulb temperature T. Solving for w as a function of T

$$w = \frac{M_w P v}{RT} - \frac{M_w}{M_a}$$

Enthalpy

For humidification operations, a convenient reference state for enthalpy calculations is (T_{ref} = 0 °C, P_{ref} = 1 bar) with water a saturated liquid.

For a dry bulb temperature T, the specific enthalpy of saturated water vapor relative to the reference temperature is

$$\hat{h}_w(T) = \Delta\hat{H}_{vap}(T_{ref}) + \hat{C}_{p,vap}(T - T_{ref})$$

The specific enthalpy of air

$$\hat{h}_a(T) = \hat{C}_{p,air}(T - T_{ref})$$

Combining these for a mixture of 1 kilogram of dry air and w kilograms of water vapor, the enthalpy per kilogram of dry air is

$$h(T,w) = \hat{C}_{p,air}(T - T_{ref}) + w\left(\Delta\hat{H}_{vap}(T_{ref}) + \hat{C}_{p,vap}(T - T_{ref})\right)$$

Psychrometric charts typically show contours of constant specific enthalpy. Solving for w in terms of h and T

$$w = \frac{h - \hat{C}_{p,air}(T - T_{ref})}{\Delta\hat{H}_{vap}(T_{ref}) + \hat{C}_{p,vap}(T - T_{ref})}$$

Wet Bulb Temperature

Considered an insulated chamber containing a long, closed water bath. Humid air enters at temperature T with w kilogram of water vapor per kilogram of dry air. Upon passing over the water bath, evaporative cooling lowers the bath temperature to a steady-state value T_s. To makeup for evaporative losses, liquid water is added to the bath at temperature T_s and a rate $w_s - w$ per kilogram of dry air. The following model assumes the humid air passes exits the chamber at temperture T_s and a partial pressure of water equal to the vapor pressure at temperature T_s.

The enthalpy of humid air per kilogram of dry air is given by

$$h(T,w) = \hat{C}_{p,air}\,(T - T_{ref}) + w\left[\Delta\hat{H}_{vap}(T_{ref}) + \hat{C}_{p,vap}(T - T_{ref})\right]$$

where T_{ref} is a common reference temperature. The enthalpy balance for the humidification device per kilogram of dry air is

$$h(T,w) + (w_s - w)\hat{C}_{p,liq.water}(T_s - T_{ref}) = h(T_s, w_s)$$

The enthalpy balance establishes a relationship between T_s and w_s given the initial conditions of the humid air stream, T and w.

For the psychrometric chart it is necessary to plot w as a function of T. Given T_s and w_s, the solution for w is

$$w = \frac{w_s\left[\Delta\hat{H}(T_{ref}) + \hat{C}_{p,vap}(T_s - T_{ref}) - \hat{C}_{p,liq}(T_s - T_{ref})\right] - \hat{C}_{p,air}\,(T - T_s)}{\Delta\hat{H}(T_{ref}) + \hat{C}_{p,vap}(T - T_{ref}) - \hat{C}_{p,liq}(T_s - T_{ref})}$$

Psychrometric Chart

The following cell exploits the analysis above to create a function for plotting psychrometric charts. The function relies on the functions in prior cells, so execute those before attempting to run this function.

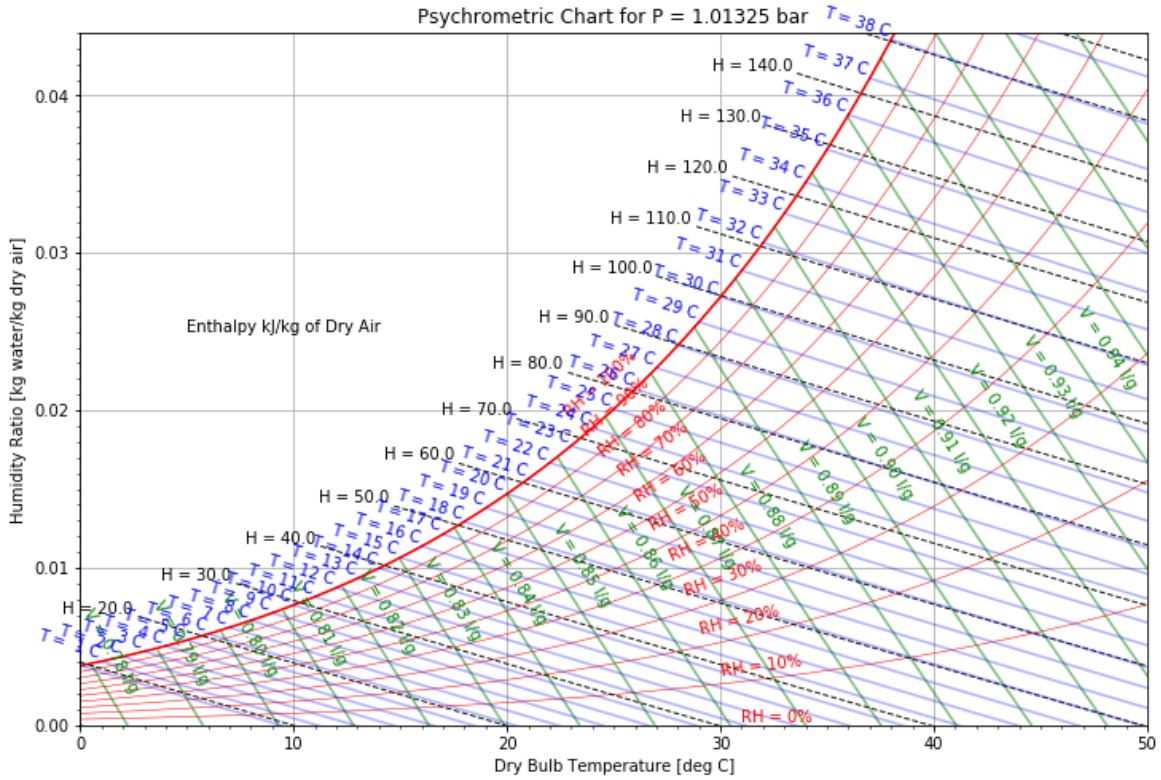
Function Definition

In [24]:

```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from scipy.optimize import fsolve

P = 1.01325         # bar
R = 0.08314         # m3 bar /K kg-mol
Tref = 0.0          # Celsius

Cpair = 1.005       # kJ/kg K
Cpvap = 1.912       # kJ/kg K
Cpliq = 4.182       # kJ/kg K
Hvap = 2500.9       # kJ/kg at Tref

def psychrometric_chart(P = 1.01325, wmax=0.044, tmin=0.0, tmax=50.0,
            Relative_Humidity=True, Volume=True, Enthalpy=True, Wet_Bulb=True):

    fig = plt.figure(figsize = (12, 8))
    plt.axis([tmin, tmax, 0.0, wmax])

    ax = plt.axes()
    ax.xaxis.set_major_locator(ticker.MultipleLocator(10))
    ax.xaxis.set_minor_locator(ticker.MultipleLocator(1))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(0.010))
    ax.yaxis.set_minor_locator(ticker.MultipleLocator(0.001))
```

```
    ax.yaxis.set_minor_locator(tick.MultipleLocator(0.001))

    ax.set_title('Psychrometric Chart for P = {0:0.5f} bar'.format(P))
    ax.set_xlabel('Dry Bulb Temperature [deg C]')
    ax.set_ylabel('Humidity Ratio [kg water/kg dry air]')
    ax.grid(True)

    # scale factors to convert slopes to angle of rotation on display
    DX = (tmax - tmin)/fig.get_size_inches()[0]
    DY = wmax/fig.get_size_inches()[1]

    # temperature where saturation curve intersects display axes
    wmin = (Mw/Ma)/(P/pSatW(tmin) - 1.0)

    tmid = min(tmax, fsolve(lambda t: (Mw/Ma)/(P/pSatW(t) - 1.0) - wmax, (tmin+tmax)/2.0
)[0])
    wmid = (Mw/Ma)/(P/pSatW(tmid) - 1.0)

    # plot saturation curve
    T = np.linspace(tmin, tmax, 200)
    ax.plot(T, (Mw/Ma)/(P/pSatW(T)-1.0), 'r')

    # plot contours of constant relative humidity
    if Relative_Humidity:
        k = 2*len(T)/3
        dk = len(T)/60
        for rh in np.linspace(0, 1.0, 11):
            w = (Mw/Ma)*rh/(P/pSatW(T) - rh)
            ax.plot(T , w, 'r', linewidth=0.5)
            k -= dk
            idx = round(k)
            ax.text(T[idx], w[idx], "RH = {0:0.0f}%".format(100*rh), color="r",
va="bottom", ha="center",
                 rotation = 180.0*np.arctan2((w[idx+1]-w[idx])/DY,((T[idx+1]-T[idx]))/DX
)/np.pi)

    if Volume:
        vmin = round((1.0/Ma)*R*(tmin+273.15)/P, 2)
        vmax = round((1.0/Ma + wmax/Mw)*R*(tmax+273.15)/P, 2)
        for v in np.linspace(vmin, vmax, round((vmax-vmin)/0.01 + 1)):
            w0 = 0.0
            t0 = P*v/(R*(1.0/Ma + w0/Mw)) - 273.15
            t1 = t0
            w1 = w0
            for k in range(1, 5):
                w1 = (w1 + (Mw/Ma)/(P/pSatW(t1) - 1.0))/2.0
                t1 = P*v/(R*(1.0/Ma + w1/Mw)) - 273.15
            if t1 < tmax:
                t = np.linspace(t1,t0)
                w = Mw*P*v/(R*(t+273.15)) - Mw/Ma
                ax.plot(t, w, 'g', alpha=0.5)
            tc = (t0 + t1)/2.0
            wc = (w0 + w1)/2.0
            if tc > tmin and tc < tmax and wc < wmax:
                ax.text(tc, wc, "V = {0:0.2f} l/g".format(v), color = 'g', ha = 'center
, va = 'bottom',
                     rotation = 180.0*np.arctan2((w0-w1)/DY,(t0-t1)/DX)/np.pi)

    if Enthalpy:
        hmax = round(Cpair*(tmax-Tref) + wmax*(Hvap + Cpvap*(tmax-Tref)), -1)+10
        hmin = round(Cpair*(tmin-Tref), -1)-10
        for h in np.linspace(hmin, hmax, round((hmax-hmin)/10 + 1)):

            f = lambda t: (Mw/Ma)/(P/pSatW(t) - 1.0)
            g = lambda t: (h - Cpair*(t - Tref))/(Hvap + Cpvap*(t - Tref))

            t = -3 + fsolve(lambda t: f(t)-g(t), round((tmin + tmid)/2.0))

            t = np.linspace(t, tmax)
```

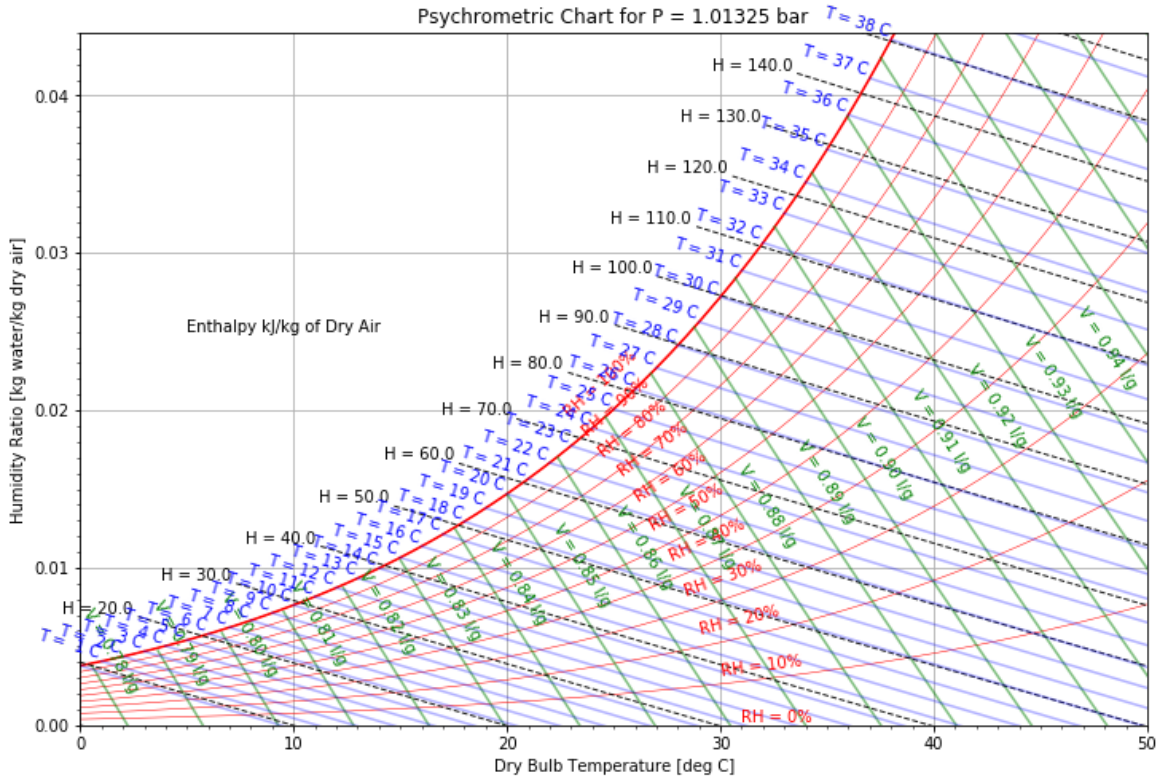
```
            w = (h - Cpair*(t-Tref))/(Hvap + Cpvap*(t-Tref))
            ax.plot(t, w, 'k--', linewidth=1, alpha=0.9)
            if (t[0] > tmin and t[0] < tmax and w[0] < 0.97*wmax):
                ax.text(t[0], w[0], "H = {0:0.1f} ".format(h), color = 'k', ha = 'right'
, va = 'bottom',
                        rotation = 0*180.0*np.arctan2(-Cpair/Hvap/DY,1.0/DX)/np.pi)

        hc = (hmax+hmin)/2.0
        tc = (tmid-tmin)*(hc-hmin)/(hmax-hmin) - 2.0
        wc = (hc - Cpair*(tc-Tref))/(Hvap + Cpvap*(tc-Tref))
        ax.text(tc-3, wc, 'Enthalpy kJ/kg of Dry Air',
             color = 'k', va = 'bottom', ha = 'right')

    if Wet_Bulb:
        for ts in np.linspace(tmin,round(tmax),round(tmax-tmin)+1):
            ws = (Mw/Ma)/(P/pSatW(ts) - 1.0)
            t = np.linspace(ts, tmax)
            num = Cpair*(ts-t) + ws*(Hvap + Cpvap*(ts-Tref) - Cpliq*(ts-Tref))
            den = Hvap + Cpvap*(t-Tref) - Cpliq*(ts-Tref)
            w = num/den
            ax.plot(t, w, 'b', alpha=0.3)

            if ts > tmin and ts < tmid:
                ax.text(ts,ws,"T = {0:-2.0f} C".format(ts), color = 'b', ha = 'right', '
a = 'bottom',
                        rotation = 180.0*np.arctan2(-Cpair/Hvap/DY,1.0/DX)/np.pi)

    plt.savefig('figures/PsychrometricChart.png')


psychrometric_chart()
```



Interactive Use with ipywidgets

In [26]:

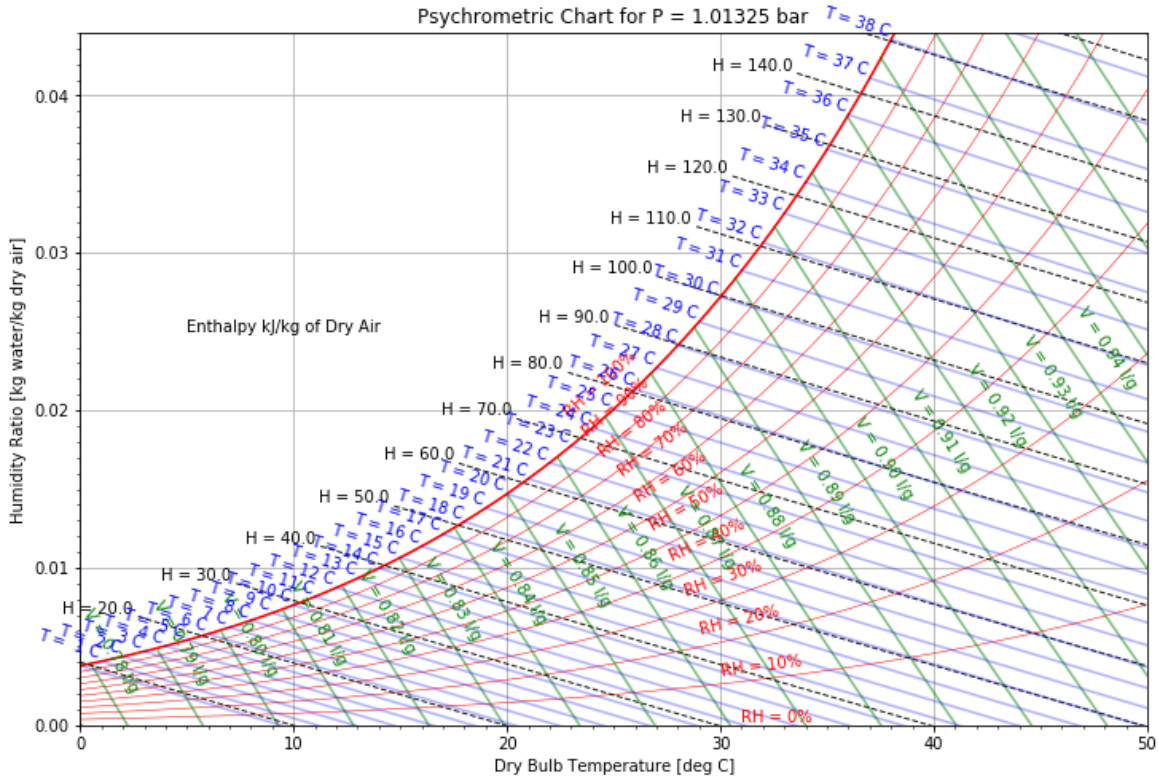
```
from ipywidgets import interact
interact(psychrometric_chart,
         P=(0.5, 3, 0.1),
         wmax = (0.02, 0.06, 0.005),
         tmin = (-20, 20, 2),
         tmax = (30, 60, 2));
```



Interactive Use with Google Colaboratory

In [28]:

```
P = 1.01325      # bar
wmax = 0.044     # max humidity range for plotting
tmin = 0.0       # min temperature for plotting
tmax = 50.0      # max temperature for plotting

psychrometric_chart(P, wmax, tmin, tmax)
```



References

1. http://www.vaisala.com/Vaisala%20Documents/Application%20notes/Humidity_Conversion_Formulas_B210973EN-F.pdf
2. W. Wagner and A. Pruß. The IAPWS Formulation 1995 for the Thermodynamic Properties of Ordinary Water Substance for General and Scientific Use. J. Phys. Chem. Ref. Data, Vol. 31, No. 2, 2002. http://thermophysics.ru/pdf_doc/IAPWS_1995.pdf

Example

Is WaterSeer, an Indiegogo project, plausible?

WaterSeer is an indiegogo project developed by students at UC Berkeley for the capture and condensation of water vapor from air. The goal of the project is to develop a practical device to augment water supplies in regions of the world without access to safe water.

In [6]:

```
from IPython.display import IFrame
IFrame('https://player.vimeo.com/video/182748120?title=0&byline=0&portrait=0',
width=640, height=360)
```

Out[6]:

Can you validate the claim that a device like this can potentially extract 37 liters of water per day in an arid or semi-arid region of the world? Assume an average day of 24°C with 70% relative humidity (a typical October day in Los Angeles). Assume the soil temperature is 54°F ≈ 12°C.

1. What is the Humidity Ratio of the incoming air? (i.e., water vapor content in units of kg water / kg dry air.)
2. What is the dew point of the incoming air?
3. How much water can be extracted per kilogram of dry air?
4. What volume of humid air must be treated to produce 37 liters of water per day? Does this seem plausible? If the pipe connecting the upper and lower part of the unit has an internal diameter of 6 inches, what is the average air speed? (Keep in mind that air must be routed in both directions.)
5. How much heat is released by condensation? Express your answer in watts. Where does the heat go? Think carefully about this one, and devise a way to discharge at least a portion of the heat in the exiting air.

Solution

Question 1. Using the psychrometric chart, the mixing ratio (i.e., humidity ratio) for a temperature 24 °C with 70% relative humidity is

$$w_{in} = 0.013 \text{ kg/kg dry air}$$

Question 2. Starting with humid air at 24°C with 70% relative humidity, the dew point is found by cooling air until the saturation curve is reached on the psychrometric chart. The dew point found is approximately 18°C.

Question 3. The condensor can operate at a temperature no lower than the soil temperature of 12 °C. At 12°C the humidity ratio

$$w_{out} = 0.009 \text{ kg/kg dry air}$$

The total amount of water that can be extracted is

$$w_{out} - w_{in} = 0.013 - 0.009 = 0.004 \text{ kg/kg dry air}$$

Question 4. Assuming a liquid density of 1 kg/liter, the requirement is to produce $\dot{m}_W = 37$ kg of water per day. From the mass balance, the unit must process

$$\dot{m}_A = \frac{\dot{m}_W}{w_{in} - w_{out}} = \frac{37}{0.013 - 0.009} = 9250 \text{ kg dry air / day}$$

From the psychrometric chart, the specific volume v_{in} is 0.855 cubic meters per kg dry air. The total volume of air to be treated is

$$\dot{V}_{in} = v_{in}\dot{m}_W = 0.855 \times 9250 = 7,909 \text{cu. m/day}$$

The cross sectional area of a 6 inch pipe is 182 sq. cm, of which half, 91 sq cm = 0.0091 sq m would be available for downflow to the condenser chamber. The average air speed would be 7,909/24/3600/0.0091 = 10 meters per second.

Question 5. The specific enthalpy of the entering air h_{in} is approximately 58 kJ/kg dry air. If the processed air exits at 12°C the specific enthalpy h_{out} would be 34 kJ/kg dry air. The specific enthalpy of the exiting water is $\hat{C}_{p,liq}(T - T_{ref})$

In [8]:

```
mw = 9250.0     # inlet flow of humid air, in kg dry air
win = 0.013     # kg water / kg dry air
wout = 0.009    # kg water / kg dry air
hin = 58.0      # kJ/kg dry air
hout = 34.0     # kJ/kg dry air  at 24 deg C
hliq = Cpliq*(12.0 - 0.0)

Q = mw*(hin - hout) - mw*(win-wout)*hliq
print("Condenser Heat Duty = ", Q, " kJ/day")
print("Condenser Heat Duty = ", Q/24/3600, " kJ/sec")
```

```
Condenser Heat Duty =  220143.192  kJ/day
Condenser Heat Duty =  2.547953611111111  kJ/sec
```

The condenser heat duty could be reduced by using the exiting air stream to cool incoming air. The maximum heat recovery corresponds to an exit air stream at the external temperature with specific enthalpy h_{out} of 46 kJ/kg dry air. In that case

In [9]:

```
mw = 9250.0     # inlet flow of humid air, in kg dry air
win = 0.013     # kg water / kg dry air
wout = 0.009    # kg water / kg dry air
hin = 58.0      # kJ/kg dry air
hout = 46.0     # kJ/kg dry air  at 24 deg C
hliq = Cpliq*(12.0 - 0.0)

Q = mw*(hin - hout) - mw*(win-wout)*hliq
print("Condenser Heat Duty = ", Q, " kJ/day")
print("Condenser Heat Duty = ", Q/24/3600, " kJ/sec")
```

```
Condenser Heat Duty =  109143.192  kJ/day
Condenser Heat Duty =  1.2632313888888889  kJ/sec
```

These numbers are high for a device with no external power supplies and to be built at very low cost. Of particular concern are the high gas flow rates which limit the opportunity for heat transfer, and the substantial condenser heat duty.

Exercises

Warmups

1. What is the dew point of air 20 °C at 50% relative humidity?
2. Normal human breathing warms air to body temperature, 37°C, saturated with water vapor. What is the water content of exhaled air? If inhalation and exhalation rates are the same, and inhaled air is at 20°C and a relative humidity of 40%, how much water is lost per 12 hours by normal breathing?
3. The makeup air supply for a commercial building must take exterior air at 30°C and 80% relative humidity and condition it for distribution at 20°C and 40% relative humidity. Show the steps needed for this operation on a psychrometric chart. How much energy is required per kilogram of dry air?

Laboratory Air Supply

A university educational science laboratory must supply fresh air at rate of 5 liters/sec/person. On a cool day in South Bend, Indiana, the outdoor air temperature is 0°C at 20% relative humidity. The air must be heated and conditioned to 21°C at 60% relative humidity before venting into the laboratory.

1. What is the mass flow of air into the laboratory (on a dry air basis)? What is the volumetric flow?
2. How much water must be added through humidification?
3. What is the energy requirement (in watts)?
4. Can you suggest means of recoving energy and water from the laboratory exhaust without compromising air quality? What is the maximum possible energy efficienty?

Snowmaking

Snowmaking equipment for ski resorts operate by spraying fine water droplets in a large flow of air. The droplet temperature is lowered by evaporative cooling and can be no higher than -2 degrees Celsius for effective snow making.

1. If the liquid water temperature is initially the same temperature as the air, what is the maximum air temperature for which snowmaking is possible?
2. Assuming the relative humidity is 30%, sketch a flow diagram. What should be the water flowrate per cubic meter of air?

8.6 Adiabatic Flame Temperature

Summary

This notebook demonstrates the calculation of the adiabatic flame temperature for the combustion of methane. The example is adapted from *Computational Methods for Engineers with MATLAB Applications* with permission of the author, James B. Riggs.

Problem Statement

The adiabatic flame temperature is the temperature that results when a combustible material is reacted with oxygen or air under the following conditions:

- The reaction is carried out without heat exchange with the environment (i.e., adiabatic conditions).
- All of the combustible material and the oxygen are consumed.

As a result, the adiabatic flame temperature represents the upper temperature limit of a combustion process. Using the following data, determine the adiabatic flame temperature of methane (CH_4) burned in air where the reactants are initially at 25 deg C and 1 atm pressure.

Solution

The combustion of methane proceeds according to the balanced reaction

$$CH_4 + 3O_2 \longrightarrow CO_2 + 2H_2O$$

with a heat of combustion of -890.4 kJ/gmol. For this calculation we choose a basis of 1 gmol of CH_4 because no amount of reactant or product was specified. Next, we apply material balances. Because all of the CH_4 is reacted and no excess O_2 is used, the mole balances are relatively simple. From the reaction equation, one mole of CO_2 and two moles of H_2O are formed from the complete combustion of 1 gmol of CH_4 and the N_2 in the air used leaves in the product gas. Because 3 moles of O_2 are required to consume one mole of CH4 and air is approximately 79 mol% N_2, 11.2857 moles of N_2 (i.e., (3/0.21) 0.79) leave in the product gas.

In [ ]:

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

n = dict()
n['CO2'] = 1
n['H2O'] = 2
n['N2'] = 3*(.79/0.21)

for k in n.keys():
    print("n[{:3s}] =  {:5.2f}".format(k,n[k]))
```

Now consider the energy balance for this system. There is no external work being done on the system, and because it is adiabatic there is no external heat being produced or added to the system. The reactants enter at 25 C and 1 atm, which we use as the reference state for computing changes in specific energy. The system is operated at constant pressure, so the energy balance can be written as an enthalpy balance

$$0 = \Delta H_{Rxn} + \Delta H_{Prod}(T)$$

where the heat liberated by reaction goes into raising the temperature of the product gases. The adiabatic flame temperature is found by solving the enthalpy balance for T.

The basis is 1 gmol of CH_4 and all of it reacts, the heat of reaction is simply -890,400 J.

In [ ]:

```
DeltaH_Rxn = -890400
```

The enthalpy change of the product gases is given by

$$\Delta H_{Prod}(T) = \int_{25}^{T} \left(n_{CO_2} C_{p,CO_2(g)}(T) + n_{H_2O} C_{p,H_2O(g)}(T) + n_{N_2} C_{p,N_2(g)}(T) \right) dT$$

For computational purposes, the temperature dependent heat capacities are represented by anonymous functions. $\Delta H_{Prod}(T)$ is then a function which integrates the heat capacities for a given temperature.

In [ ]:

```
Cp = dict()
Cp['CO2','g'] = lambda T: 36.11 + 4.233e-2*T - 2.887e-5*T**2 + 7.464e-9*T**3
Cp['H2O','g'] = lambda T: 33.46 + 0.688e-2*T + 0.7604e-5*T**2 - 3.593e-9*T**3
Cp['N2','g']  = lambda T: 29.00 + 0.2199e-2*T + 0.5723e-5*T**2 - 2.871e-9*T**3

from scipy import integrate

def DeltaH_Prod(T):
    h1,err = integrate.quad(Cp['CO2','g'],25,T)
    h2,err = integrate.quad(Cp['H2O','g'],25,T)
    h3,err = integrate.quad(Cp['N2','g'],25,T)
    return n['CO2']*h1 + n['H2O']*h2 + n['N2']*h3
```

The adiabatic flame temperature is found by finding the temperature for which the right-hand side of the enthalpy balance has a value of zero.

In [ ]:

```
f = lambda T: DeltaH_Rxn + DeltaH_Prod(T)
```

In [ ]:

```
T = np.linspace(25,3000,200)
plt.plot(T,[f(T) for T in T],[25,3000],[0,0])
plt.xlabel('Temperature [degrees C]')
plt.ylabel('Net Enthalpy [Joules/gmol]');
```

The adiabatic flame temperature is found by using a root-finding algorithm to find a root to the enthalpy balance equation.

In [ ]:

```
from scipy.optimize import brentq

T = brentq(f,25,3000)
print('The adiabatic flame temperature is {:6.1f} degrees C.'.format(T))
```

In [ ]:

```

```

8.7 Torpedo Propulsion

Summary

This notebook demonstrates a variety of energy calculations analyzing the design requirements for propulsion systems of naval torpedos.

Background

Torpedo technology contributes to the security and safety of naval forces. In this notebook we explore the extreme requirements and design alternatives for torpedo propulsion.

"MK46 torpedo launch" by United States Navy, Mass Communication Specialist John L. Beeman - Navy NewsStand Photo ID: 070412-N-9851B-007 Navy NewsStand Home. Licensed under Public Domain via Wikimedia Commons.

Torpedo propulsion systems must operate in the absence of atmospheric oxygen for an oxidizer, provide the power necessary for fast and deep dives, be stable in storage and safe in marine and airborne environments. In this note we'll look at

1. Calculating the energy and power required to for torpedo propulsion.
2. A monopropellant chemistry powering a hot-gas piston engine in the Mk 46 and Mk 48 torpedos.
3. A lithum/sulfur hexaflouride chemistry powering a self-contained Rankine cycle engine for the Mk 50 where the combustion products are more dense than the reactants.
4. An AgO/Al battery system powering an electrically driven torpedo.

Power and Energy Requirements

The drag force on a torpedo is given by the formula

$$F_D = \frac{1}{2}\rho v^2 C_D A$$

where C_D is the drag coefficient, ρ is the density of the fluid, v is velocity, and A is the cross-sectional area. Assume C_D = 0.095 for a well-designed torpedo.

The Mk-48 ADCAP, which is 21 inches in diameter and a mass of 1,663 kg, is claimed to have a range of 42,530 yards at 55 knots.

- What is the power requirement in kw and horsepower at these speeds?
- How much energy is required to accelerate the torpedo to its operating speed?
- How much energy is required to cover the range of operation?
- Assuming the propulsion system has an overall efficiency of 25%, how much fuel energy is required?

Hydrodynamic Power Requirement

In [1]:

```
%matplotlib inline
import numpy as np

# One knot = 0.51444 meters per sec
v = 55*0.51444                # m/s

# One inch = 0.0254 meters
A = 0.25*np.pi*(21*0.0254)**2     # m**2

Cd = 0.095
rho = 1025.0                  # kg/m**3 (sea water)
Fd = 0.5*rho*v**2*Cd*A        # Newtons

print("Drag force at 55 knots = {:6.1f} Newtons".format(Fd))

W = v*Fd                      # Joules

print("Power requirement at 55 knots = {:6.1f} kW".format(W/1000.0))
print("Power requirement at 55 knots = {:6.1f} Hp".format(W/745.7))
```

```
Drag force at 55 knots = 8709.8 Newtons
Power requirement at 55 knots =  246.4 kW
Power requirement at 55 knots =  330.5 Hp
```

Energy to Accelerate to Operating Speed

In [2]:

```
m = 1663.0        # kg
v = 55*0.51444    # m/s
Ea = 0.5*m*v**2   # Joules

print("Energy required to accelerate to 55 knots= {:6.1f} kJ".format(Ea/1000.0))
```

```
Energy required to accelerate to 55 knots=  665.7 kJ
```

Energy to Reach Terminal Range

In [3]:

```
d = 42530*0.9144        # meters
v = 55*0.51444          # m/s
t = d/v                 # seconds to range
Er = W*t                # energy required

print("Time required for range = {:6.1f} sec".format(t))
print("Energy required for range = {:7.1f} kJ".format(Er/1000))
```

```
Time required for range = 1374.5 sec
Energy required for range = 338719.2 kJ
```

Fuel Energy Requirement

In [4]:

```
E_fuel = (Er + Ea)/0.25/1000.0      # kJ
print("Fuel Energy Requirment at 25% eff. = {:6.1f} kJ".format(E_fuel))
```

```
Fuel Energy Requirment at 25% eff. = 1357539.5 kJ
```

Monopropellant Propulsion for the Mk 48 Torpedo

"Mk 48 torpedo maintenance 1982" by United States Navy - U.S. Defenselmagery photo VIRIN: DN-SC-86-00553. Licensed under Public Domain via Wikimedia Commons.

The Mk 48 and torpedos use a monopropellant called Otto Fuel II to drive a hot-gas piston engine. The principle component of Otto fuel is propylene glycol dinitrate (PGDN, also called by its IUPAC name 1,2-propanediol dinitrate) that makes up 76% of the fuel by weight. Assuming the reaction proceeds as

$C_3H_6N_2O_6 \rightarrow 3\ CO_2 + 3\ NH_3$

- What is the specific energy content measured as kJ/kg of fuel at standard conditions?
- The Mk 48 torpedo has an estimated maximum operating depth of 800m. What is specific energy content of the fuel at the pressure encountered at that depth?
- How much fuel will be required by the Mk 48 ADCAP?

Specific Enthalpy of Reaction

The density of Otto fuel is 1.232 g/ml. The thermochemical data for 1,2-propanediol dinitrate is available from the NIST Webbook.

In [5]:

```
mw = dict()
mw['PGDN'] = 166.0895    # g/gmol

Hf = dict()
Hf['PGDN'] = -296.9      # kJ/gmol
Hf['CO2'] = -393.5
Hf['NH3'] = -45.94

dH = 3*Hf['CO2'] + 3*Hf['NH3'] - Hf['PGDN']
print("Molar Heat of Reaction = {:6.1f} kJ/gmol PGDN".format(dH))

dH_fuel = 0.76*1000*dH/mw['PGDN']
print("Heat of Reaction = {:6.1f} kJ/kg fuel".format(dH_fuel))
```

```
Molar Heat of Reaction = -1021.4 kJ/gmol PGDN
Heat of Reaction = -4673.9 kJ/kg fuel
```

Specific Enthalpy of Reaction at 800m

In [6]:

```
# volume of fuel per kg-mol of PGDN
rho = 1232.0                        # kg/m**3
gmols = rho*0.76*1000.0/mw['PGDN'] # gmols/m**3
Vhat = 1.0/gmols                    # m**3/gmol

Patm = 101325.0                     # N/m**2
P800 = Patm + 800.0*1025.0*9.81     # N/m**2
R = 8.314                           # m**3Pa/K/gmol
T = 298.15                          # standard temperature

Hf['PGDN'] = -296.9 + Vhat*(P800-Patm)/1000
Hf['CO2'] = -393.5 + R*T*np.log(P800/Patm)/1000
Hf['NH3'] = -45.94 + R*T*np.log(P800/Patm)/1000

dH = 3*Hf['CO2'] + 3*Hf['NH3'] - Hf['PGDN']
print("Molar Heat of Reaction = {:6.1f} kJ/gmol PGDN".format(dH))

dH_fuel_800 = 0.76*1000*dH/mw['PGDN']
print("Heat of Reaction = {:6.1f} kJ/kg fuel".format(dH_fuel_800))
```

```
Molar Heat of Reaction = -957.6 kJ/gmol PGDN
Heat of Reaction = -4381.8 kJ/kg fuel
```

Fuel Requirements

In [7]:

```
mass_fuel = E_fuel/(-dH_fuel)
print("Otto Fuel Required = {:6.1f} kg".format(mass_fuel))
```

```
Otto Fuel Required =  290.5 kg
```

We can also try and estimate the fuel capacity from available diagrams of the Mk 48 torpedo.

In [8]:

```
L = (400.0/1951.0)*5.86          # length of fuel tank in m
A = 0.25*np.pi*(0.50)**2            # cross-sectional area in m
V = A*L*1000.0                   # volume in liters
rho = 1.232                      # Otto fuel density in g/ml

print("Maximum fuel capacity = {:6.1f} liters".format(V))
print("Maximum fuel capacity = {:6.1f} kg".format(V*rho))
```

```
Maximum fuel capacity =  235.9 liters
Maximum fuel capacity =  290.6 kg
```

Second Estimate of Fuel Requirements

Compute the engine work requirement assuming an 85% propulsion system efficiency, and a motor conversion efficiency of 88% (same as a Tesla roadster).

In [9]:

```
W_engine = (Ea+Er)/0.85/0.88
```

Estimate the adiabatic combustion temperature assuming that 76% of the fuel mass is available for a sensible temperature increase.

In [10]:

```
Cp = dict()
Cp['CO2'] = 35.6            # J/K/gmol
Cp['NH3'] = 37.0            # J/K/gmol

Tcomb = 298.15 - 0.76*dH*1000.0/(3*Cp['CO2'] + 3*Cp['NH3'])
print("Estimated Combustion Temperature = {:6.1f} K".format(Tcomb))
```

```
Estimated Combustion Temperature = 3639.6 K
```

Assume that work is extracted by a 12-to-1 adiabatic expansion of the combustion gases in a hot-gas piston engine, compute the moles of hot gas required.

In [11]:

```
R = 8.314                    #  m**3 Pa/K/gmol
gamma = (0.52+0.21)/(0.4 + 0.16)
alpha = 1.0/(gamma-1)
Vr = 1.0 - 12.0**(1-gamma)
n_gas = W_engine/alpha/8.314/Tcomb/Vr
print("gram-moles of hot gas required = {:6.1f}".format(n_gas))
```

```
gram-moles of hot gas required = 8593.5
```

Assume the fuel additives have the same average molecular weight as the ammonia/carbon dioxide mixture.

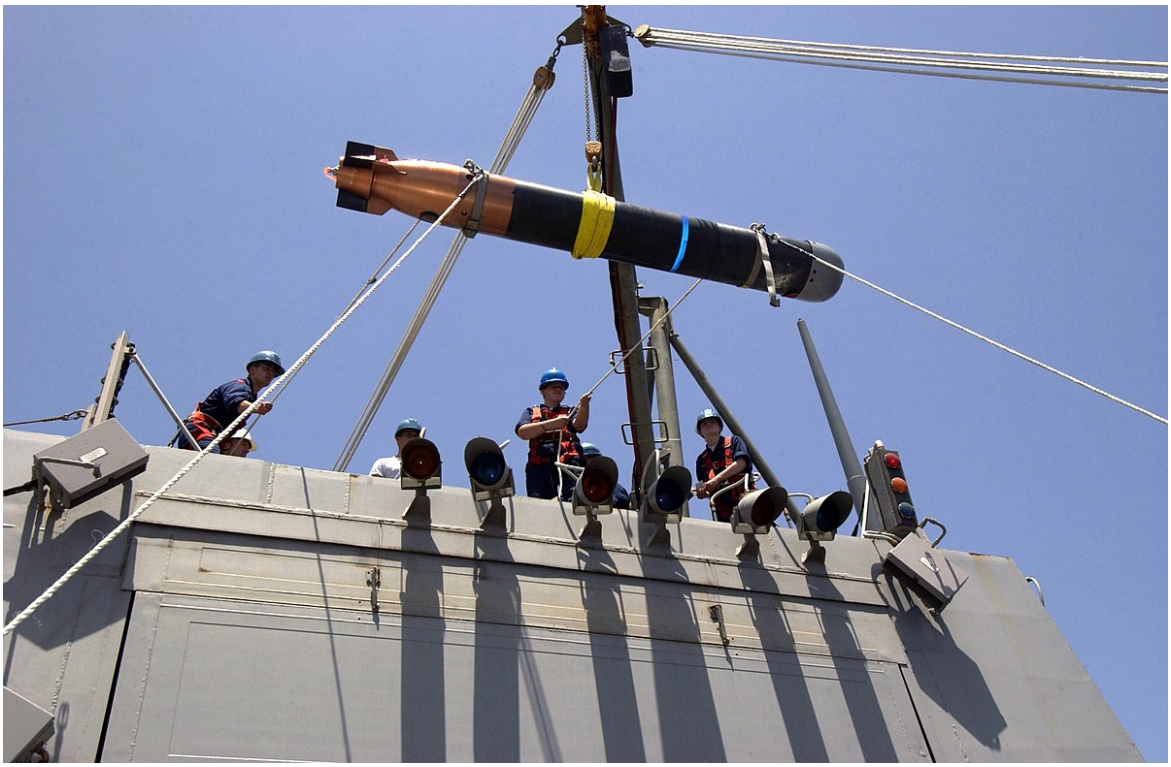
In [12]:

```
n_pgdn = (0.76*n_gas)/6.0

mass_pgdn = mw['PGDN']*n_pgdn
mass_otto = mass_pgdn/0.76/1000.0

print("Otto fuel required = {:6.1f} kg".format(mass_otto))
```

```
Otto fuel required =  237.9 kg
```

Mk 50 Torpedo

The Mk 50 torpedo was designed a lightweight device that could be used against fast and deep-diving submarines. It is a much smaller device only 9.5 ft long, 12.75 inches in diameter, and 800 pounds. The small size allows it to launched from a variety of platforms, including ships, airplanes, and helicopters.



"US Navy 040623-N-5319A-001 Sailors assigned to the weapons department aboard the guided missile destroyer USS Bulkeley (DDG 84), hoist an Anti-Submarine Warfare (ASW) MK-50 Torpedo off of the flight deck" by U.S. Navy photo by Photographer's Mate 1st Class Brien Aho - This Image was released by the United States Navy with the ID 040623-N-5319A-001. Licensed under Public Domain via Wikimedia Commons.

A critical part of its design is the Stored Chemical Energy Propulsion System (SCEPS) in which sulfur hexaflouride gas is sprayed onto a solid block of lithium to produce the heat necessary to run a steam-powered, closed-cycle rankine engine. Sulfer hexaflouride has a number of interesting properties which are demonstrated in these videos.

In [13]:

```
from IPython.display import YouTubeVideo
YouTubeVideo("d-XbjFn3aqE",420,315,rel=0)
```

Out[13]:

Sulfur hexaflouride is relatively inert but reacts with lithium at an operating temperature of about 790 °C via the stoichiometric reaction

8 Li + SF_6 → Li_2S + 6 LiF

This reaction produces no gaseous products which allows the propulsion system to be completely self-contained - an enormous design advantage for a torpedo that must operate under high pressure conditions in deep ocean waters.

- Estimate the heat of reaction at the normal operating conditions.
- What is the heat of reaction per unit mass of reactants? Per unit volume? Compare to the lower heating values of some common fuels.
- The heat is used to generate steam for a closed cycle Rankine engine employing a steam turbine. The creep limit of stainless steel limits the inlet temperature of the steam turbine to 565 °C. If the condenser temperature is 30 °C, what is the maximum acheivable Carnot efficiency of the steam engine?
- An alternative formula for estimating the heat engine efficiency under maximum power conditions is the Curzon-Ahlborn efficiency given by $\eta = 1 - \sqrt{\frac{T_{cold}}{T_{hot}}}$. Estimate the Curzon-Ahlborn efficiency of this steam engine.
- The Mk 50 torpedo has a diameter of 12.75 inches, an estimated speed of 50 knots, and an estimated range of 15 km. Assuming a drag coefficient 0.10, an engine efficiency equal to the Curzon-Ahlborn efficiency, and an overall propulsion system efficiency of 85%, how much fuel is required?

The core issue for this problem is finding reliable thermodynamic information. Here's one source (http://www.update.uu.se/~jolkkonen/pdf/CRC_TD.pdf) that you may find useful in addition to the NIST Webbook.

Heat of Reaction

In [14]:

```
Hf = dict()    # kJ/gmol
Cp = dict()    # J/K/gmol

Hf['Li'] = 0.0
Cp['Li'] = 24.8

Hf['SF6'] = -1220.5
Cp['SF6'] = 97.0

Hf['Li2S'] = -441.4
Cp['Li2S'] = 54.1           # Missing Data, use Li2O

Hf['LiF'] = -616.0
Cp['LiF'] = 41.6

Hr = Hf['Li2S'] + Cp['Li2S']*(25-790)/1000.0 \
     + 6*(Hf['LiF'] + Cp['LiF']*(25-790)/1000.0) \
     - 8*Hf['Li'] - Hf['SF6']

print("Heat of Reaction at 790 deg C = {:6.1f} kJ/gmol".format(Hr))
```

```
Heat of Reaction at 790 deg C = -3149.2 kJ/gmol
```

Specific Heat of Reaction

In [15]:

```
mw = dict()
mw['Li'] = 6.941
mw['SF6'] = 146.06

dH = Hr*1000.0/(8*mw['Li'] + mw['SF6'])
print("Heat of Reaction at 790 deg C = {:6.1f} kJ/kg".format(dH))
```

```
Heat of Reaction at 790 deg C = -15622.1 kJ/kg
```

Carnot Efficiency

In [16]:

```
eta_carnot = 1 - (30+273.15)/(565+273.15)
print("Carnot efficiency = {:6.3f}".format(eta_carnot))
```

```
Carnot efficiency =  0.638
```

Curzon-Ahlborn Efficiency

In [17]:

```
eta_ca = 1 - np.sqrt((30+273.15)/(565+273.15))
print("Curzon-Ahlborn efficiency = {:6.3f}".format(eta_ca))
```

```
Curzon-Ahlborn efficiency =  0.399
```

Fuel Requirement

In [18]:

```
A = 0.25*np.pi*(12.75*0.0254)**2      # area in m**2
rho = 1025.0                          # kg/m**3
v = 50*0.5144                         # m/s
Cd = 0.10
Fd = 0.5*rho*Cd*A*v**2                # drag force in Newtons

print("Drag force at 50 knots = {:6.1f} Newtons".format(Fd))

W = v*Fd

print("Power requirement at 50 knots = {:6.1f} Watts".format(W))
print("Power requirement at 50 knots = {:6.1f} Hp".format(W/745.7))

t = 15000.0/v                         # time in sec
E = W*t/1000.0                        # energy required in kJ

print("Hydrodynamic Energy requirement at 50 knots = {:6.1f} kJ".format(E))

E_fuel = E/0.85/eta_ca

print("Fuel Energy requirement at 50 knots = {:6.1f} kJ".format(E_fuel))

print("Fuel Requirement at 50 knots = {:6.1f} kg".format(E_fuel/(-dH)))
w_Li = 8*mw['Li']/(8*mw['Li'] + mw['SF6'])
print("Li Requirement = {:6.2f} kg".format(w_Li*E_fuel/(-dH)))
print("SF6 Requirement = {:6.2f} kg".format((1-w_Li)*E_fuel/(-dH)))
```

```
Drag force at 50 knots = 2792.6 Newtons
Power requirement at 50 knots = 71826.5 Watts
Power requirement at 50 knots =   96.3 Hp
Hydrodynamic Energy requirement at 50 knots = 41889.5 kJ
Fuel Energy requirement at 50 knots = 123638.9 kJ
Fuel Requirement at 50 knots =    7.9 kg
Li Requirement =   2.18 kg
SF6 Requirement =   5.73 kg
```

The Black Shark Electric Torpedo

The Black Shark torpedo is an electrically powered, 21 inch diameter torpedo produced by Whitehead Sistemi Subacquei (WASS) group of Italy. It has an estimated range of 22 km at a speed of 52 knots.

<img src = "http://www.naval.com.br/blog/wp-content/uploads/2009/02/al-torpedos.jpg"/ style = "height:600px">

The Black Shark torpedo employs a salt water activated [AgO-Al battery[(http://www.saftbatteries.com/force_download/Torpedo_brochure09_13.pdf) The anode half-reaction is

2 Al + 8 OH^- → 2 $Al0_2^-$ + 4 H_2O + 6 electrons

The OH^- is produced by the dissolution of NaOH when then the torpedo enters the water. The cathode half-reaction is

3 AgO + 3 H_2O + 6 electrons → 3 Ag + 6 OH^-

for an overall electrochemical reaction

2 Al + 3 AgO + 2 OH^- → 2 AlO_2^- + 3 AgO + H_2O

An individual battery cell has a measured potential of E_{cell} = 1.33 volts under operating conditions. The Gibb's free energy ΔG, is related to the number of electrons transferred in the reaction, n, and the cell potential by

$$\Delta G = -nFE_{cell}$$

where $F = 96,485$ Columbs/gmol is the charge on one gram-mole of electrons.

Assuming the battery/motor system has an overall efficiency of a Tesla roadster (88% efficient), and the propulsion system is 85% efficient, estimate the chemical reactants necessary to power the Black Shark torpedo.

Gibb's Free Energy

In [19]:

```
dG = -6.0*96485.0*1.33/1000.0
print("Gibb's Free Energy = {:6.1f} kJ".format(dG))

mw['Al'] = 26.98
mw['AgO'] = 231.735
mw['NaOH'] = 39.997

dG = 1000.0*dG/(2*mw['Al'] + 3*mw['AgO'] + 2*mw['NaOH'])
print("Gibb's Free Energy = {:6.1f} kJ/kg".format(dG))
```

```
Gibb's Free Energy = -770.0 kJ
Gibb's Free Energy = -928.6 kJ/kg
```

Energy Requirement

In [20]:

```
A = 0.25*np.pi*(21.0*0.0254)**2      # area in m**2
rho = 1025.0                         # kg/m**3
v = 52*0.5144                        # m/s
Cd = 0.095
Fd = 0.5*rho*Cd*A*v**2               # drag force in Newtons

print("Drag force at 52 knots = {:6.1f} Newtons".format(Fd))

W = v*Fd

print("Power requirement at 52 knots = {:6.1f} Watts".format(W))
print("Power requirement at 52 knots = {:6.1f} Hp".format(W/745.7))

t = 22000.0/v                        # time in sec
E = W*t/1000.0                       # energy required in kJ

print("Hydrodynamic Energy requirement at 52 knots = {:6.1f} kJ".format(E))
```

```
Drag force at 52 knots = 7784.3 Newtons
Power requirement at 52 knots = 208221.8 Watts
Power requirement at 52 knots =  279.2 Hp
Hydrodynamic Energy requirement at 52 knots = 171255.6 kJ
```

Battery Requirement

In [21]:

```
E_battery = E/0.85/0.88

print("Battery Energy requirement at 52 knots = {:6.1f} kJ".format(E_battery))

print("Battery Reactant Requirements at 52 knots = {:6.1f}
kg".format(E_battery/(-dG)))
```

```
Battery Energy requirement at 52 knots = 228951.3 kJ
Battery Reactant Requirements at 52 knots =  246.6 kg
```

In [ ]:

```

```

Appendix A. Products: Product Design and Analysis

A.1 Diesel Engine Emissions Control

Summary

This Jupyter notebook demonstrates an analysis of NO_x emissions control in diesel engines.

Background

Diesel engines offer greater energy efficiency than equivelently sized gasoline engines by operating at higher compression and combustion temperatures. The problem, however, is that diesel engines produce more particulates because the fuel is injected late in the compression cycle, and produces greater NO_x emissions because of the higher combustion temperatures.

Here we look into a particular technology for treating NO_x that has found wide spread use in heavy trucks and has been recently introduced in the automobile and light truck marketplace.

In [1]:

```
from IPython.display import YouTubeVideo
YouTubeVideo("uQHvi2Lgnac",420,315,rel=0)
```

Out[1]:

The Mercedes-Benz “BlueTEC” system, for example, comprises a selective catalytic reduction (SCR) converter and the AdBlue reagent. Adblue is a registered trademark for AUS32 (Aqueous Urea Solution 32.5%) consisting of 32.5 wt% Urea in deionized water with a density of 1.09 g/ml. The same solution can be purchased from many sources under the generic name Diesel exhaust fluid (DEF)

Problem 1

An average chemical formula for diesel fuel is $C_{12}H_{23}$ with a density of 0.832 kg/liter. A typical diesel light truck exhibits gets about 21 mpg on the highway. Assuming air (which is a mixture of 21 mol% O_2 and 79 mol% N_2) is mixed with diesel fuel at the stoichiometric ratio. Assume complete combustion of the diesel fuel producing CO_2, H_2O. What is the exhaust gas flowrate in units of g/mile?

Hint: The purpose of this first calculation is obtain an estimate for the exhaust gas flowrate. Because NO_x is a trace component of the exhaust, at this stage it is not necessary to consider any additional reactions other than combustion of $C_{12}H_{23}$.

Problem 2

DEF is injected into a hot, post-combustion exhaust stream where the water evaporates and the urea forms solid droplets

$CO(NH_2)_2(aq) \rightarrow CO(NH_2)_2(s)$

The urea immediately heats up and undergoes thermal decomposition to form ammonia and isocyanic acid

$CO(NH_2)_2(s) \rightarrow NH_3(g) + HNCO(g)$

The isocyanic acid is stable in the gas phase. The hot gas stream is passed over a solid oxide catalyst to hydrolyze the isocyanic acid to form additional ammonia.

$HNCO(g) + H_2O(g) \rightarrow NH_3(g) + CO_2(g)$

Using a generation/consumption analysis, what is the net reaction?

Problem 3

Following hydrolysis of the urea, the hot exhaust then passes over the selective reduction catalyst that promotes reaction between ammonia and the NO_x:

$4\ NH_3 + 4\ NO + O_2 \rightarrow 4\ N_2 + 6\ H_2O$

$4\ NH_3 + 2\ NO + 2\ NO_2 \rightarrow 4\ N_2 + 6\ H_2O$

Excess oxygen is present in the exhaust stream under normal conditions for most diesel engines. Assume the diesel exhaust initially contains 200 ppm by mass NO and 50 ppm by mass NO_2. Field measurements demonstrate that 95% of each is converted to N_2. What are the extents of reaction for each reaction?

Hint: ppm means 'parts per million' by mass For example, to compute the mass flow of NO, take $(200 \times 10^{-6}) \times \dot{m}_{exhaust}$ where $\dot{m}_{exhaust}$ is the mass flow computed in part 1 of the homework set.

Problem 4

The commerical route to urea is to produce ammonia from natural gas, then convert the ammonia to urea. Ammonia production is a two step process in which hydrogen produced by the steam reforming of natural gas followed by water-gas shift

$CH_4 + H_2O \rightarrow CO + 3\ H_2$

$CO + H_2O \rightarrow CO_2 + H_2$

followed by the Haber-Bosch process to produce ammonia.

$N_2 + 3\ H_2 \rightarrow 2\ NH_3$

The Bosch-Meiser process for urea production involves two additional steps. Ammonia is combined with dry ice to produce ammonium carbamate and water

$2\ NH_3 + CO_2 \rightarrow H_2NCOONH_4$

which decomposes to form urea and water

$H_2NCOONH_4 \rightarrow CO(NH_2)_2 + H_2O$

For an integrated urea production facility, what is the overall stoichiometry for the production of urea from natural gas?

Problem 5

In a review published by Consumer Reports on January 4, 2010, regarding the Mercedes-Benz GL320

The total bill just for adding AdBlue? A stunning 316.99 dollars. We were down to 18% full on the additive at 16,566 miles. It took 7.5 gallons to fill the tank, costing an eye-opening 241.50 dollars for the fluid alone. The labor to add the fluid plus tax accounted for the rest. None of this was covered by the warranty." Consumer Reports, January 4, 2010.

(a) Using the results you found above, how much Adblue would you estimate is needed to drive 16,566 miles assuming a vehicle gets 21 mpg? How does this compare to the numbers quoted in the review? What assumption would you need to revise in order to match the observed consumption of Adblue?

(b) Current spot prices for bulk urea can be found on-line. What is the cost of the raw materials needed to produce one gallon of Adblue? How does this compare to the price of Adblue implied by the story in Consumer Reports?

A.2 Pyrotechnic Design for Airbags

Summary

This notebook introduces the analysis and design of the chemistry underlying the automotive airbags.

Background

Automobile airbags are designed to protect occupants in the event of front and side impact crashes, and have been mandated in all new cars since 1998. The following video provides background on the chemistry and engineering challenges involved in the design of modern airbags.

In [2]:

```
from IPython.display import YouTubeVideo
YouTubeVideo("9vynOdF61aM",560,315,rel=0)
```

Out[2]:

In older technologies, airbags were inflated with nitrogen generated by the explosive decomposition of sodium azide.

$2\ NaN_3 \rightarrow 2\ Na + 3\ N_2\ (g)$

Highly reactive sodium would be a hazard at a crash site. A clever piece of chemical engineering is to add potassium nitrate (KNO_3) to oxidize the sodium by the reaction

$10\ Na + 2\ KNO_3 \rightarrow K_2O + 5\ Na_2O + N_2\ (g)$

K_2O and Na_2O are also highly reactive. In the presence of water, which is common near crash sites, these compounds would react to form KOH and NaOH, both of which would be hazardous to passengers and emergency personnel. So another clever piece of engineering is to add silica (SiO_2) to the airbag which will react with K_2O and Na_2O to form silicate glass though the reactions

$K_2O + SiO_2 \rightarrow K_2O_3Si$ (potassium silicate glass)

$Na_2O + SiO_2 \rightarrow Na_2O_3Si$ (sodium silicate glass)

With proper design of the initial charge one can avoid the net production of sodium with the added benefit of generating further nitrogen.

Exercises

Generation/Consumption Analysis

Perform a generation consumption analysis to determine a process stoichiometry that avoids the net production of Na, Na_2O, or K_2O. Show your work, including the stoichiometric matrix and the final net stoichiometry.

Design of the Reactant Mixture

Approximately 98 grams of N_2 are needed to fill a typical airbag. Calculate (in grams) the required amount of sodium azide, potassium nitrate, and silica to be loaded into the airbag.

Design of an Alternative Product using guanidine nitrate

As mentioned in the video, newer airbag technologies replace sodium azide with other pyrotechnic agents, such as guanidine nitrate. One example from the vast patent literature cites mixture of 45.35 wt% ammonium nitrate (NH_4NO_3), 8.0 wt% potassium nitrate (KNO_3), and 46.65 wt% guanidine nitrate ($C(NH_2)_3NO_3$).

Species	MW	wt%
NH_4NO_3	80.05	45.35
KNO_3	101.1	8.0
$C(NH_2)_3NO_3$	122.1	46.65

Assume

- the gaseous reaction products are carbon dioxide (CO_2), nitrogen (N_2), and water (H_2O), and
- KNO_3 reacts to form K_2O, and
- silica (SiO_2) will be added to produce a silicate glass.

How much of the mixture will be required to produce the same gas volume as in problem 2?

Hint

The problem does not provide a list of reactions. But we can make some inferences based on knowledge of the reaction products. The first step is to consider the decomposition of the reactants to reaction products plus elemental species.

$NH_4NO_3 \rightarrow N_2 + 2\ H_2O + 1/2\ O_2$

$C(NH_2)_3NO_3 + O_2 \rightarrow 2\ N_2 + CO_2 + 3\ H_2O$

$2\ KNO_3 \rightarrow K_2O + N_2 + 5/2\ O_2$

Since oxygen doesn't appear in the list of reactants (and, in any event, would certainly be completely consumed in the course of the pyrotechnic combustion), we'll combine these three reactions into two:

R_1: $C(NH_2)_3NO_3 + 2\ NH_4NO_3 \rightarrow 4\ N_2 + CO_2 + 7\ H_2O$

R_2: $5\ C(NH_2)_3NO_3 + 4\ KNO_3 \rightarrow 2\ K_2O + 12\ N_2 + 5\ CO_2 + 15\ H_2O$

The third reaction to consider is

R_3: $K_2O + SiO_2 \rightarrow K_2O_3Si$ (potassium silicate glass)

The way to proceed with this problem is to, first, choose a basis of, say, 100 grams of pyrotechnic charge. What will be the extents of each reaction? How would you scale up or down the charge?

Alternative Chemistries

At about the 2:40 mark in the above video, the narrator refers to alternative chemistries for gas generation combining guanidine nitrate (C(NH2)3NO3) as a fuel, and metallic compounds such as iron oxide (Fe2O3), molybdenum disulfide, or cupric oxide (CuO) as oxydizers. For each oxydizer, propose a balanced reaction showing

- the maximum amount of gas that can be generated per gram of charge, and
- the approximate temperature of the gas mixture following expansion.

Among these options, which would seem most appropriate for its intended application to airbags?

In [ ]:

A.3 Flameless Cooking

Summary

This notebook outlines a project for the design of a flameless cooking product. The project requires use of generation/consumption analysis, mass and energy balances,and creative thought about product design.

Overview

Recently, a number of vendors, such as MagicCook, BaroCook, and MealSpec have developed portable devices for cooking single servings of hot food that don't require flame or electricity. These devices could be used by backpackers, military, or workers seeking a hot meal in a location without access to traditional cooking appliances. The goal of this project is to analyze the performance of these devices, and to engage your creativity in the design of new products.

In [2]:

```
from IPython.display import YouTubeVideo
YouTubeVideo("Coz7LRoTnYM",560,315)
```

Out[2]:

The most widely used device of this kind is the Flameless Ration Heater (FRE) used by the U.S. Army to heat single serving Meals Ready-to-Eat (MRE) food rations. The history of its development provides insight into the many considerations and tradeoffs incorporated into a product of this type. The patented product consists of 7.5 grams of a magnesium/iron powder alloy in a 95/5 wt% ratio

Understandably, none of these companies is forthcoming with the chemistry of their heating packs. One reviewer suggests the MealSpec heating pack contains calcium oxide (CaO, also known as quicklime) and aluminum (Al) powder which react in the presence of water.

There are a number of reactions that have been proposed for these applications. Among the most well known is the hydration of quicklime via the exothermic reaction

$$CaO(s) + H_2O(l) \longrightarrow Ca(OH)_2(aq)$$

Aluminum can react with water in various ways to produce by-product hydrogen gas

$$Al + 3H_2O \longrightarrow Al(OH)_3 + \frac{3}{2}H_2$$

$$Al + 2H_2O \longrightarrow AlOOH + \frac{3}{2}H_2$$

$$Al + \frac{3}{2}H_2O \longrightarrow \frac{3}{2}Al_2O_3 + frac32H_2$$

Katoite can be formed by several mechanisms including

$$Al_2O_3 + 3CaO + 6H2O \longrightarrow Ca_3\,Al_2[(OH)_4]_3$$

and

$$Ca(OH)_2 + Al(s) + 8H_2O \longrightarrow Ca_3\,Al_2((OH)_4)_3 + 6H_2$$

E.G. Avvakumov, E.T. Devyatkina, N.V. Kosova, Mechanochemical Reactions of Hydrated Oxides, Journal of Solid State Chemistry, Volume 113, Issue 2, December 1994, Pages 379-383, ISSN 0022-4596, http://dx.doi.org/10.1006/jssc.1994.1384. (http://www.sciencedirect.com/science/article/pii/S0022459684713843)

Goals

The goal of this project is to design a flameless cooking product that could be a adapted to widespread civilian and military use. In the baseline design, the product should meet the U.S. Military specification for a Flameless Ration Heater, namely to raise the temperature of an 8 oz. (226.8g) entree by 100 degrees F (56.8 C) in twelve minutes, with no visible flame.

1. Select a chemistry. Above all, the product must be safe, effective, inexpensive to manufacture, have a long shelf life, and disposable in an environmentally sound manner. How can you validate your proposed chemistry?
2. Product design. How would you package the product for the marketplace? What is your intended market? How would you package the technology? How do you make a sustainable business?

In [ ]:

A.4 Artificial Gills for Underwater Breathing?

Exercises

What are the oxygen requirements to sustain underwater activity?

For this exercise you will need to find data on that allows you estimate the amount of oxygen necessary to support a meaningful level of human activity underwater. Before jumping to Google, think carefully about what agencies or organizations are likely to have acquired such data, and whether the source of data would be reliable enough for the purposes of developing a commercial device.

- How much oxygen is required?
- What is the composition requirement of breathing case for underwater use?
- What is the oxgyen content of the gas expired by a diver?
- What other requirements must be met? For example, what are the limits on carbon dioxide? How do these requirements change with depth?

Use this data to develop specifications for the required oxygen composition and flowrate.

How much oxygen is dissolved in seawater?

- How can you estimate the amount of oxygen dissolved in water? What chemical data is needed for a first principles estimate?
- Is the amount of dissolved oxygen dependent on temperature? On depth?
- Can you find literature data to compare your calculations to real world measurements?

What are the liquid handling requirements?

- How much seawater has to be processed to produce the required oxygen?
- How would you propose to circulate that much water through your device?
- How does that compare to large organisms that survice in the water, such as sharks, tuna, and groupers? What does that say about the feasibility of this project?

Evalute Commercial Membrane Contacting Device

- 3M sells supported membrances specifically designed for removing dissolved gases from water. Examine the 3M Liqui-Cel product literature. Would these devices work for this application?
- Specifically what model would you select and why?

First Principles Design

Polymeric membrances are generally well suited to gas separation. Sketch a complete flowsheet for the breathing device feature a membrance based oxygen separation device

- What are compositions and flowrates of the gas stream?
- What is the composition of the inlet water stream?
- Would you use a cocurrent or countercurrent contact?
- Using data from the linked article, estimate the surface area required for the separation. How would design the device to provide the required area?
- Are there any fundamental limits on the amount of dissolved oxygen that can be extracted from water?

What about carbon dioxide?

Is this a Feasible Technology?

Putting all of your work together, what is your opinion regarding the feasibility of a portable oxygen breathing device for underwater swimming?

Appendix B. Projects: Process Systems Analysis

B.1 West Virginia Chemical Spill

Summary

This notebook describes a series of simple calculations for the analyis of a chemical spill in West Virgina that occurred in 2014.

Background

On January 9, 2014, approximately 300,000 residents in nine counties in West Virginia, including the city of Charleston and environs, were told to stop using tap water due a chemical spill. A few days later it was estimated that a total of 7,500 gallons of 4-methylcyclohexane methanol (MCHM) leaked into the Elk River from a 40,000 gallon tank on the property of Freedom Industries. The site is about a mile upstream of West Virginia American Water which is the water utility for Charleston and the surrounding area.

MCHM is used in froth floatation process for cleaning coal. Relatively little information is available. It is more often described as an 'irritant' rather than 'toxic' material, but there is little known about its carcinogenic, mutagenic, or development toxicology properties. From various media reports we find the maximum permissible levels of 4-methylcyclohexane methanol in drinking water is 1ppm by weight. The density of MCHM is 0.9074 g/cm^3.

From Wikipedia we find the Elk River has an average flow of 2,650 cu ft/sec, with a minimum recorded flow of 271 cu ft/s in 1972.

Problems

Assume the leak occurred over a 4 hour period and that MCHM is soluble and completely mixed with the river water when it reaches the treatment plant. What is the approximate concentration of MCHM under average river flow conditions?

In [4]:

```
# densities in kg/liter
rhoMCHM = 0.9074
rhoW = 1.0

# Mass flows in kg/sec
mRiver = 2650*28.31*rhoW
mMCHM = 7500*3.785*rhoMCHM/(4*3600)

# concentration in ppm by mass
cMCHM = mMCHM*1e6/(mRiver+mMCHM)
print("MCHM Concentration = {:0.1f} ppm".format(cMCHM))
```

```
MCHM Concentration = 23.8 ppm
```

Under minimum flow conditions?

In [5]:

```
# densities in kg/liter
rhoMCHM = 0.9074
rhoW = 1.0

# Mass flows in kg/sec
mRiver = 271*28.31*rhoW
mMCHM = 7500*3.785*rhoMCHM/(4*3600)

# concentration in ppm by mass
cMCHM = mMCHM*1e6/(mRiver+mMCHM)
print("MCHM Concentration = {:0.1f} ppm".format(cMCHM))
```

```
MCHM Concentration = 233.1 ppm
```

Suppose Freedom Industries installs a tank monitoring system to detect leaks. The objective is to detect any leak that would lead to 0.1ppm or higher in the river water, even under minimum flow conditions. What is minimum leak rate (in liter/hour) that you would have to detect?

In [6]:

```
# densities in kg/liter
rhoMCHM = 0.9074
rhoW = 1.0

# Mass flows in kg/sec
mRiver = 271*28.31*rhoW
mMCHM = 0.1e-6*mRiver

# Volumetric flow in liter/hour
vMCHM = 3600*mMCHM/rhoMCHM
print("MCHM Flowrate = {:0.1f} liters/hour".format(vMCHM))
```

```
MCHM Flowrate = 3.0 liters/hour
```

In [ ]:

```

```

B.2 Ajka Alumina Plant Spill

Summary

This notebook demonstrates the analysis of the 'Red Sludge' tragedy that took place in Ajka, Hungary, in 2010.

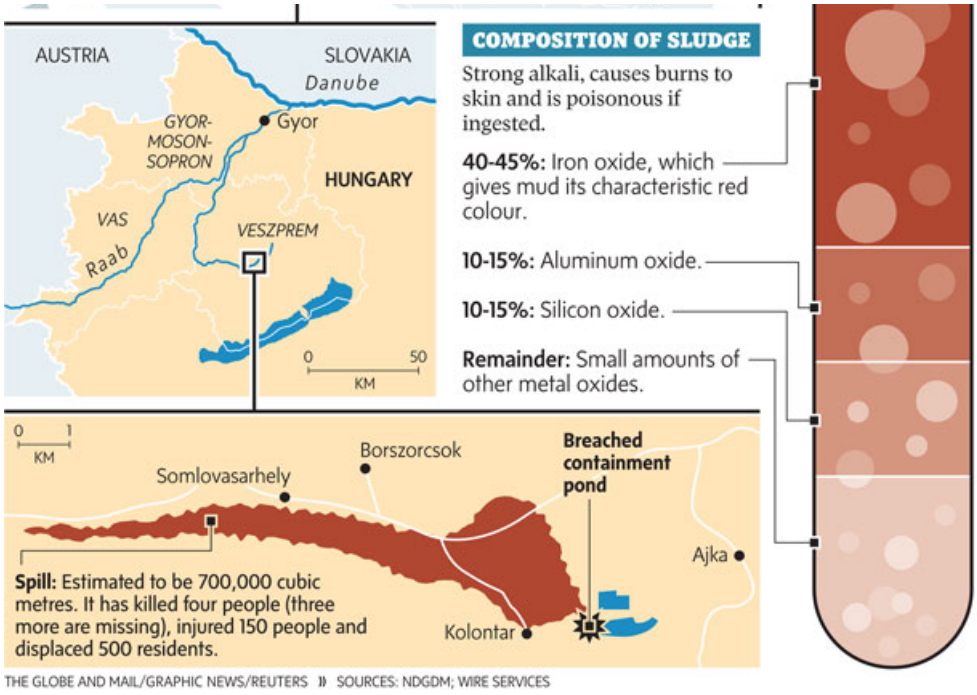
Background

On Monday, October 4th, 2010, an earthen dam failed at an alumina ore processing facility near Ajka, Hungary, resulting in the release of red mud over a large area in western Hungary. According to the New York Times, a flood of 700,000 cubic meters of sludge swept cars off bridges, damaged roads, and flooded businesses and homes throughout the affected region. There were reports of up to eight deaths and hundreds of displaced families. The red mud entered creeks about 45km upstream of the Danube raising the potential for devastating environmental damage. At the time there were fears that other earthen dams at the same processing facility were showing signs of impending failure.

In [1]:

```
from IPython.display import YouTubeVideo
YouTubeVideo("xEMWh6EjJoY",560,315,rel=0)
```

Out[1]:



'Red mud' (or 'red sludge') is the by-product of processing of bauxite ore to produce alumina. Alumina has a wide variety of commercial uses including plastic fillers, catalysts, textile dyes, sunscreens, ceramics, and as the raw material for the production of aluminum metal. The Bayer process is the most commonly used process for the refining of bauxite ore.

In the Bayer process, the aluminum oxide (Al_2O_3) present in the raw bauxite ore is digested in a hot solution of caustic soda (NaOH) at 175 °C. The caustic solution reacts selectively with the aluminum oxide to produce aluminum hydroxide that dissolves in solution. The other components in the bauxite ore do not dissolve and are mechanically separated from the liquid stream. The stream of solid waste and entrained processing liquid forms the 'red mud' that was stored behind the earthen dam at the Hungarian facility.

News reports suggest the red mud produced at this facility was stored with no further processing. This would be inconsistent with modern practice in plants of this type, but if true then the processing liquid entrained in the red mud would be highly caustic and constitute a significant health and environmental hazard.

The purpose of this project is to use your chemical engineering knowledge to gain a deeper understanding of the issues associated with this tragedy. In particular, the news reports provide little hard information about the actual health or environmental hazards posed by the red mud. Some reports refer to the trace metal content, others cite the caustic components with pH ranging from 10 to 13.

We approach this problem starting with the basic chemistry involved in the processing bauxite to produced refined alumina. From this we develop a simple process flowsheet and determine the basic operating constraints on the process. This provides a basis for a worst case estimate of the caustic component of the red mud and assessing possible remediation strategies.

Exercises

Generation/Consumption Analysis

The portion of the Bayer process we're considering consists of two reactions. The digestion reaction results in the dissolution of aluminum oxide Al_2O_3 under caustic conditions

Al_2O_3 (s) + 2 OH^- (aq) + 3 H_2O (l) → 2 $Al(OH)^-_4$ (aq)

Following separation of the undissolved solids, the remaining liquor is cooled under controlled conditions in a crystallizer to precipitate aluminum hydroxide $Al(OH)_3$.

$Al(OH)^-_4$ (aq) → $Al(OH)_3$ (s) + OH^- (aq)

The $Al(OH)_3$ precipitate is subsequently heated in kiln to produce the purified alumina product of the process. We omit those downstream operations from our analysis because they do not contribute to the production of red mud. So considering only the upstream operations, prepare a generation/consumption analysis to determine the overall process stoichiometry for the production of the $Al(OH)_3$ precipitate.

Estimating Caustic Soda

Typical bauxite contains about 40-60 wt% Al_2O_3 with the remainder comprised of iron oxide, silica, and other minerals. Initial news reports on the Hungarian situation provided information from which we can estimate the approximate composition of the processed bauxite ore as shown in this table.

Component	wt %
Al_2O_3	55%
Fe_2O_3	35%
SiO_2	10%

Following digestion and separation, assume the solid components of the red mud have a density of 4.0 g/cc, and that 35 vol% of the red mud is comprised of liquid retained in the void spaces. Based on this information, sketch an input/output diagram for the production of $Al(OH)_3$(s) and estimate the minimum amount of water required to process 1000 kg/hr of bauxite ore.

From this information, can you estimate the amount of caustic soda (NaOH) required to process the bauxite ore?

Free Energy of Reaction

The dissolution process involves reaction with hydroxide ions (OH^-). Hydroxide ions are formed by the dissociation of water

H_2O (l) → H^+ (aq) + OH^- (aq)

The dissociation constant at 25 °C is

$$K_{H_2O} = \frac{a_{H^+} a_{OH^-}}{a_{H_2O}} = 1.2 \times 10^{-14}$$

The molar enthalpy and molar Gibbs free energy of formation of H^+(aq) is normally taken to be zero at STP (1 atm, 25 °C). Using this information and additional data from Table B.3 of your textbook, estimate $\Delta\hat{G}°_{f,OH^-}$ for OH^- (aq).

Estimating Equilibrium Constants

The standard free energies of formation for selected compounds of aluminum at 25 °C. are given in the following table. Data is from Parks, George, American Mineralogist, Vol. 57, pp. 1163-1189 (1972).

Component	$\Delta\hat{G}^\circ_f$ kcal/gmol
$\alpha Al_2O_3(s)$	-378.2
$Al(OH)^-_4(aq)$	-311.0
$\alpha Al(OH)_3$	-275.3

In the Bayer process, alumina is dissolved by digesting Bauxite ore with sodium hydroxide (NaOH) solution at 175 °C. The digestion selectively dissolves the alumina by forming $Al(OH)^-_4(aq)$ according to the reaction

R_1: Al_2O_3 (s) + 2 OH^- (aq) + 3 H_2O (l) → 2 $Al(OH)^-_4(aq)$

(a) Using the results of problem 1 and the other available data, estimate the equilibrium constant K $_{R1</sub>}$ of this reaction at standard pressure and temperature.

(b) Because we don't have enough information to compute $\Delta\hat{H}^\circ_{rxn}$, use the Gibbs free energy as a crude estimate for $\Delta\hat{H}^\circ_{rxn}$. Estimate the equilibrium constant $K_{R1</sub>}$ at 175 °C.

(c) Write an equation for the relationship between equilibrium concentrations of OH^- and $Al(OH)^-_4$ in the digester.

Estimate Equilibrium Yield

Following digestion and separation of the undissolved solids, the liquid from the digester is cooled and seeded to precipitate $Al(OH)_3$ by the reaction

R_2: $Al(OH)_4^-$ (aq) → $Al(OH)_3$ (s) + OH^- (aq)

The $Al(OH)_3$ precipitate is a white, fluffy powder.

(a) Estimate the standard molar free energy of reaction $\Delta\hat{G}^\circ_{R2</sub>}$ and the associated equilibrium constant $K_{R2</sub>}$.

(b) Write an equation for the equilibrium relationship between the liquid phase concentrations of $Al(OH)_4^-$ and OH^- in the crystallizer.

Flowsheet Analysis

Prepare a mass balance of the Bayer process flowsheet shown in the accompanying figure assuming a basis of 1000 kg/hr of bauxite ore.

Assume that the dissolution of Al_2O_3 reaches equilibrium in the digester at an operating temperature of 175 °C, and that the precipitation reaction reaches equilibrium in the crystallizer at an operating temperature of 25 °C. The spent solution from the crystallizer is mixed with makeup NaOH and water before recycling to the digester for reuse. The portion of the liquor from the digester lost through entrainment in the red mud is equal to 35% by volume of the red mud. The density of solids is 4 g/cc and the density of liquids is 1 g/cc. Adjust the flow of caustic soda to maximize the recovery of $Al(OH)_3$(s).

(a) What is the maximum recovery of $Al(OH)_3$(s)?

(b) What is the pH of the red mud when recovery of $Al(OH)_3$(s) is maximized?

How much acid should be shipped to the accident site?

There have been several proposals for remediating the environmental impacts of spilled red mud. At the time of the spill, one village poured vinegar into a river contaminated by the red mud. Suppose 10,000 cubic meters of sludge entered a creek. Assuming vinegar is approximately 5 wt% acetic acid, how much vinegar would be needed to completely neutralize the effects of the red mud? Roughly how many railroad tank cars would be required to transport the vinegar?

Long Term Remediation

There have been several proposals for a long term strategy to remediate the red mud from alumina facilities through CO_2 sequestration. Global alumina production is estimated to be 80,000,000 metric tons per year of Al_2O_3.

Use the results of your calculations above to estimate how much CO_2 would be required to neutralize the red mud produced as a by-product of alumina production through the reaction

$CO_2 + 2\ OH^- \rightarrow H_2O + CO_3^{-2}$

Laboratory measurements show that about 5.3 g of CO_2 can be sequestered per 100 g of red mud. How does this compare to your estimate? Where would the necessary CO_2 come from?

In [ ]: