Chemics Documentation

Release 24.1

Gavin Wiggins

Jan 02, 2024

GETTING STARTED

The Chemics package is a collection of Python functions for performing calculations in the field of chemical engineering. Source code for the package is available on GitHub and contributions from the community are encouraged.

CHAPTER

ONE

INSTALLATION

If you don't have Python installed on your computer, the Anaconda or Miniconda distribution of Python is recommended for scientific computing. After setting up Python, the Chemics package can be downloaded and installed using the pip package manager.

Install Chemics using the pip package manager:

```
pip install chemics
```

CHAPTER

TWO

USAGE

The example below imports the Chemics package and uses the `Gas` class to calculate the density and viscosity of nitrogen gas at a temperature of 773 K and pressure of 101,325 Pa.

```
import chemics as cm

gas = cm.Gas("N2", 773)
rho = gas.density()
mu = gas.viscosity()

print("Nitrogen gas properties at 773 K and 101,325 Pa")
print(f"density    {rho:.4f} kg/m^3")
print(f"viscosity  {mu:.2f} P")
```

This prints the following:

```
Nitrogen gas properties at 773 K and 101,325 Pa
density    0.4416 kg/m^3
viscosity  363.82 P
```

This example uses the `ChemicalEquation` class to get properties of the reactants and products from a given chemical equation.

```
import chemics as cm

ce = cm.ChemicalEquation('2 HCl + 2 Na -> 2 NaCl + H2')
ce.is_balanced()
# This returns True for balanced equation

ce.rct_properties
# This returns a dataframe of the reactant properties
#                HCl        Na
# moles            2         2
# species        HCl        Na
# molwt       36.458     22.99
# mass        72.916     45.98
# molfrac        0.5       0.5
# massfrac  0.613275  0.386725
```

See the **Examples** section for more ways to use Chemics.

CHAPTER

THREE

CONTRIBUTING

See the CONTRIBUTING document on GitHub for guidelines on contributing to the Chemics package. Thank you for contributing!

CHAPTER

FOUR

BIOMASS COMPOSITION

The `biocomp()` function uses ultimate analysis data to estimate biomass composition in terms of cellulose, hemicellulose, lignins, and extractives. The example below uses the carbon `yc` and hydrogen `yh` mass fractions from ultimate analysis data to calculate the cellulose mass fraction on a dry ash-free basis.

```
>>> yc = 0.534
>>> yh = 0.06
>>> bc = cm.biocomp(yc, yh)
>>> bc['y_daf'][0]
0.293...
```

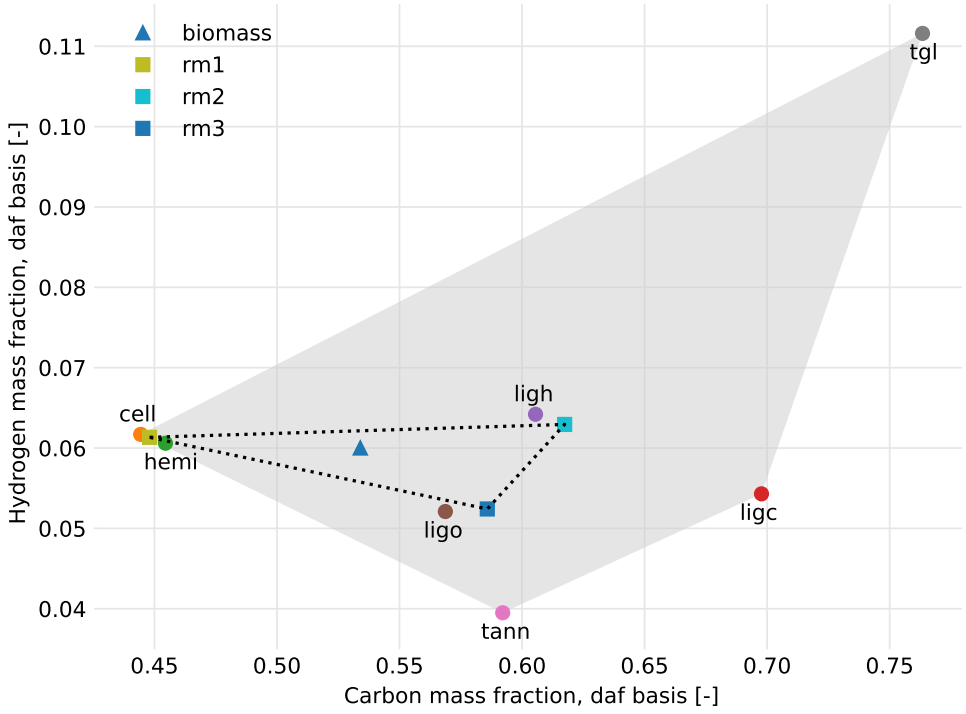
The entire biomass composition can be printed to the screen with `printcomp=True`.

```
>>> bc = cm.biocomp(yc, yh, printcomp=True)
basis    cell    hemi    ligc    ligh    ligo    tann    tgl
x_daf    0.4118  0.2745  0.0627  0.1529  0.0981  0.0000  0.0000
x_wet    0.4118  0.2745  0.0627  0.1529  0.0981  0.0000  0.0000
y_daf    0.2936  0.1595  0.0713  0.2934  0.1822  0.0000  0.0000
y_wet    0.2936  0.1595  0.0713  0.2934  0.1822  0.0000  0.0000
y_wetash 0.2936  0.1595  0.0713  0.2934  0.1822  0.0000  0.0000
```

Use the `plot_biocomp()` function to plot the biomass (triangle symbol) in relation to the reference mixtures (square symbols) as shown below.

```
# Carbon and hydrogen mass fractions from ultimate analysis
yc = 0.534
yh = 0.06

# Calculate the biomass composition
bc = cm.biocomp(yc, yh)

# Plot the biomass and reference mixtures
fig, ax = plt.subplots(tight_layout=True)
cm.plot_biocomp(ax, yc, yh, bc['y_rm1'], bc['y_rm2'], bc['y_rm3'])

plt.show()
```

Notice that the C and H mass fractions may give negative biomass composition values when the default splitting parameters are used as shown below. This is also depicted by the biomass marker going outside the bounds (dashed triangle) of the reference mixtures.



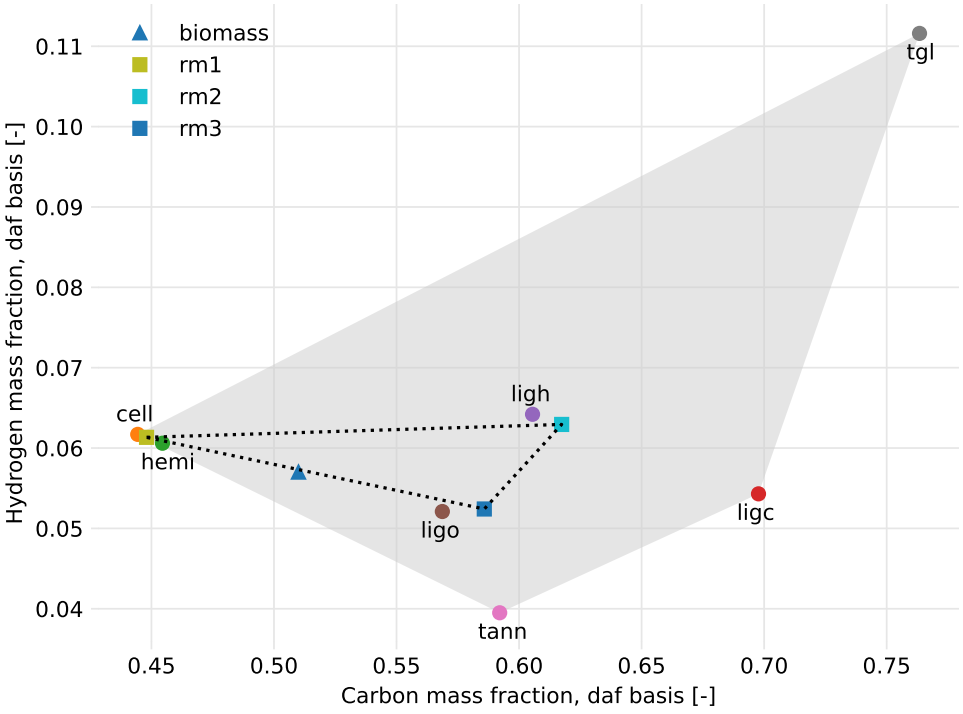

```
>>> yc = 0.51
>>> yh = 0.057
>>> _ = cm.biocomp(yc, yh, printcomp=True)
basis    cell    hemi    ligc    ligh     ligo    tann    tgl
x_daf    0.4534  0.3023  0.0489  -0.0106  0.2061  0.0000  -0.0000
x_wet    0.4534  0.3023  0.0489  -0.0106  0.2061  0.0000  -0.0000
y_daf    0.3527  0.1916  0.0605  -0.0223  0.4175  0.0000  -0.0000
y_wet    0.3527  0.1916  0.0605  -0.0223  0.4175  0.0000  -0.0000
y_wetash 0.3527  0.1916  0.0605  -0.0223  0.4175  0.0000  -0.0000
```

```
yc = 0.51
yh = 0.057
bc = cm.biocomp(yc, yh)

fig, ax = plt.subplots(tight_layout=True)
cm.plot_biocomp(ax, yc, yh, bc['y_rm1'], bc['y_rm2'], bc['y_rm3'])

plt.show()
```

In this particular case, adjust the `epsilon` splitting parameter to properly characterize the biomass for the C and H mass fractions.

```
>>> yc = 0.51
>>> yh = 0.057
>>> _ = cm.biocomp(yc, yh, epsilon=0.4, printcomp=True)
basis    cell    hemi    ligc    ligh    ligo    tann    tgl
x_daf    0.4623  0.3082  0.0259  0.0504  0.0532  0.0998  0.0000
```

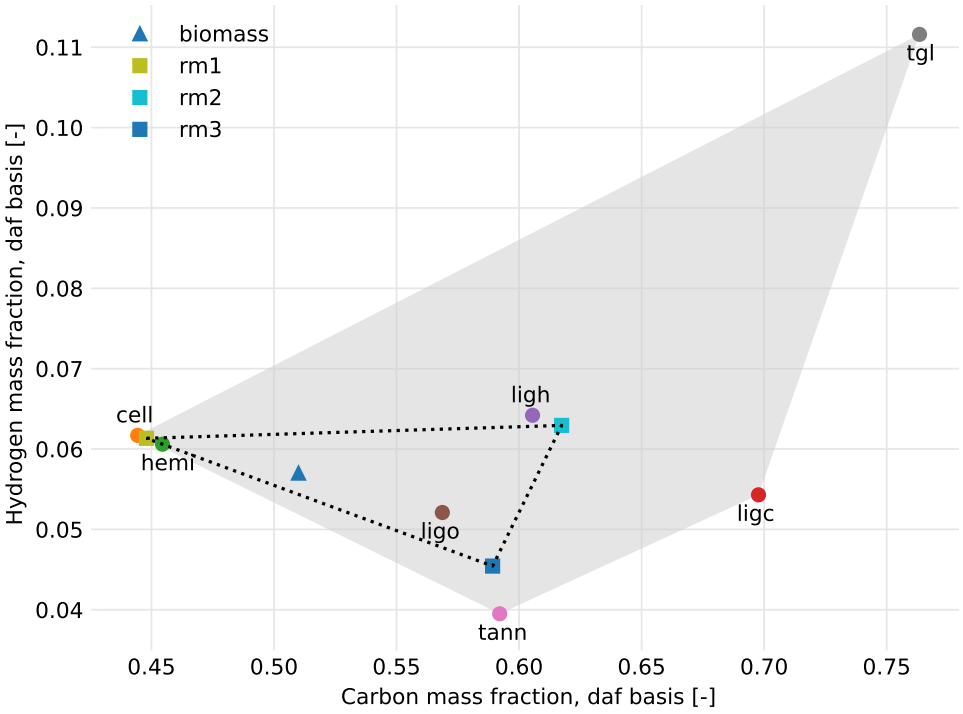
(continues on next page)



(continued from previous page)

```
x_wet    0.4623  0.3082  0.0259  0.0504  0.0532  0.0998  0.0000
y_daf    0.3800  0.2064  0.0339  0.1116  0.1140  0.1540  0.0000
y_wet    0.3800  0.2064  0.0339  0.1116  0.1140  0.1540  0.0000
y_wetash 0.3800  0.2064  0.0339  0.1116  0.1140  0.1540  0.0000
```

```
yc = 0.51
yh = 0.057
bc = cm.biocomp(yc, yh, epsilon=0.4)

fig, ax = plt.subplots(tight_layout=True)
cm.plot_biocomp(ax, yc, yh, bc['y_rm1'], bc['y_rm2'], bc['y_rm3'])

plt.show()
```


biomass
rm1
rm2
rm3
cell
hemi
ligh
ligo
ligc
tgl
tann
0.11
0.10
0.09
0.08
0.07
0.06
0.05
0.04
Hydrogen mass fraction, daf basis [-]
0.45
0.50
0.55
0.60
0.65
0.70
0.75
Carbon mass fraction, daf basis [-]

CHAPTER

FIVE

CHEMICAL EQUATION

The `ChemicalEquation` class provides product and reactant properties from an equation that represents a chemical reaction.

```
>>> eq = cm.ChemicalEquation('2 HCl + 2 Na -> 2 NaCl + H2')

# Check if atomic elements of the reactants and products are balanced
>>> eq.is_balanced()
True

# Total number of atomic elements for each product
>>> eq.prod_elements
{'Na': 2.0, 'Cl': 2.0, 'H': 2.0}

# Total number of moles for the products
>>> eq.prod_moles
3.0

# Total mass of the products
>>> eq.prod_mass
118.896

# Total number of atomic elements for each reactant
>>> eq.rct_elements
{'H': 2.0, 'Cl': 2.0, 'Na': 2.0}

# Total number of moles for the reactants
>>> eq.rct_moles
4.0

# Total mass of the reactants
>>> eq.rct_mass
118.896...

# Properties of the products
>>> eq.prod_properties
              NaCl        H2
moles          2.0       1.0
species       NaCl        H2
molwt        58.44     2.016
mass        116.88     2.016
```

(continues on next page)

(continued from previous page)

```
molfrac   0.666667  0.333333
massfrac  0.983044  0.016956

# Properties of the reactants
>>> eq.rct_properties
               HCl        Na
moles          2.0       2.0
species        HCl        Na
molwt       36.458     22.99
mass        72.916     45.98
molfrac        0.5       0.5
massfrac  0.613275  0.386725
```

Names can be assigned to chemical species using the `names` parameter.

```
>>> names = {'CHAR': 'C', 'CH2OHCHO': 'C2H4O2'}
>>> eq = cm.ChemicalEquation('5 NaCl + 3 H2O -> CH4 + 2.2 CHAR + CH2OHCHO', names)
>>> eq.prod_elements
{'C': 5.2, 'H': 8.0, 'O': 2.0}
```

CHAPTER

SIX

CONVERSIONS

Convert a list of mass fractions `y` to mole fractions where `mw` is the molecular weight in g/mol for each component. In this example, the components represent C, H, O, and N.

```
>>> y = [0.36, 0.16, 0.20, 0.28]
>>> mw = [12.011, 1.008, 15.999, 14.007]
>>> cm.massfrac_to_molefrac(y, mw)
array([0.135..., 0.717..., 0.0565..., 0.0903...])
```

Convert a list of mole fractions `x` to mass fractions.

```
>>> x = [0.36, 0.16, 0.20, 0.28]
>>> mw = [12.011, 1.008, 15.999, 14.007]
>>> cm.molefrac_to_massfrac(x, mw)
array([0.372..., 0.0138..., 0.275..., 0.337...])
```

CHAPTER

SEVEN

DIMENSIONLESS NUMBERS

The examples below demonstrate various dimensionless number functions availabe in chemics.

7.1 Reynolds number

Use the `reynolds()` function to calculate the Reynolds number.

```
u = 2.6     # flow speed in m/s
d = 0.025   # characteristic length or dimension in meters
rho = 910   # density of the fluid or gas in kg/m^3
mu = 0.38   # dynamic viscosity of the fluid or gas in kg/(ms)

re = cm.reynolds(u, d, rho=rho, mu=mu)
print(f'Reynolds number Re is {re:.2f}')
```

```
Reynolds number Re is 155.66
```

The kinematic viscosity can be used as an input parameter instead of the density and dynamic viscosity.

```
u = 0.25      # flow speed in m/s
d = 0.102     # characteristic length or dimension in meters
nu = 1.4e-6   # kinematic viscosity of the fluid or gas in m^2/s

re = cm.reynolds(u, d, nu=nu)
print(f'Reynolds number Re is {re:.2f}')
```

```
Reynolds number Re is 18214.29
```

CHAPTER

EIGHT

GAS PROPERTIES

Gas properties such as molecular weight, density, heat capacity, thermal conductivity, and viscosity can be calculated using the `Gas` class. The example given below calculates nitrogen gas properties at 773 K and 101,325 Pa which is the default pressure.

```
import chemics as cm

gas = cm.Gas('N2', 773)
mw = gas.molecular_weight
rho = gas.density()
cp = gas.heat_capacity()
k = gas.thermal_conductivity()
mu = gas.viscosity()

print('Nitrogen gas properites')
print(f'molecular weight      {mw:.2f} g/mol')
print(f'density               {rho:.3f} kg/m^3')
print(f'heat capacity         {cp:.2f} J/(molK)')
print(f'thermal conductivity  {k:.3f} W/(mK)')
print(f'viscosity             {mu:.2f} microPoise (P)')
```

This prints the output shown below.

```
Nitrogen gas properites
molecular weight      28.01 g/mol
density               0.442 kg/m^3
heat capacity         31.24 J/(molK)
thermal conductivity  0.054 W/(mK)
viscosity             363.82 microPoise (P)
```

CHAPTER

NINE

GAS MIXTURE PROPERTIES

This example calculates the molecular weight and viscosity of a gas mixture using the `GasMixture` class. The gas mixture is initialized with a list of `Gas` objects and their associated mole fractions.

```
import chemics as cm

gas1 = cm.Gas('H2', 773)
gas2 = cm.Gas('N2', 773)
gas3 = cm.Gas('CH4', 825)

gas_mixture = cm.GasMixture([gas1, gas2, gas3], [0.4, 0.1, 0.5])
mw = gas_mixture.molecular_weight()
mu = gas_mixture.viscosity()

print('Gas mixture properties')
print(f'molecular weight  {mw:.2f} g/mol')
print(f'viscosity         {mu:.2f} microPoise (P)')
```

This prints the output shown below.

```
Gas mixture properties
molecular weight  11.63 g/mol
viscosity         226.75 microPoise (P)
```

CHAPTER

TEN

PROXIMATE AND ULTIMATE ANALYSIS

Use the `Proximate` class to express proximate analysis values as different bases. Bases are as-determined (ad), as-received (ar), dry (d), and dry ash-free (daf).

```
>>> prox = cm.Proximate([47.26, 40.05, 4.46, 8.23], 'ad')
>>> print(prox)
                ad        ar        d        daf
FC           47.26     39.53     51.50     54.13
VM           40.05     33.50     43.64     45.87
ash           4.46      3.73      4.86         -
moisture      8.23     23.24         -         -
total       100.00    100.00    100.00    100.00
```

```
>>> prox = cm.Proximate([39.53, 33.50, 3.73, 23.24], 'ar')
>>> print(prox)
                ad        ar        d        daf
FC           47.26     39.53     51.50     54.13
VM           40.05     33.50     43.64     45.87
ash           4.46      3.73      4.86         -
moisture      8.23     23.24         -         -
total       100.00    100.00    100.00    100.00
```

```
>>> prox = cm.Proximate([39.53, 33.50, 3.73, 23.24], 'ar')
>>> prox.daf_basis
array([54.12844037, 45.87155963])
```

Use the `Ultimate` class to express ultimate analysis values as different bases. Bases are as-determined (ad), as-received (ar), dry (d), and dry ash-free (daf).

```
>>> ult = cm.Ultimate([60.08, 5.44, 25.01, 0.88, 0.73, 7.86, 9.00], 'ad')
>>> print(ult)
             ad        ar        d        daf
C         60.08     46.86     66.02     72.26
H          5.44      6.71      4.87      5.33
O         25.01     39.05     18.70     20.47
N          0.88      0.69      0.97      1.06
S          0.73      0.57      0.80      0.88
ash        7.86      6.13      8.64         -
moisture   9.00     29.02         -         -
total    109.00    129.02    100.00    100.00
```

```
>>> ult = cm.Ultimate([46.86, 6.70, 39.05, 0.69, 0.57, 6.13, 29.02], 'ar')
>>> print(ult)
             ad        ar        d        daf
C         60.08     46.86     66.02     72.26
H          5.43      6.70      4.86      5.32
O         25.02     39.05     18.71     20.47
N          0.88      0.69      0.97      1.06
S          0.73      0.57      0.80      0.88
ash        7.86      6.13      8.64         -
moisture   9.00     29.02         -         -
total    109.00    129.02    100.00    100.00
```

```
>>> ult = cm.Ultimate([46.86, 3.46, 13.27, 0.69, 0.57, 6.13, 29.02], 'ar', HO=False)
>>> print(ult)
             ad        ar        d        daf
C         60.08     46.86     66.02     72.26
H          5.44      3.46      4.87      5.34
O         25.01     13.27     18.70     20.46
N          0.88      0.69      0.97      1.06
S          0.73      0.57      0.80      0.88
ash        7.86      6.13      8.64         -
moisture   9.00     29.02         -         -
total    109.00    100.00    100.00    100.00
```

```
>>> ult = cm.Ultimate([46.86, 6.70, 39.05, 0.69, 0.57, 6.13, 29.02], 'ar')
>>> ult.d_basis
array([66.01...,  4.86..., 18.70...,  0.97...,  0.80..., 8.63...])
```

CHAPTER

ELEVEN

PYROLYSIS NUMBER

Functions are available to calculate the Biot and pyrolysis numbers PyI and PyII as shown below.

```
h = 500          # convective heat transfer coefficient in W/(m^2K)
d = 0.00001      # biomass particle diameter in meters
k = 0.12         # biomass thermal conductivity in W/(mK)

kr = 1.4         # reaction rate constant in 1/s
rho = 540        # biomass density in kg/m^3
cp = 3092.8      # biomass heat capacity in J/(kgK)

r = d / 2
biot = cm.biot(h, r, k)
pyI = cm.pyrolysis_one(k, kr, rho, cp, r)
pyII = cm.pyrolysis_two(h, kr, rho, cp, r)

print(f'Biot number is {biot:.4f}')
print(f'PyI number is {pyI:.2f}')
print(f'PyII number is {pyII:.2f}')
```

```
Biot number is 0.0208
PyI number is 2052.90
PyII number is 42.77
```

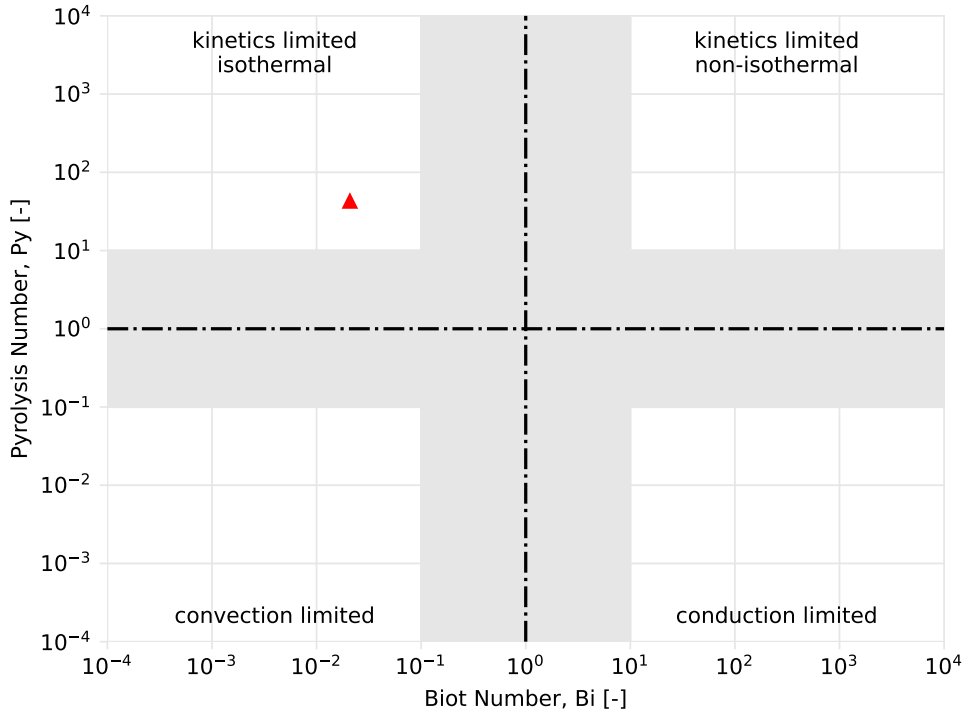
The Biot and pyrolysis numbers can be used to create a regime map for a biomass particle as shown here.

```
h = 500          # convective heat transfer coefficient in W/(m^2K)
d = 0.00001      # biomass particle diameter in meters
k = 0.12         # biomass thermal conductivity in W/(mK)

kr = 1.4         # reaction rate constant in 1/s
rho = 540        # biomass density in kg/m^3
cp = 3092.8      # biomass heat capacity in J/(kgK)

r = d / 2
biot = cm.biot(h, r, k)
pyI = cm.pyrolysis_one(k, kr, rho, cp, r)
pyII = cm.pyrolysis_two(h, kr, rho, cp, r)

if biot < 1.0:
    py = pyII
```

(continues on next page)

(continued from previous page)

```
else:
    py = pyI

fig, ax = plt.subplots(tight_layout=True)
ax.plot(biot, py, 'r^')
ax.set_xlabel('Biot Number, Bi [-]')
ax.set_ylabel('Pyrolysis Number, Py [-]')

ax.text(0.2, 0.91, 'kinetics limited\nisothermal', ha='center', transform=ax.transAxes)
ax.text(0.8, 0.91, 'kinetics limited\nnon-isothermal', ha='center', transform=ax.
↪transAxes)
ax.text(0.2, 0.03, 'convection limited', ha='center', transform=ax.transAxes)
ax.text(0.8, 0.03, 'conduction limited', ha='center', transform=ax.transAxes)

ax.axvline(1, c='k', ls='-.')
ax.axvspan(10**-1, 10**1, color='0.9')
ax.axhline(1, c='k', ls='-.')
ax.axhspan(10**-1, 10**1, color='0.9')
ax.grid(color='0.9')
ax.set_frame_on(False)
ax.set_xlim(10**-4, 10**4)
ax.set_ylim(10**-4, 10**4)
ax.set_xscale('log')
ax.set_yscale('log')
ax.tick_params(color='0.9')
plt.minorticks_off()

plt.show()
```



CHAPTER

TWELVE

WOOD PROPERTIES

The example below uses the `cp_wood()` function to calculate heat capacity of wood with 12% moisture content at a temperature of 340 Kelvin.

```
>>> cp = cm.cp_wood(12, 340)
>>> print(f'cp is {cp:.4f} kJ/(kgK)')
cp is 1.9146 kJ/(kgK)
```

This example estimates thermal conductivity of wood with a basic specific gravity of 0.54, volumetric shrinkage of 12.3%, and 10% moisture content.

```
>>> k = cm.k_wood(0.54, 12.3, 10)
>>> print(f'k is {k:.4f} W/(mK)')
k is 0.1567 W/(mK)
```

CHAPTER

THIRTEEN

ATMOSPHERIC PRESSURE

Calculate atmospheric pressure for a given elevation with the `patm` function. Function for atmospheric pressure.

`chemics.atm_pressure.`**`patm`**(*alt*)

Calculate atmospheric pressure.

Determine atmospheric pressure at altitudes up to 51 km. Based on article by Roland Stull[1].

Parameters
`alt` (*`float`*) – Altitude or elevation above sea level in meters

Returns
Patm (*float*) – Atmospheric pressure in pascal

References

[1] Practical Meteorology: An Algebra-based Survey of Atmospheric Science by Roland Stull.

CHAPTER

FOURTEEN

ATOMIC ELEMENTS

Atomic elements.

Dictionary containing atomic number, name, and atomic weight of elements in the periodic table. Conventional atomic weight is used for atomic weight where applicable. Values taken from IUPAC.

References

IUPAC Periodic Table of the Elements. International Union of Pure and Applied Chemistry, 2016. https://iupac.org/what-we-do/periodic-table-of-elements/.

```
"""

atomic_elements = {
    "H": {"atomic_number": 1, "name": "hydrogen", "atomic_weight": 1.008},
    "He": {"atomic_number": 2, "name": "helium", "atomic_weight": 4.0026},
    "Li": {"atomic_number": 3, "name": "lithium", "atomic_weight": 6.94},
    "Be": {"atomic_number": 4, "name": "beryllium", "atomic_weight": 9.0122},
    "B": {"atomic_number": 5, "name": "boron", "atomic_weight": 10.81},
    "C": {"atomic_number": 6, "name": "carbon", "atomic_weight": 12.011},
    "N": {"atomic_number": 7, "name": "nitrogen", "atomic_weight": 14.007},
    "O": {"atomic_number": 8, "name": "oxygen", "atomic_weight": 15.999},
    "F": {"atomic_number": 9, "name": "fluorine", "atomic_weight": 18.998},
    "Ne": {"atomic_number": 10, "name": "neon", "atomic_weight": 20.180},
    "Na": {"atomic_number": 11, "name": "sodium", "atomic_weight": 22.990},
    "Mg": {"atomic_number": 12, "name": "magnesium", "atomic_weight": 24.305},
    "Al": {"atomic_number": 13, "name": "aluminium", "atomic_weight": 26.982},
    "Si": {"atomic_number": 14, "name": "silicon", "atomic_weight": 28.085},
    "P": {"atomic_number": 15, "name": "phosphorus", "atomic_weight": 30.974},
    "S": {"atomic_number": 16, "name": "sulfur", "atomic_weight": 32.06},
    "Cl": {"atomic_number": 17, "name": "chlorine", "atomic_weight": 35.45},
    "Ar": {"atomic_number": 18, "name": "argon", "atomic_weight": 39.948},
    "K": {"atomic_number": 19, "name": "potassium", "atomic_weight": 39.098},
    "Ca": {"atomic_number": 20, "name": "calcium", "atomic_weight": 40.078},
    "Sc": {"atomic_number": 21, "name": "scandium", "atomic_weight": 44.956},
    "Ti": {"atomic_number": 22, "name": "titanium", "atomic_weight": 47.867},
    "V": {"atomic_number": 23, "name": "vanadium", "atomic_weight": 50.942},
    "Cr": {"atomic_number": 24, "name": "chromium", "atomic_weight": 51.996},
    "Mn": {"atomic_number": 25, "name": "manganese", "atomic_weight": 54.938},
    "Fe": {"atomic_number": 26, "name": "iron", "atomic_weight": 55.845},
```

(continues on next page)

(continued from previous page)


```
    "Co": {"atomic_number": 27, "name": "cobalt", "atomic_weight": 58.933},
    "Ni": {"atomic_number": 28, "name": "nickel", "atomic_weight": 58.693},
    "Cu": {"atomic_number": 29, "name": "copper", "atomic_weight": 63.546},
    "Zn": {"atomic_number": 30, "name": "zinc", "atomic_weight": 65.38},
    "Ga": {"atomic_number": 31, "name": "gallium", "atomic_weight": 69.723},
    "Ge": {"atomic_number": 32, "name": "germanium", "atomic_weight": 72.630},
    "As": {"atomic_number": 33, "name": "arsenic", "atomic_weight": 74.922},
    "Se": {"atomic_number": 34, "name": "selenium", "atomic_weight": 78.971},
    "Br": {"atomic_number": 35, "name": "bromine", "atomic_weight": 79.904},
    "Kr": {"atomic_number": 36, "name": "krypton", "atomic_weight": 83.798},
    "Rb": {"atomic_number": 37, "name": "rubidium", "atomic_weight": 85.468},
    "Sr": {"atomic_number": 38, "name": "strontium", "atomic_weight": 87.62},
    "Y": {"atomic_number": 39, "name": "yttrium", "atomic_weight": 88.906},
    "Zr": {"atomic_number": 40, "name": "zirconium", "atomic_weight": 91.224},
    "Nb": {"atomic_number": 41, "name": "niobium", "atomic_weight": 92.906},
    "Mo": {"atomic_number": 42, "name": "molybdenum", "atomic_weight": 95.95},
    "Tc": {"atomic_number": 43, "name": "technetium", "atomic_weight": None},
    "Ru": {"atomic_number": 44, "name": "ruthenium", "atomic_weight": 101.07},
    "Rh": {"atomic_number": 45, "name": "rhodium", "atomic_weight": 102.91},
    "Pd": {"atomic_number": 46, "name": "palladium", "atomic_weight": 106.42},
    "Ag": {"atomic_number": 47, "name": "silver", "atomic_weight": 107.87},
    "Cd": {"atomic_number": 48, "name": "cadmium", "atomic_weight": 112.41},
    "In": {"atomic_number": 49, "name": "indium", "atomic_weight": 114.82},
    "Sn": {"atomic_number": 50, "name": "tin", "atomic_weight": 118.71},
    "Sb": {"atomic_number": 51, "name": "antimony", "atomic_weight": 121.76},
    "Te": {"atomic_number": 52, "name": "tellurium", "atomic_weight": 127.60},
    "I": {"atomic_number": 53, "name": "iodine", "atomic_weight": 126.90},
    "Xe": {"atomic_number": 54, "name": "xenon", "atomic_weight": 131.29},
    "Cs": {"atomic_number": 55, "name": "caesium", "atomic_weight": 132.91},
    "Ba": {"atomic_number": 56, "name": "barium", "atomic_weight": 137.33},
    "La": {"atomic_number": 57, "name": "lanthanum", "atomic_weight": 138.91},
    "Ce": {"atomic_number": 58, "name": "cerium", "atomic_weight": 140.12},
    "Pr": {"atomic_number": 59, "name": "praseodymium", "atomic_weight": 140.91},
    "Nd": {"atomic_number": 60, "name": "neodymium", "atomic_weight": 144.24},
    "Pm": {"atomic_number": 61, "name": "promethium", "atomic_weight": None},
    "Sm": {"atomic_number": 62, "name": "samarium", "atomic_weight": 150.36},
    "Eu": {"atomic_number": 63, "name": "europium", "atomic_weight": 151.96},
    "Gd": {"atomic_number": 64, "name": "gadolinium", "atomic_weight": 157.25},
    "Tb": {"atomic_number": 65, "name": "terbium", "atomic_weight": 158.93},
    "Dy": {"atomic_number": 66, "name": "dysprosium", "atomic_weight": 162.50},
    "Ho": {"atomic_number": 67, "name": "holmium", "atomic_weight": 164.93},
    "Er": {"atomic_number": 68, "name": "erbium", "atomic_weight": 167.26},
    "Tm": {"atomic_number": 69, "name": "thulium", "atomic_weight": 168.93},
    "Yb": {"atomic_number": 70, "name": "ytterbium", "atomic_weight": 173.05},
    "Lu": {"atomic_number": 71, "name": "lutetium", "atomic_weight": 174.97},
    "Hf": {"atomic_number": 72, "name": "hafnium", "atomic_weight": 178.49},
    "Ta": {"atomic_number": 73, "name": "tantalum", "atomic_weight": 180.95},
    "W": {"atomic_number": 74, "name": "tungsten", "atomic_weight": 183.84},
    "Re": {"atomic_number": 75, "name": "rhenium", "atomic_weight": 186.21},
    "Os": {"atomic_number": 76, "name": "osmium", "atomic_weight": 190.23},
    "Ir": {"atomic_number": 77, "name": "iridium", "atomic_weight": 192.22},
    "Pt": {"atomic_number": 78, "name": "platinum", "atomic_weight": 195.08},
```


(continues on next page)

(continued from previous page)


```
    "Au": {"atomic_number": 79, "name": "gold", "atomic_weight": 196.97},
    "Hg": {"atomic_number": 80, "name": "mercury", "atomic_weight": 200.59},
    "Tl": {"atomic_number": 81, "name": "thallium", "atomic_weight": 204.38},
    "Pb": {"atomic_number": 82, "name": "lead", "atomic_weight": 207.2},
    "Bi": {"atomic_number": 83, "name": "bismuth", "atomic_weight": 208.98},
    "Po": {"atomic_number": 84, "name": "polonium", "atomic_weight": None},
    "At": {"atomic_number": 85, "name": "astatine", "atomic_weight": None},
    "Rn": {"atomic_number": 86, "name": "radon", "atomic_weight": None},
    "Fr": {"atomic_number": 87, "name": "francium", "atomic_weight": None},
    "Ra": {"atomic_number": 88, "name": "radium", "atomic_weight": None},
    "Ac": {"atomic_number": 89, "name": "actinium", "atomic_weight": None},
    "Th": {"atomic_number": 90, "name": "thorium", "atomic_weight": 232.04},
    "Pa": {"atomic_number": 91, "name": "protactinium", "atomic_weight": 231.04},
    "U": {"atomic_number": 92, "name": "uranium", "atomic_weight": 238.03},
    "Np": {"atomic_number": 93, "name": "neptunium", "atomic_weight": None},
    "Pu": {"atomic_number": 94, "name": "plutonium", "atomic_weight": None},
    "Am": {"atomic_number": 95, "name": "americium", "atomic_weight": None},
    "Cm": {"atomic_number": 96, "name": "curium", "atomic_weight": None},
    "Bk": {"atomic_number": 97, "name": "berkelium", "atomic_weight": None},
    "Cf": {"atomic_number": 98, "name": "californium", "atomic_weight": None},
    "Es": {"atomic_number": 99, "name": "einsteinium", "atomic_weight": None},
    "Fm": {"atomic_number": 100, "name": "fermium", "atomic_weight": None},
    "Md": {"atomic_number": 101, "name": "mendelevium", "atomic_weight": None},
    "No": {"atomic_number": 102, "name": "nobelium", "atomic_weight": None},
    "Lr": {"atomic_number": 103, "name": "lawrencium", "atomic_weight": None},
    "Rf": {"atomic_number": 104, "name": "rutherfordium", "atomic_weight": None},
    "Db": {"atomic_number": 105, "name": "dubnium", "atomic_weight": None},
    "Sg": {"atomic_number": 106, "name": "seaborgium", "atomic_weight": None},
    "Bh": {"atomic_number": 107, "name": "bohrium", "atomic_weight": None},
    "Hs": {"atomic_number": 108, "name": "hassium", "atomic_weight": None},
    "Mt": {"atomic_number": 109, "name": "meitnerium", "atomic_weight": None},
    "Ds": {"atomic_number": 110, "name": "darmstadtium", "atomic_weight": None},
    "Rg": {"atomic_number": 111, "name": "roentgenium", "atomic_weight": None},
    "Cn": {"atomic_number": 112, "name": "copernicium", "atomic_weight": None},
    "Nh": {"atomic_number": 113, "name": "nihonium", "atomic_weight": None},
    "Fl": {"atomic_number": 114, "name": "flerovium", "atomic_weight": None},
    "Mc": {"atomic_number": 115, "name": "moscovium", "atomic_weight": None},
    "Lv": {"atomic_number": 116, "name": "livermorium", "atomic_weight": None},
    "Ts": {"atomic_number": 117, "name": "tennessine", "atomic_weight": None},
    "Og": {"atomic_number": 118, "name": "oganesson", "atomic_weight": None},
}
```

CHAPTER

FIFTEEN

BIOMASS COMPOSITION

Biomass composition can be represented in terms of cellulose, hemicellulose, lignins, and extractives. If experimental data is not available, the components of biomass can be estimated from the ultimate analysis using a characterization method developed by Debiagi, Pecchi, Gentile, Frassoldati, Cuoci, Faravelli, and Ranzi.

Use the *biocomp()* function to calculate biomass composition. Use the *plot_biocomp()* function to create a Matplotlib figure of the biomass composition results.

chemics.**biocomp**(*yc*, *yh*, *yo=None*, *yh2o=0*, *yash=0*, *alpha=0.6*, *beta=0.8*, *gamma=0.8*, *delta=1*, *epsilon=1*, *printcomp=False*)

Biomass composition.

Determine bimoass composition from ultimate analysis mass fractions of C, H, and O. Composition returned as cellulose, hemicellulose, lignins, and extractives based on method discussed in the Debiagi 2015 paper[1].

Parameters

- **yc** (*float*) – Mass fraction of carbon in biomass, dry ash free basis
- **yh** (*float*) – Mass fraction of hydrogen in biomass, dry ash free basis
- **yo** (*float, optional*) – Mass fraction of oxygen in biomass, if not given then value is calculated as difference, dry ash free basis. Default is None.
- **yh2o** (*float, optional*) – Mass fraction of water in biomass, as received basis. Default is 0.
- **yash** (*float, optional*) – Mass fraction of ash in biomass, as received basis. Default is 0.
- **alpha** (*float, optional*) – Splitting parameter as molar ratio of cellulose and hemicellulose contained in reference mixture RM1. Default is 0.6.
- **beta** (*float, optional*) – Splitting parameter as molar ratio of lignin LIG-O and lignin LIG-C contained in reference mixture RM2. Default is 0.8.
- **gamma** (*float, optional*) – Splitting parameter as molar ratio of lignin LIG-H and lignin LIG-C contained in reference mixture RM3. Default is 0.8.
- **delta** (*float, optional*) – Splitting parameter as molar ratio of lignins (LIG-H and LIG-C) and extractive TGL to define reference mixture RM2. Default is 1.0.
- **epsilon** (*float, optional*) – Splitting parameter as molar ratio of lignins (LIG-O and LIG-C) and extractive TANN to define reference mixture RM3. Default is 1.0.
- **printcomp** (*bool, optional*) – Print composition results if True. Default is False.

[1] Paulo Eduardo Amaral Debiagi, Chiara Pecchi, Giancarlo Gentile, Alessio Frassoldati, Alberto Cuoci, Tiziano Faravelli, and Eliseo Ranzi. Extractives Extend the Applicability of Multistep Kinetic Scheme of Biomass Pyrolysis. Energy and Fuels, vol. 29, no. 10, pp. 6544-6555, 2015.

Returns

comp (*dict*) – Dictionary representing reference mixtures and biomass compositions on the basis of mole fractions (x) and mass fractions (y). The dictionary contains the following items for the reference mixtures where each item represents the C, H, O mass fraction.

- *y_rm1* reference mixture RM1
- *y_rm2* reference mixture RM2
- *y_rm3* reference mixture RM3

The dictionary also contains the following items for the biomass composition where each item represents the CELL, HEMI, LIGC, LIGH, LIGO, TANN, TGL mole or mass fraction.

- *x_daf* mole fractions as dry ash-free basis
- *x_wet* mole fractions as wet basis
- *y_daf* mass fractions as dry ash-free basis
- *y_wet* mass fractions as wet basis
- *y_wetash* mass fractions as wet + ash basis

Raises

ValueError – When sum of mass fractions is not equal to one.

Examples

Use the carbon and hydrogen mass fractions of the biomass to calculate the biomass composition.

```
>>> yc = 0.534
>>> yh = 0.06
>>> bc = cm.biocomp(yc, yh)
```

The cellulose mass fraction on a dry ash-free basis.

```
>>> cell = bc['y_daf'][0]
>>> cell
0.293...
```

The hemicellulose mass fraction on a dry ash-free basis.

```
>>> hemi = bc['y_daf'][1]
>>> hemi
0.159...
```

References

chemics.**plot_biocomp**(*ax*, *yc*, *yh*, *y_rm1*, *y_rm2*, *y_rm3*)

Plot biomass composition.

Plot characterization of biomass sample and calculated reference mixtures as mass fractions, dry ash-free basis.

Parameters

- **ax** (*Axes*) – The Matplotlib axes from a figure
- **yc** (*float*) – Mass fraction of carbon in biomass, dry ash free basis

- **yh** (*float*) – Mass fraction of hydrogen in biomass, dry ash free basis
- **y_rm1** (*array*) – Mass fraction of reference mixture RM1 where y_rm1 = [yC, yH, yO]
- **y_rm2** (*array*) – Mass fraction of reference mixture RM2 where y_rm2 = [yC, yH, yO]
- **y_rm3** (*array*) – Mass fraction of reference mixture RM3 where y_rm3 = [yC, yH, yO]

Returns

ax (*Axes*) – The Matplotlib axes for the plot figure

CHAPTER

SIXTEEN

CHEMICAL EQUATION

Use the ChemicalEquation class to determine properties of the reactants and products in a given chemical equation.

class chemics.**ChemicalEquation**(*eq*, *names=None*)

Properties of the reactants and products in a given chemical equation.

Parameters

- **eq** (*str*) – Chemical equation such as A + B -> C + D
- **names** (*dict, optional*) – Names of unique species and their corresponding chemical formula.

Variables

- **eq** (*str*) – Chemical equation such as A + B -> C + D
- **names** (*dict, optional*) – Names of unique species and their corresponding chemical formula.
- **rct_properties** (*dataframe*) – Number of moles, chemical species, molecular weight, mass, mole fraction, and mass fraction for each reactant.
- **rct_elements** (*dict*) – Total number of atomic elements on reactant side of the equation.
- **rct_moles** (*float*) – Total number of moles on reactant side of the equation.
- **rct_mass** (*float*) – Total mass of the reactants.
- **prod_properties** (*dataframe*) – Number of moles, chemical species, molecular weight, mass, mole fraction, and mass fraction for each product.
- **prod_elements** (*dict*) – Total number of atomic elements on product side of the equation.
- **prod_moles** (*float*) – Total number of moles on product side of the equation.
- **prod_mass** (*float*) – Total mass of the products.

Examples

```
>>> ce = cm.ChemicalEquation('2 HCl + 2 Na -> 2 NaCl + H2')
>>> ce.is_balanced()
True
```

```
>>> ce = cm.ChemicalEquation('2 HCl + 2 Na -> 2 NaCl + H2')
>>> ce.rct_properties
               HCl        Na
```

(continues on next page)

(continued from previous page)

```
moles          2.0       2.0
species        HCl        Na
molwt       36.458     22.99
mass        72.916     45.98
molfrac        0.5       0.5
massfrac  0.613275  0.386725
```

```
>>> ce = cm.ChemicalEquation('2 HCl + 2 Na -> 2 NaCl + H2')
>>> ce.rct_properties.loc['massfrac']
HCl    0.613275
Na     0.386725
...
```

```
>>> ce = cm.ChemicalEquation('2 HCl + 2 Na -> 2 NaCl + H2')
>>> ce.rct_elements
{'H': 2.0, 'Cl': 2.0, 'Na': 2.0}
```

```
>>> ce = cm.ChemicalEquation('2 HCl + 2 Na -> 2 NaCl + H2')
>>> ce.rct_moles
4.0...
```

```
>>> ce = cm.ChemicalEquation('2 HCl + 2 Na -> 2 NaCl + H2')
>>> ce.rct_mass
118.896...
```

is_balanced() → bool

Check balance of atomic elements in reactants or products.

CHAPTER

SEVENTEEN

CONVERSIONS

Helper functions for unit conversions.

chemics.**massfrac_to_molefrac**(*y*, *mw*)

Convert mass fraction to mole fraction. Assumes a total mass of 100 grams.

$$m_i = y_i \times 100$$

$$x_i = \frac{\frac{m_i}{MW_i}}{\sum \frac{m_i}{MW_i}}$$

where m is mass [g], y is mass fraction [-], x is mole fraction [-], and MW is molecular weight [g/mol] of each component.

Parameters

- **y** (*list, tuple or array*) – Mass fraction of each component
- **mw** (*list, tuple or array*) – Molecular weight of each component in g/mol

Returns

x (*array*) – Mole fractions of each component

Example

```
>>> y = [0.36, 0.16, 0.20, 0.28]
>>> mw = [12.011, 1.008, 15.999, 14.007]
>>> cm.massfrac_to_molefrac(y, mw)
array([0.135..., 0.717..., 0.056..., 0.090...])
```

chemics.**molefrac_to_massfrac**(*x*, *mw*)

Convert mole fraction to mass fraction. Assumes total moles is 100.

$$n_i = x_i \times 100$$

$$y_i = \frac{n_i\, MW_i}{\sum n_i\, MW_i}$$

where n is moles [mol], x is mole fraction [-], y is mass fraction [-], and MW is molecular weight [g/mol] of each component.

Parameters

- **x** (*list, tuple, or array*) – Mole fraction of each component [-]
- **mw** (*list, tuple or array*) – Molecular weight of each component [g/mol]

Returns

y (*array*) – Mass fraction of each component [-]

Example

```
>>> x = [0.36, 0.16, 0.20, 0.28]
>>> mw = [12.011, 1.008, 15.999, 14.007]
>>> cm.molefrac_to_massfrac(x, mw)
array([0.372..., 0.0138..., 0.275..., 0.337...])
```

chemics.**slm_to_lpm**(*slm*, *pgas*, *tgas*)

Convert volumetric gas flow.

Convert volumetric gas flow from standard liters per minute (SLM or SLPM) to liters per minute (LPM) where STP is defined as 273.25 K and 101,325 Pa.

$$1\ LPM = 1\ SLPM \times \frac{T_{gas}}{273.15\,K} \times \frac{14.696\,psi}{P_{gas}}$$

Parameters

- **slm** (*float*) – Volumetric gas flow in standard liters per minute [SLM]
- **pgas** (*float*) – Absolute gas pressure [kPa]
- **tgas** (*float*) – Gas temperature [K]

Returns

lpm (*float*) – Volumetric gas flow in liters per minute [LPM]

Example

```
>>> cm.slm_to_lpm(580, 150, 773)
1108.74...
```

References

Wikipedia contributors. (2018, February 8). Standard litre per minute. In Wikipedia online. Retrieved from https://en.wikipedia.org/wiki/Standard_litre_per_minute

CHAPTER

EIGHTEEN

DIMENSIONLESS NUMBERS

Functions are provided for the following dimensionless numbers:

- Archimedes number (Ar)
- Biot number (Bi)
- Peclet number (Pe)
- Prandtl number (Pr)
- Pyrolysis number I (Py I)
- Pyrolysis number II (Py II)
- Reynolds number (Re)
- Schmidt number (Sc)
- Sherwood number (Sh)
- Flow regime

18.1 Source code

chemics.**archimedes**(*dp*, *rhog*, *rhos*, *mu*)

Calculate the dimensionless Archimedes number.

$$Ar = \frac{d_p^3 \rho_g (\rho_s - \rho_g) g}{\mu^2}$$

Parameters

- **dp** (*float*) – Particle diameter in meters
- **rhog** (*float*) – Gas density in kg/m^3
- **rhos** (*float*) – Solid density in kg/m^3
- **mu** (*float*) – Dynamic viscosity in kg/(ms)

Returns

ar (*float*) – Archimedes number

Example

```
>>> cm.archimedes(0.001, 910, 2500, 0.001307)
8309.1452...
```

References

Daizo Kunii and Octave Levenspiel. Fluidization Engineering. Butterworth-Heinemann, 2nd edition, 1991.

chemics.**biot**(*h*, *d*, *k*)

Calculate the dimensionless Biot number.

$$Bi = \frac{h\,d}{k}$$

Parameters

- **h** (*float*) – Convective heat transfer coefficient in W/(m^{2}K)
- **d** (*float*) – Characteristic length or dimension in meters
- **k** (*float*) – Thermal conductivity in W/(mK)

Returns

bi (*float*) – Biot number

Example

```
>>> cm.biot(4.63, 0.001, 3.84)
0.0012057...
```

References

Daizo Kunii and Octave Levenspiel. Fluidization Engineering. Butterworth-Heinemann, 2nd edition, 1991.

chemics.**peclet**(*ui*, *L*, *Dax*)

Calculate the dimensionless Peclet number for mass transfer.

The Peclet number is defined as the ratio between the bulk mass transport (convection) and the molecular diffusion.

$$Pe = \frac{u_i L}{D_{ax}}$$

Parameters

- **ui** (*float*) – Interstitial velocity in m/s
- **L** (*float*) – Length or characteristic dimension in meters
- **Dax** (*float*) – Axial dispersion coefficient in m^2/s

Returns

pe (*float*) – Peclet number

Example

```
>>> cm.peclet(3.0e-3, 0.25, 4.7e-4)
1.5957...
```

References

J.D. Seader, E.J. Henley, D.K. Roper. Separation Process Principles. John Wiley & Sons, Inc., 3rd edition, 2011.

chemics.**prandtl**(*cp=None*, *mu=None*, *k=None*, *nu=None*, *alpha=None*)

Calculate the dimensionless Prandtl number for a fluid or gas.

$$Pr = \frac{c_p \mu}{k} = \frac{\nu}{\alpha}$$

Parameters

- **cp** (*float*) – Specific heat in J/(kgK)
- **mu** (*float*) – Dynamic viscosity in kg/(ms)
- **k** (*float*) – Thermal conductivity in W/(mK)
- **nu** (*float, optional*) – Kinematic viscosity in m^2/s
- **alpha** (*float, optional*) – Thermal diffusivity in m^2/s

Returns

pr (*float*) – Prandtl number

Examples

```
>>> cm.prandtl(cp=4188, mu=0.001307, k=0.5674)
9.647...
```

```
>>> cm.prandtl(nu=1.5064e-5, alpha=2.1002e-5)
0.71726...
```

Raises

ValueError – Must provide (cp, mu, k) or (nu, alpha)

References

Daizo Kunii and Octave Levenspiel. Fluidization Engineering. Butterworth-Heinemann, 2nd edition, 1991.

chemics.**pyrolysis_one**(*k*, *kr*, *rho*, *cp*, *r*)

Calculate the pyrolysis number Py I for a biomass particle.

$$Py^I = \frac{k}{\rho\, C_p\, R^2\, K}$$

Parameters

- **k** (*float*) – Thermal conductivity of the biomass particle in W/(mK)
- **kr** (*float*) – Rate constant in 1/s

- **rho** (*float*) – Density of the biomass particle in kg/m^3
- **cp** (*float*) – Heat capacity of the biomass particle in J/(kgK)
- **r** (*float*) – Radius or characteristic length of the biomass particle in meters

Returns

pyro_one (*float*) – Pyrolysis number Py I

Example

```
>>> cm.pyrolysis_one(k=0.12, kr=1.38556, rho=540, cp=3092.871049, r=0.0001847)
1.52...
```

References

D.L. Pyle and C.A. Zaror. Heat Transfer and Kinetics in the Low Temperature Pyrolysis of Solids. Chemical Engineering Science, vol. 39, no. 1, pg. 147-158, 1984.

chemics.**pyrolysis_two**(*h*, *kr*, *rho*, *cp*, *r*)

Calculate the pyrolysis number Py II for a biomass particle.

$$Py^{II} = \frac{h}{\rho\, C_p\, R\, K}$$

Parameters

- **h** (*float*) – Convective heat transfer coefficient in W/m^{2}K
- **kr** (*float*) – Rate constant in 1/s
- **rho** (*float*) – Density of the biomass particle in kg/m^3
- **cp** (*float*) – Heat capacity of the biomass particle in J/(kgK)
- **r** (*float*) – Radius or characteristic length of the biomass particle in meters

Returns

pyro_two (*float*) – Pyrolysis number Py II

Example

```
>>> cm.pyrolysis_two(h=862.6129, kr=1.38556, rho=540, cp=3092.871049, r=0.0001847)
2.018...
```

References

D.L. Pyle and C.A. Zaror. Heat Transfer and Kinetics in the Low Temperature Pyrolysis of Solids. Chemical Engineering Science, vol. 39, no. 1, pg. 147-158, 1984.

chemics.**reynolds**(*u*, *d*, *rho=None*, *mu=None*, *nu=None*)

Calculate the dimensionless Reynolds number for a fluid or gas flow.

$$Re = \frac{\rho\, u\, d}{\mu} = \frac{u\, d}{\nu}$$

Parameters

- **u** (*float*) – Flow speed in m/s
- **d** (*float*) – Characteristic length or dimension in meters
- **rho** (*float, optional*) – Density of the fluid or gas in kg/m^3
- **mu** (*float, optional*) – Dynamic viscosity of the fluid or gas in kg/(ms)
- **nu** (*float, optional*) – Kinematic viscosity of the fluid or gas in m^2/s

Returns

re (*float*) – Reynolds number

Examples

```
>>> cm.reynolds(2.6, 0.025, rho=910, mu=0.38)
155.65789...
```

```
>>> cm.reynolds(0.25, 0.102, nu=1.4e-6)
18214.2857...
```

Raises

ValueError – Must provide (u, d, rho, mu) or (u, d, nu)

References

Daizo Kunii and Octave Levenspiel. Fluidization Engineering. Butterworth-Heinemann, 2nd edition, 1991.

chemics.**schmidt**(*mu*, *rho*, *Dm*)

Calculate the dimensionless Schmidt number.

The Schmidt number represents the ratio between momentum diffusivity (kinematic viscosity) and mass diffusivity.

$$Sc = \frac{\mu}{\rho D_m}$$

Parameters

- **mu** (*float*) – Viscosity of the fluid flowing through the packed bed in Pas
- **rho** (*float*) – Density of the fluid flowing through the packed bed in kg/m^3
- **Dm** (*float*) – Molecular diffusion coefficient in m^2/s

Returns

sc (*float*) – Schmidt number

Example

```
>>> cm.schmidt(8.90e-4, 997.07, 2.299e-9)
388.26...
```

References

J.D. Seader, E.J. Henley, D.K. Roper. Separation Process Principles. John Wiley & Sons, Inc., 3rd edition, 2011.

chemics.**sherwood**(*k*, *d*, *Dm*)

Calculate the dimensionless Sherwood number.

The Sherwood number represents the ratio between the convective mass transfer and the rate of diffusive mass transport.

$$Sh = \frac{kd}{D_m}$$

Parameters

- **k** (*float*) – Convective mass transfer coefficient in m/s
- **d** (*float*) – Particle diameter or characteristic length in meters
- **Dm** (*float*) – Molecular diffusion coefficient in m^2/s

Returns

sh (*float*) – Sherwood number

Example

```
>>> cm.sherwood(2.3e-4, 5.0e-6, 4.0e-9)
0.2875...
```

References

J.D. Seader, E.J. Henley, D.K. Roper. Separation Process Principles. John Wiley & Sons, Inc., 3rd edition, 2011.

chemics.**flow_regime**(*Re=None*, *u=None*, *d=None*, *rho=None*, *mu=None*, *nu=None*)

Flow regime.

Determine flow regime (laminar, transitional or turbulent) considering the Reynolds number boundaries for the case of a straight, non-smooth pipe.

Laminar regime …….. Re < 2100
Transitional regime … 2100 <= Re <= 4000
Laminar regime …….. Re > 4000

Parameters

- **Re** (*float, optional*) – Reynolds number
- **u** (*float, optional*) – Flow speed in m/s

- **d** (*float, optional*) – Characteristic length or dimension in meters
- **rho** (*float, optional*) – Density of the fluid or gas in kg/m^3
- **mu** (*float, optional*) – Dynamic viscosity of the fluid or gas in kg/(ms)
- **nu** (*float, optional*) – Kinematic viscosity of the fluid or gas in m^2/s

Returns

regime (*string*) – Flow regime. One of laminar, transitional or turbulent.

Examples

```
>>> cm.flow_regime(u=2.6, d=0.025, rho=910, mu=0.38)
'laminar'
```

```
>>> cm.flow_regime(Re=3250)
'transitional'
```

```
>>> cm.flow_regime(u=0.25, d=0.102, nu=1.4e-6)
'turbulent'
```

Raises

ValueError – Must provide Re or (u, d, rho, mu) or (u, d, nu)

References

R.H. Perry, D.W. Green. Perry's Chemical Engineers' Handbook. McGraw-Hill, 8th edition, 2008.

CHAPTER

NINETEEN

GAS

Use the `Gas` class to calculate gas phase properties.

class chemics.**Gas**(*formula*, *temperature*, *pressure=101325*, *cas_number=None*)

Gas properties class.

Parameters

- **formula** (*str*) – Molecular formula of the gas.
- **temperature** (*float*) – Temperature of the gas in kelvin (K).
- **pressure** (*float, optional*) – Pressure of the gas in pascal (Pa). Default value is 101,325 Pa for standard atmosphere.
- **cas_number** (*str, optional*) – CAS (Chemical Abstracts Service) number of the gas, may be required for some species.

Variables

- **formula** (*str*) – Molecular formula of the gas.
- **temperature** (*float*) – Temperature of the gas in kelvin (K).
- **pressure** (*float*) – Pressure of the gas in pascal (Pa).
- **cas_number** (*str, optional*) – CAS (Chemical Abstracts Service) number of the gas, may be required for some species.
- **molecular_weight** (*float*) – Molecular weight of the gas in g/mol.

density()

Gas density.

Calculate gas density using the molecular weight, pressure, and temperature of the gas.

Returns

rho (*float*) – Density of the gas in kg/m^3.

Examples

```
>>> gas = cm.Gas('N2', 773)
>>> gas.density()
0.4416...
```

heat_capacity()

Gas heat capacity.

Calculate gas heat capacity as a function of temperature using Yaws' coefficients[1]. The CAS (Chemical Abstracts Service) number may be required for some species.

$$C_p = A + B\,T + C\,T^2 + D\,T^3 + E\,T^4 + F\,T^5 + G\,T^6$$

Raises

- **ValueError** – If provided CAS number is not found.
- **ValueError** – If multiple substances found for given formula.
- **ValueError** – If gas chemical formula not found.
- **ValueError** – If given temperataure is out of range for calculation.

Returns

cp (*float*) – Heat capacity of the gas in J/(molK).

Examples

```
>>> gas = cm.Gas('CBrClF2', 700)
>>> gas.heat_capacity()
97.4982...
```

```
>>> gas = cm.Gas('C5H10O2', 850, cas_number='75-98-9')
>>> gas.heat_capacity()
268.4920...
```

```
>>> gas = cm.Gas('NO2', 900)
>>> gas.heat_capacity()
51.0686...
```

References

thermal_conductivity()

Gas thermal conductivity.

Calculate gas thermal conductivity as a function of temperature using Yaws' coefficients[2]. The CAS (Chemical Abstracts Service) number may be required for some species.

Raises

[1] Carl L. Yaws. Heat capacity of gas Tables 39 and 40 in Yaws' Critical Property Data for Chemical Engineers and Chemists. Published by Knovel, 2014.

[2] Carl L. Yaws. Thermal Conductivity Gas Tables 84 and 85 in Yaws' Critical Property Data for Chemical Engineers and Chemists. Published by Knovel, 2014.

- **ValueError** – If provided CAS number is not found.
- **ValueError** – If multiple substances found for given formula.
- **ValueError** – If gas chemical formula not found.
- **ValueError** – If given temperataure is out of range for calculation.

Returns

k (*float*) – Thermal conductivity of the gas in W/(mK).

Examples

```
>>> gas = cm.Gas('N2', 773)
>>> gas.thermal_conductivity()
0.0535...
```

```
>>> gas = cm.Gas('C18H38O', 920, cas_number='593-32-8')
>>> gas.thermal_conductivity()
0.0417...
```

References

viscosity(*method='yaws'*)

Gas viscosity.

Calculate gas viscosity as a function of temperature using Ludwig's coefficients[3] or Yaws' coefficients[4]. The CAS (Chemical Abstracts Service) number may be required for some species.

The Ludwig coefficients are used with the following correlation

$$\mu = A + B\,T + C\,T^2$$

The Yaws coefficients are used with the following correlation

$$\mu = A + B\,T + C\,T^2 + D\,T^3$$

Parameters

method (*str, optional*) – Method for determining coefficients, choose yaws or ludwig. Default method is yaws.

Raises

- **ValueError** – If provided CAS number is not found.
- **ValueError** – If multiple substances found for given formula.
- **ValueError** – If gas chemical formula not found.
- **ValueError** – If given temperataure is out of range for calculation.

Returns

mu (*float*) – Gas viscosity in microPoise (P).

[3] A. Kayode Coker. Table C-2 Viscosity of Gas in Ludwig's Applied Process Design for Chemical and Petrochemical Plants, Volume 2, 4th Edition. Elsevier, 2010.

[4] Carl L. Yaws. Viscosity Gas Tables 80 and 81 in Yaws' Critical Property Data for Chemical Engineers and Chemists. Published by Knovel, 2014.

Examples

```
>>> gas = cm.Gas('CH4', 810)
>>> gas.viscosity(method='yaws')
234.21...
```

```
>>> gas = cm.Gas('C2Cl2F4', 900, cas_number='374-07-2')
>>> gas.viscosity()
314.90...
```

```
>>> gas = cm.Gas('H2', 404)
>>> gas.viscosity()
113.18...
```

```
>>> gas = cm.Gas('NH3', 850)
>>> gas.viscosity(method='ludwig')
300.84...
```

```
>>> gas = cm.Gas('C2H4O', 920, cas_number='75-07-0')
>>> gas.viscosity(method='ludwig')
242.46...
```

References

CHAPTER

TWENTY

GAS MIXTURE

Use the `GasMixture` class to calculate properties of a gas mixture.

class chemics.**GasMixture**(*gases*, *mole_fractions*)

Gas mixture class.

Parameters

- **gases** (*list of Gas objects*) – Gas objects representing each component of the gas mixture.
- **mole_fractions** (*list*) – Mole fraction of each gas component.

Variables

- **molecular_weights** (*list of float*) – Molecular weight of each gas component in g/mol.
- **viscosities** (*list of float*) – Viscosity of each gas component in microPoise (P).
- **mole_fractions** (*list of float*) – Mole fraction of each gas component.

Raises

ValueError – If sum of mole fractions does not equal 1.

molecular_weight()

Calculate molecular weight of the gas mixture as a weighted mean.

Returns

mw_mixture (*float*) – Molecular weight of the gas mixture in g/mol.

Examples

```
>>> gas1 = cm.Gas('H2', 773)
>>> gas2 = cm.Gas('N2', 773)
>>> gas_mixture = cm.GasMixture([gas1, gas2], [0.8, 0.2])
>>> gas_mixture.molecular_weight()
7.2156...
```

viscosity(*method='graham'*)

Gas mixture viscosity.

Calculate viscosity of the gas mixture using Graham's method[1] or the Herning and Zipperer method[2]. For

[1] Thomas Graham. On the Motion of Gases. Philosophical Transactions of the Royal Society of London, vol. 136, pp. 573-631, 1846.

[2] F. Herning and L. Zipperer. Calculation of the Viscosity of Technical Gas Mixtures From the Viscosity of the Individual Gases. Gas-und Wasserfach, vol. 79, pp. 69-73, 1936.

the Graham method, Equation 1 from the Davidson report[3] is used

$$\mu_{mix} = \sum(x_i \cdot \mu_i)$$

where μ_{mix} is viscosity of the gas mixture, x_i is mole fraction [-] of each component, and μ_i is gas viscosity of each component. For the Herning and Zipperer method, Equation 1 from the Davidson report is used

$$\mu_{mix} = \frac{\sum(\mu_i \cdot x_i \cdot \sqrt{MW_i})}{\sum(x_i \cdot \sqrt{MW_i})}$$

where μ_{mix} is viscosity of the gas mixture, x_i is mole fraction [-] of each component, μ_i is gas viscosity of each component, and MW_i is the molecular weight [g/mol] of each gas component.

Parameters

method (*str*) – Method for calculating the gas mixture viscosity, choose graham or herning. Default value is graham.

Returns

mu_mixture (*float*) – Viscosity of the gas mixture in units of microPoise (P).

Examples

```
>>> gas1 = cm.Gas('H2', 773)
>>> gas2 = cm.Gas('N2', 773)
>>> gas_mixture = cm.GasMixture([gas1, gas2], [0.85, 0.15])
>>> gas_mixture.viscosity()
207.34...
```

```
>>> gas1 = cm.Gas('H2', 773)
>>> gas2 = cm.Gas('N2', 773)
>>> gas_mixture = cm.GasMixture([gas1, gas2], [0.85, 0.15])
>>> gas_mixture.viscosity(method='herning')
252.78...
```

References

[3] Thomas Davidson. A Simple and Accurate Method for Calculating Viscosity of Gaseous Mixtures. United States Department of the Interior: Report of Investigations 9456, 1993.

CHAPTER

TWENTYONE

LIQUID

Liquid properties such as heat capacity can be calculated using the method available in the `Liquid` class. The method is based on a correlation and is named accordingly; for example, the `cp_yaws()` method uses Yaws' coefficients to calculate the heat capacity for a range of temperatures.

class chemics.**Liquid**(*formula*)

Liquid properties class.

Parameters
formula (*str*) – Molecular formula of the liquid

Variables

- **formula** (*str*) – Molecular formula of the liquid
- **mw** (*float*) – Molecular weight of the liquid in g/mol

cp_yaws(*temp*, *cas=None*, *disp=False*)

Liquid heat capacity.

Liquid heat capacity as a function of temperature using Yaws' coefficients[1]. The CAS(Chemical Abstracts Service) number may be required for some species.

$$C_p = A + B\,T + C\,T^2 + D\,T^3 + E\,T^4$$

Parameters

- **temp** (*float*) – Temperature of the liquid in Kelvin
- **cas** (*str, optional*) – CAS number of the liquid, required for some species
- **disp** (*bool*) – Display information about the calculation such as the CAS number, applicable temperature range in Kelvin, and values for regression coefficients.

Raises

- **ValueError** – If provided CAS number is not found.
- **ValueError** – If multiple substances found for given formula.
- **ValueError** – If gas chemical formula not found.
- **ValueError** – If given temperataure is out of range for calculation.

Returns
cp (*float*) – Heat capacity of the liquid in J/(molK)

[1] Carl L. Yaws. Heat capacity of liquid Tables 43 and 44 in Yaws' Critical Property Data for Chemical Engineers and Chemists. Published by Knovel, 2014.

Examples

```
>>> liquid = cm.Liquid('CBrF3')
>>> liquid.cp_yaws(250)
107.2774...
```

```
>>> liquid = cm.Liquid('C38H76')
>>> liquid.cp_yaws(400, cas='61828-17-9')
1307.0624...
```

References

CHAPTER

TWENTYTWO

MOLECULAR WEIGHT

Functions to calculate molecular weight of an element, compound, or gas mixture. Note that mole fractions must sum to one otherwise an error is raised.

chemics.**molecular_weight**(*formula*)

Molecular weight.

Tokenize a molecular formula to determine total molecular weight. Calculation is based on atomic weight values from IUPAC[1].

Parameters
formula (*str*) – Molecular formula or element

Returns
mw (*float*) – Molecular weight of the formula or element in g/mol

Examples

```
>>> cm.molecular_weight('C')
12.011...
```

```
>>> cm.molecular_weight('CH4')
16.04...
```

```
>>> cm.molecular_weight('(NH4)2SO4')
132.13...
```

References

[1] IUPAC Periodic Table of the Elements. International Union of Pure and Applied Chemistry, 2016. https://iupac.org/what-we-do/periodic-table-of-elements/.

CHAPTER

TWENTYTHREE

PROXIMATE ANALYSIS BASES

Proximate analysis provides the percentage composition of a material in terms of fixed carbon (FC), volatile matter (VM), ash, and moisture. It can be represented on an as-determined (ad), as-received basis (ar), dry basis (d), and dry-ash free basis (daf).

23.1 Source code

class chemics.**Proximate**(*vals*, *basis*, *ADL=16.36*)

Proximate analysis.

Proximate analysis values expressed as different bases. Such bases are as-determined (ad), as-received (ar), dry (d), and dry ash-free (daf).

Parameters

- **vals** (*list*) – Proximate analysis values given as weight percent (wt. %), g/g (trace elements), or Btu/lb (gross calorific value). Order of values is [FC, VM, ash, moisture].
- **basis** (*str*) – Basis of given proximate analysis values. Options are 'ad' for as-determined basis or 'ar' for as-received basis.
- **ADL** (*float, optional*) – Air-dry loss as weight percent. Default value is 16.36 weight percent.

Variables

- **ad_basis** (*ndarray*) – As-determined basis (ad)
- **ar_basis** (*ndarray*) – As-received basis (ar)
- **d_basis** (*ndarray*) – Dry basis (d)
- **daf_basis** (*ndarray*) – Dry ash-free basis (daf)

Raises

ValueError – If basis is not ad or ar

Example

```
>>> prox = cm.Proximate([47.26, 40.05, 4.46, 8.23], 'ad')
>>> prox.ar_basis
array([39.52..., 33.49...,  3.73..., 23.24...])
```

References

ASTM D3180-15, Standard Practice for Calculating Coal and Coke Analyses from As-Determined to Different Bases, ASTM International, West Conshohocken, PA, 2015.

__str__()

Return string representation.

CHAPTER

TWENTYFOUR

ULTIMATE ANALYSIS BASES

Ultimate analysis provides the percentage composition of a material in terms of carbon (C), hydrogen (H), oxygen (O), nitrogen (N), sulfur (S), ash, and moisture. It can be represented on an as-determined basis (ad), as-received basis (ar), dry basis (dry), and dry-ash free basis (daf).

24.1 Source code

class chemics.**Ultimate**(*vals*, *basis*, *ADL=22*, *HO=True*)

Ultimate analysis.

Ultimate analysis values expressed as different bases. Such bases are as-determined (ad), as-received (ar), dry (d), and dry ash-free (daf).

Parameters

- **vals** (*list*) – Ultimate analysis values given as weight percent (wt. %), g/g (trace elements), or Btu/lb (gross calorific value). Order of values is [C, H, O, N, S, ash, moisture].
- **basis** (*str*) – Basis of given ultimate analysis values. Options are 'ad' for as-determined basis or 'ar' for as-received basis
- **ADL** (*float, optional*) – Air-dry loss as weight percent
- **HO** (*bool, optional*) – If *True* then the given H and O values include H and O from moisture. If *False* then the given H and O values exclude H and O from the moisture content.

Variables

- **ad_basis** (*ndarray*) – As-determined basis (ad)
- **ar_basis** (*ndarray*) – As-received basis (ar)
- **d_basis** (*ndarray*) – Dry basis (d)
- **daf_basis** (*ndarray*) – Dry ash-free basis (daf)

Raises

ValueError – If basis is not ad or ar.

Example

```
>>> ult = cm.Ultimate([60.08, 5.44, 25.01, 0.88, 0.73, 7.86, 9.00], 'ad')
>>> ult.ar_basis
array([46.8624,  6.705 , 39.046 ,  0.6864,  0.5694,  6.1308, 29.02  ])
```

References

ASTM D3180-15, Standard Practice for Calculating Coal and Coke Analyses from As-Determined to Different Bases, ASTM International, West Conshohocken, PA, 2015.

__str__()

Return string representation.

CHAPTER

TWENTYFIVE

WOOD

Wood properties can be calculated using the functions available in the wood module.

chemics.**cp_wood**(*x*, *tk*)

Wood heat capacity.

Heat capacity of wood based on moisture content and temperature

$$c_{p,x} = \left(c_{p0} + c_{pw}\frac{x}{100}\right) / \left(1 + \frac{x}{100}\right) + A_c$$

where $c_{p,x}$ is heat capacity of wet wood [kJ/(kg K)], c_{p0} is heat capacity of dry wood [kJ/(kg K)], c_{pw} is heat capacity of water as 4.18 kJ/(kg K), x is moisture content [%], and Ac is an adjustment factor that accounts for the additional energy in the wood–water bond[1].

The c_{p0} term is determined from

$$c_{p0} = 0.1031 + 0.003867\,T$$

where T is temperature in Kelvin. The A_c term is calculated from

$$A_c = x(b_1 + b_2 T + b_3 x)$$

with $b_1 = -0.06191$, $b_2 = 2.36e \times 10^{-4}$, and $b_3 = -1.33 \times 10^{-4}$.

Parameters

- **x** (*float*) – Moisture content as percent
- **tk** (*float*) – Temperature in Kelvin

Returns

cp (*float*) – Heat capacity of wood in kJ/(kgK)

Example

```
>>> cm.cp_wood(12, 340)
1.91...
```

[1] Samuel V. Glass and Samuel L. Zelinka. Moisture Relations and Physical Properties of Wood. Chapter 4 in Wood Handbook, pp. 1-19, 2010.

References

chemics.**k_wood**(*gb*, *so*, *x*)

Wood thermal conductivity.

Thermal conductivity of wood based on moisture content, volumetric shrinkage, and basic specific gravity

$$k = G_x(B + Cx) + A$$

where k is thermal conductivity [W/(mK)] of wood, G_x is specific gravity [-] based on volume at moisture content x [%] and A, B, C are constants.

The G_x term is determined from

$$G_x = \frac{G_b}{1 - S_x/100}$$

where G_b is basic specific gravity [-] and S_x is volumetric shrinkage [%] from green condition to moisture content x.

The S_x term is calculated from

$$S_x = S_o \left(1 - \frac{x}{MC_{fs}}\right)$$

where S_o is volumetric shrinkage [%] from Table 4-3[2] and MC_{fs} is the fiber saturation point assumed to be 30% moisture content.

Parameters

- **gb** (*float*) – Basic specific gravity
- **so** (*float*) – Volumetric shrinkage in percentage
- **x** (*float*) – Moisture content in percentage

Returns

k (*float*) – Thermal conductivity in W/(mK)

Example

```
>>> cm.k_wood(0.54, 12.3, 10)
0.1567...
```

References

[2] Samuel V. Glass and Samuel L. Zelinka. Moisture Relations and Physical Properties of Wood. Chapter 4 in Wood Handbook, pp. 1-19, 2010.

CHAPTER

TWENTYSIX

PACKAGE INDEX AND MODULES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX