

```

print('==== Problem 1 ====')
Sh_y, Sh_x, p = 10, 36, 1*atm
yB, xB, t = .01, .005, 298*kelvin
vpA = q(126, 'mmHg').to('atm').magnitude * atm

print('==== Liquid Phase ====')
c = (1.012 * gram / cm**3) / (18 * gram / mol)
kX = HW4.SherwoodSolver(Sh_x, # Sherwood Number
    1*cm, # characteristic length
    c, # molarity: mol/volume
    q(HW4.WilkeChangDiffusivity(
        298, # temperature in K
        15.999 + 2, # molecular weight of B in g/mol
        0.8818, # viscosity of B in cp
        42.78, # molar volume of A at normal boiling in cm**3/mol
        2.6 # association factor ??
    ), 'cm**2/s').magnitude * cm**2 / second, # diffusivity: area/time
    symbols('k'), # mtc: mol/(area*time)
    symbols('k'), # variable to solve for
    False # Low mass transfer
)

print('==== Gas Phase ====')
c = convert_to(p / (molar_gas_constant * t), mol/m**3)
print(f"Molarity: {c}")

kY = HW4.SherwoodSolver(Sh_y, # Sherwood Number
    1*cm, # characteristic length
    c, # molarity: mol/volume
    q(HW4.FullerDiffusivity(
        1, # pressure in atmospheres
        298, # temperature in kelvin
        32.042, # molecular weight species A
        14.007 * 2, # molecular weight species B
        15.9 + (2.31 * 4) + 6.11, # diffusion volume species A
        18.5, # diffusion volume species B
        'Methanol', # name of species A
        'N2' # name of species B
    ), 'cm**2/s').magnitude * cm**2 / second, # diffusivity: area/time
    symbols('k'), # mtc: mol/(area*time)
    symbols('k'), # variable to solve for
    False # Low mass transfer
)

xI, yI = HW4.Solve_MoleFracInterface(
    HW4.RaoultLaw(
        xI = symbols('xI'),
        yI = symbols('yI'),
        vpA = vpA,
        p = p,
        gamma = 1.725,
        idealSolution = False
    ),
    xB, # mole fraction at bulk for x phase
    yB, # mole fraction at bulk for y phase
    kX, # mtc of phase x
    kY, # mtc of phase y
)

```

```

xI, yI = HW4.Solve_MoleFracInterface(
    HW4.RaoultsLaw(
        xI = symbols('xI'),
        yI = symbols('yI'),
        vpA = vpA,
        p = p,
        gamma = 1.725,
        idealSolution = False
    ),
    xB, # mole fraction at bulk for x phase
    yB, # mole fraction at bulk for y phase
    kX, # mtc of phase x
    kY, # mtc of phase y
)

nAX, nAY = HW4.FluxBetweenPhases(xB, xI, yB, yI, kX, kY)

oKX, oKY = HW4.SOLVE_OVERALL_MTC(
    kX,
    kY,
    HW4.CalcSlope_OMTC(vpA, p,
        lawUsed='raoults',
        idealSolution=False,
        gamma=1.725
    ),
    symbols('oK'),
    symbols('oK'),
)

HW4.PhaseResistances(kX, oKX, kY, oKY)

```

```

==== Problem 1 ====
==== Liquid Phase ====
WilkeChangDiffusivity: 1.8×10-5 cm2/s
Mass transfer coefficient: 0.363625885467759*mole/(meter**2*second)
==== Gas Phase ====
Molarity: 40.8946187070611*mole/meter**3
mole_weight_ratios_Methanol_N2: 29.89
FullerDiffusivityMethanol_N2: 0.166489 cm2/s
Mass transfer coefficient: 0.680849686453113*mole/(meter**2*second)
==== Raoults law to solve mole fractions at interface ====
Mole fraction of [x,y] at the interface [0.0154504904410722, 0.00441863759972676]
Flux from y to x phase: 3.80006884020723e-7*mole/(centimeter**2*second)
Flux from x to y phase: 3.80006884020723e-7*mole/(centimeter**2*second)
Slope of overall mass transfer coefficient, m: 0.285986882848757
Overall MTC for x phase: 1.26809979614136e-5*mole/(centimeter**2*second)
Overall MTC for y phase: 4.43411873827792e-5*mole/(centimeter**2*second)
Phase resistance for x phase: 0.34873749
Phase resistance for y phase: 0.65126251
Total phase resistance sums to: 1.00000000000000

```

Base Methods

```
@staticmethod
def SherwoodSolver(Sh: float, # Sherwood Number
                    L, # characteristic length
                    c, # molarity: mol/volume
                    d_AB, # diffusivity: area/time
                    k = cnst.k, # mtc: mol/(area*time)
                    solveFor = cnst.k, # variable to solve for
                    low_mass_transfer: bool = False, # low mass transfer rates
                    ):

    if low_mass_transfer:
        soln = solve(Eq((k * L) / d_AB, Sh), solveFor)
    else:
        soln = solve(Eq((k * L) / (c * d_AB), Sh), solveFor)

    print(f'Mass transfer coefficient: {convert_to(soln[0], mol / (m**2 * second))}')
    return soln[0]
```

```
@staticmethod
def WilkeChangDiffusivity( # A:solute, B:solvent
                            t: float, # temperature in K
                            mw_B: float, # molecular weight of B in g/mol
                            mu_B: float, # viscosity of B in cp
                            mVol_A_normal: float, # molar volume of A at normal boiling in cm**3/mol
                            aF: float = 1.5 # association factor {2.6: water, 1.9: methanol, 1.5: ethanol, 1.0: unassociated}
                            ):

    d_AB = (7.4 * 10**-8 * (aF*mw_B)**(1/2) * t) / (mu_B * mVol_A_normal**0.6)

    print(f'WilkeChangDiffusivity: {q(round(d_AB, 6), "cm**2/s")}')
    return d_AB
```



```

@staticmethod
def FullerDiffusivity(
    p: float, # pressure in atmospheres
    t: float, # temperature in kelvin
    molW1: float, # molecular weight species A
    molW2: float, # molecular weight species B
    dVol1: float, # diffusion volume species A
    dVol2: float, # diffusion volume species B
    specA: str = 'A', # name of species A
    specB: str = 'B', # name of species B
):
    m_AB = 2 / ((1 / molW1) + (1 / molW2))
    d_AB = (0.00143 * t**1.75) / (p * m_AB**(1/2) * (dVol1**(1/3) + dVol2**(1/3))**2 )

    print(f'mole_weight_ratios_{specA}_{specB}: {round(m_AB,2)}')
    print(f'FullerDiffusivity_{specA}_{specB}: {q(round(d_AB, 6), "cm**2/s")}')

    return d_AB

```

```

@staticmethod
def RaoultsLaw(xI: float, # mole fraction of x at interface
    yI: float, # mole fraction of y at interface
    vpA, # vapor pressure of A
    p, # Total Pressure
    gamma: float = 1, # Activity coefficient methods
    idealSolution: bool = False
):
    # DISTILLATION
    # Ideal solutions
    print('==== Raoults law to solve mole fractions at interface ====')
    if idealSolution:
        eq = Eq((vpA / p) * xI, yI)
    else:
        if gamma == 1:
            raise Exception('Solution is non-Ideal and gamma is default value')
        else:
            eq = Eq(((gamma * vpA) / p) * xI, yI)

    return eq

```

```

@staticmethod
def FluxBetweenPhases(xB: float,
                      xI: float,
                      yB: float,
                      yI: float,
                      kX,
                      kY
                      ):
    res = []
    for phase in ['x', 'y']:
        if phase == 'x':
            nA = kX * (xI - xB)
            otherPhase = 'y'
        else:
            nA = kY * (yB - yI)
            otherPhase = 'x'

        print(f'Flux from {otherPhase} to {phase} phase: {nA}')
        res.append(nA)

    return res[0], res[1]

```

```

@staticmethod
def Solve_MoleFracInterface(eq2,
                             xB: float,
                             yB: float,
                             kX,
                             kY
                             ):
    xI, yI = symbols('xI'), symbols('yI')

    soln = solve((
        Eq(kY * (yB - yI), kX * (xI - xB)),
        eq2
    ),
    (xI, yI))

    xI, yI = soln[xI], soln[yI]
    print(f'Mole fraction of [x,y] at the interface [{xI}, {yI}]')
    return [xI, yI]

```

```

@staticmethod
def CalcSlope_OMTC(
    He_or_vpA, # Henry coefficient (henrys) or vapor pressure of A (raoult's)
    p, # total pressure
    lawUsed: str = 'henrys', # henrys or raoult's
    idealSolution: bool = True, # If raoult's law, if solution is ideal
    gamma: float = 1.0, # Activity coefficient
):
    if lawUsed.lower() == 'henrys' or \
        (lawUsed.lower() == 'raoult's' and idealSolution):
        m = He_or_vpA / p

    elif lawUsed.lower() == 'raoult's' and not idealSolution:
        m = (He_or_vpA / p) * gamma
    else:
        raise Exception('LawUsed: invalid input, law not found')

    print(f"Slope of overall mass transfer coefficient, m: {m} ")
    return m

```

```

@staticmethod
def SOLVE_OVERALL_MTC(kX, # mtc in x phase
    kY, # mtc in y phase
    He, # Henry's law coefficient
    p, # Total pressure
    oK, # Overall MTC for a phase
    solveFor, # input var being solved
):
    m = He / p

    res = []
    for phase in ['x', 'y']:
        if phase == 'x':
            soln = solve(Eq((1 / kX) + (1 / (m * kY))), 1 / oK, solveFor)
        else:
            soln = solve(Eq((1 / kY) + (m / kX)), 1 / oK, solveFor)

        print(f'Overall MTC for {phase} phase: {soln[0]} ')
        res.append(soln[0])

    return res[0], res[1]

```

```

@staticmethod
def PhaseResistances(
    kX,
    bigKX,
    kY,
    bigKY,
):

    res = []
    for phase in ['x', 'y']:
        if phase == 'x':
            k, bigK = kX, bigKX
        else:
            k, bigK = kY, bigKY

        pR = (1 / k) / (1 / bigK)

        print(f'Phase resistance for {phase} phase: {round(pR, 8)}')
        res.append(pR)

    one = res[0] + res[1]
    if round(one, 4) != 1:
        raise Exception(f"Total resistance does not add to one: {one}")
    else:
        print(f"Total phase resistance sums to: {one}")

```