- Goal: provide comsol a 3-argument (T, xWF6, xH2) table to interpolate the thermal diffusion coefficients D_T(k)

In [9]:
```
from IPython.core.display import display, HTML
import pygments
display(HTML("<style>.container { width:266mm !important; }</style>")) # to set cell widths
formatter = pygments.formatters.get_formatter_by_name('html', linenos='inline')
```

In [1]:
```
#Version 1.4
import numpy as np
import pandas as pd
import time
```

In [2]:
```
###==== INPUT DATA ========###
dfAllPolys = pd.read_excel('C:/Daten/Modeling/Stoffdaten/Polynomals_for_T_star_related_quantities.xlsx')

R_const = 8.31446 #[J mol^-1 K^-1]
speciesData={0:{'species':'H_2',
                'molar mass [kg/mol]':0.00201588,
                '(epsilon/k_B)[K]':lambda T:59.7,
                'sigma[m]':lambda T:2.827E-10,
               },
             1:{'species':'WF_6',
                'molar mass [kg/mol]':0.297830419,
                '(epsilon/k_B)[K]': lambda T:(819.92 -1.0619*T + 0.000619*T**2 - (6.62E-8)*T**3),
                'sigma[m]': lambda T:(4.9734 + 0.0009284*T + 6.5815E-7*T**2 - (7.315E-10)*T**3)*10**-10,
               },
             2:{'species':'HF',
                'molar mass [kg/mol]':0.020006,
                '(epsilon/k_B)[K]':lambda T:330,
                'sigma[m]':lambda T:3.148E-10,
               },}
'''
2:{'species':'SiH_4',
   'molar mass [kg/mol]':0.03212,
   '(epsilon/k_B)[K]':lambda T:207.6,
   'sigma[m]':lambda T:4.084E-10,
  },
'''
#references: polynomals Kleijn1991, p. 37; other data as in Table 8.1 in present dissertation
###========================###

#number of species:
N = len(speciesData)
#zip molar masses:
m_={}
for key in speciesData: m_[key] = speciesData[key]['molar mass [kg/mol]']

def create_T_dep_Dicts(T):
    '''returns T dependent literature data as compact indexed dictionaries for use in formulas;
    this has to be executed in the beginning of the T loop, which will fill the look up table,
    to provide the function calc_Md_and_detMd with needed T dependent local variables'''
    sigma_, epsilonDivKb_ = {}, {}
    for k in range(N): sigma_[k]=speciesData[k]['sigma[m]'](T)
    for k in range(N): epsilonDivKb_[k]=speciesData[k]['(epsilon/k_B)[K]'](T)
    return sigma_, epsilonDivKb_

def getQuantityValue(Quantity,T_star):
    '''
    returns value for requested Quantity (Omega_i or A*-C*)
    looks up the polynomals in table 2.5 in [Kleijn1993] depending on T_star.
    '''
    dfPolys = dfAllPolys.query('Quantity == @Quantity and T_star_low <= @T_star < T_star_high')
    return float(dfPolys['a0']+dfPolys['a1']*T_star+dfPolys['a2']*T_star**2+dfPolys['a3']*T_star**3)

def calc_Md_and_detMd(T,f_):
    '''
    returns the matrix in the denominator in the equation for D_T [Kleijn1993, Eq. 2.89] and its determinant
    needs the for loop variables T, f_ as input
    needs m_, sigma_, epsilonDivKb_, c_p_ as lokal variables
    '''
    #following lambda is an unnamed python function not to be confused with thermal conductivity
    sigma = lambda i,j: 0.5*(sigma_[i]+sigma_[j]) #[m] [Eq. 2.69]
    epsilonDivKb = lambda i,j: (epsilonDivKb_[i]*epsilonDivKb_[j])**0.5 #[K] [Eq. 2.70]
    T_star = lambda i,j: T/epsilonDivKb(i,j) #[-] [Eq. 2.71]

    Omega_mu = lambda i,j: getQuantityValue('Omega_mu',T_star(i,j))
    Omega_D  = lambda i,j: getQuantityValue('Omega_D',T_star(i,j))
    A_star   = lambda i,j: getQuantityValue('A_star',T_star(i,j))
    B_star   = lambda i,j: getQuantityValue('B_star',T_star(i,j))
    C_star   = lambda i,j: getQuantityValue('C_star',T_star(i,j))

    #to obtain lamb (thermal heat conductivity in W/(m*K) [p.43, Eq. 2.80 = 2.81 for i = j]:
    lamb = lambda i,j: (0.00263*(T*(m_[i]+m_[j])/(2*m_[i]*m_[j]))**0.5
                        /(sigma(i,j)*1E10)**2*Omega_mu(i,j)) #(K/kg*mol)**0.5/m²...units does not fit,
                                                             #but final result fit to Fig 2.5 with this


    def L_00(i,j): #[Eq. 2.90 and 2.91]
        result=0.
        if i!=j:
            result=2.*f_[i]*f_[j]/(A_star(i,j)*lamb(i,j))
            for n in range(N):
                if n!=i:
                    result+=2.*f_[i]*f_[n]*m_[j]/(m_[i]*A_star(i,n)*lamb(i,n))
                    # the f_[j] is correct compared to Hirschfelder1967
                    # with f_[i] as in the book of Kleijn, its not possible to reproduce Fig 2.5
        return result


    def L_01(i,j):
        result=0.
        if i==j:
            for n in range(N):
                if n!=i:
                    result+=(5.*f_[i]*f_[n]*m_[n]*(6./5.*C_star(i,n)-1)
                             /((m_[i]+m_[n])*A_star(i,n)*lamb(i,n)))
        if i!=j:
            result=(-5.*f_[i]*f_[j]*m_[i]*(6./5.*C_star(i,j)-1.)
                    /((m_[i]+m_[j])*A_star(i,j)*lamb(i,j)))
        return result

    def L_10(i,j): return m_[j]/m_[i]*L_01(i,j)

    def L_11(i,j):
        result=0.
        if i==j:
            result=-4.*f_[i]**2/lamb(i,j)
            for n in range(N):
                if n!=i:
                    result-=(2.*f_[i]*f_[n]*(15./2.*m_[i]**2+25./4.*m_[n]**2
                                             -3.*m_[n]**2*B_star(i,n)
                                             +4.*m_[i]*m_[n]*A_star(i,n))
                             /((m_[i]+m_[n])**2*A_star(i,n)*lamb(i,n)))
        if i!=j:
            result=(2*f_[i]*f_[j]*m_[i]*m_[j]/((m_[i]+m_[j])**2*A_star(i,j)*lamb(i,j))
                    *(55./4.-3.*B_star(i,j)-4.*A_star(i,j)))
        return result

    #filling of the matrix according to gas species i,j,
    #which start here at zero and not with one as in the book,
    #as matrix indizes and range(...) also starts at zero

    Md = np.zeros((2*N,2*N)) # matrix in denominator in Eq. 2.89
    for i in range(N): #i,j=0,1,2,... #i=row, j=column
        for j in range(N):
            Md[i,j]     = L_00(i,j)
            Md[i,j+N]   = L_01(i,j)
            Md[i+N,j]   = L_10(i,j)
            Md[i+N,j+N] = L_11(i,j)
    return Md, np.linalg.det(Md) #so that det(Md) needs to be calculated only once for all k

def kroneckerDelta(i,j):  #carefull: np.kron(i,j) is something else
    if i == j: return 1.
    else: return 0.

def calc_D_T(Md, detMd, k, f_):
    '''
    Returns thermal diffusion coefficients for different species k, which do share the same Md and detMd.
    '''
    Mc = np.zeros((2*N+1, 2*N+1))# matrix in counter
    Mc[:-1,:-1] = Md
    for j in range(N):
        Mc[N+j, 2*N] = f_[j] #last column (index start at 0, thus column 2*N e.g.=6 is the 7. column)
        Mc[2*N, j]   = f_[j]*kroneckerDelta(j,k) #last row
    return -8*m_[k]/5/R_const*np.linalg.det(Mc)/detMd

######## build look-up-dataframe for D_T(k): #######
df_D = pd.DataFrame()

###=== T & molefractions loop CONFIG ========###
T_loop    = np.linspace(300,1100,18)
f_0_loop = np.linspace(1e-6,1-1e-6,11)
f_0 = f_0_loop[0] #will be replaced by other is elements of f_0_loop during following nested for-loops
f_1_loop = [1e-6,1e-2]+np.logspace(-1.6,-0.07,12).tolist()+[1-f_0]

t0 = time.clock()
for T in T_loop:
    sigma_,epsilonDivKb_ = create_T_dep_Dicts(T)
    for f_0 in f_0_loop:
        for f_1 in f_1_loop:
            if f_0 > 0 and f_1 > 0 and f_0+f_1 < 1:
                f_=[f_0, f_1, 1-f_0-f_1]
                Md,detMd = calc_Md_and_detMd(T,f_)
                single_row = {'$T$ [K]':T}
                for k in range(N):
                    single_row['$x('+speciesData[k]['species']+')$'] = f_[k]
                    single_row['$D_T('+speciesData[k]['species']+')$ [kg/(m*s)]'] = calc_D_T(Md,detMd,k,f_)
                df_D = df_D.append(pd.DataFrame.from_records([single_row]),sort = False)

neededTime = time.clock() - t0
df_D.head()
```

Out[2]:

	T [K]	$x(H_2)$	$D_T(H_2)$ [kg/(m*s)]	$x(WF_6)$	$D_T(WF_6)$ [kg/(m*s)]	$x(HF)$	$D_T(HF)$ [kg/(m*s)]
0	300.0	0.000001	-8.608532e-13	0.000001	-7.895946e-12	0.999998	8.756800e-12
0	300.0	0.000001	-8.396021e-13	0.010000	-6.721363e-08	0.989999	6.721447e-08
0	300.0	0.000001	-8.114028e-13	0.025119	-1.361034e-07	0.974880	1.361042e-07
0	300.0	0.000001	-7.954698e-13	0.034601	-1.659782e-07	0.965398	1.659790e-07
0	300.0	0.000001	-7.751778e-13	0.047663	-1.958734e-07	0.952336	1.958741e-07

In [3]:
```
print('needed time: ' + '%.1f'%(neededTime/60) + ' min')
```

needed time: 26.9 min

In [4]:
```
# ===== EXPORT for Comsol ==============

fillingUpSpecies = 2 #the species that fills up the mole fraction to 1
                     #and is thus not needed as argument

myColumnOrder = ['$T$ [K]']
for k in range (N):
    if k != fillingUpSpecies:
        myColumnOrder += df_D.filter(like='x')\
                             .filter(like=speciesData[k]['species']).columns.tolist()
for k in range (N):
    myColumnOrder += df_D.filter(like='D_T')\
                         .filter(like=speciesData[k]['species']).columns.tolist()

destiFile = r'P:\WILMA\Leonard Raumann\Comsol\single fiber tube\input data\D_T.txt'
df_D[myColumnOrder].to_csv(destiFile,index=False, sep='\t', float_format='%1.2e',)
df_D[myColumnOrder].head()
```

Out[4]:

	T [K]	$x(H_2)$	$x(WF_6)$	$D_T(H_2)$ [kg/(m*s)]	$D_T(WF_6)$ [kg/(m*s)]	$D_T(HF)$ [kg/(m*s)]
0	300.0	0.000001	0.000001	-8.608532e-13	-7.895946e-12	8.756800e-12
0	300.0	0.000001	0.010000	-8.396021e-13	-6.721363e-08	6.721447e-08
0	300.0	0.000001	0.025119	-8.114028e-13	-1.361034e-07	1.361042e-07
0	300.0	0.000001	0.034601	-7.954698e-13	-1.659782e-07	1.659790e-07
0	300.0	0.000001	0.047663	-7.751778e-13	-1.958734e-07	1.958741e-07

In [ ]:
```
```