Electronic Supplementary Material (ESI) for Physical Chemistry Chemical Physics.

Electronic Supplementary Information (ESI)

A new technique for determining the refractive index of ices at cryogenic temperatures

James W. Stubbing[1]♣, Martin R. S. McCoustra[2] and Wendy A. Brown[1]*

[1]Department of Chemistry, School of Life Sciences, University of Sussex, Falmer, Brighton, BN1 9QJ, United Kingdom

[2]Institute of Chemical Sciences, Heriot-Watt University, Edinburgh, EH14 4AS, United Kingdom

*Corresponding author: w.a.brown@sussex.ac.uk

♣Present address: Surrey Space Centre, University of Surrey, Guildford, Surrey, GU2 7XH, United Kingdom

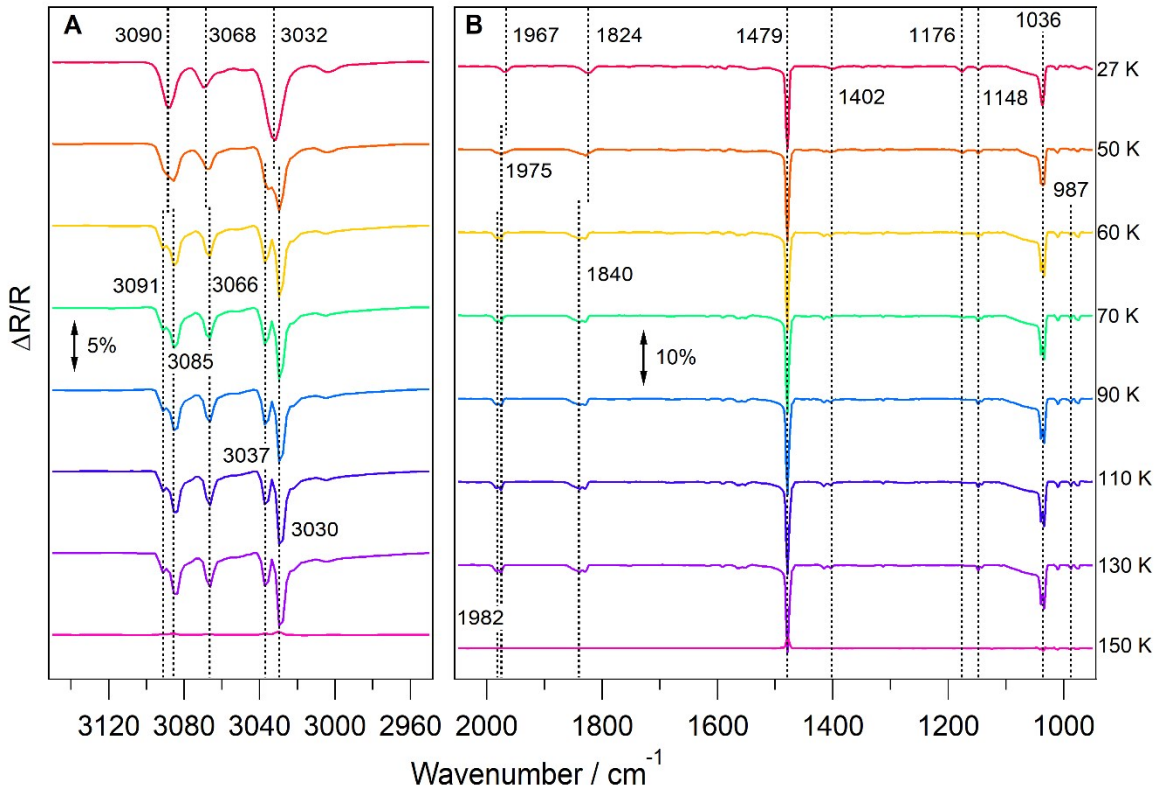

Figure E1. *The effect of annealing on the RAIR spectrum of 500 L_m C_6H_6 deposited at 27 K on HOPG. The annealing temperature is shown on the right of the figure. Band intensities and positions are shown in each panel.*

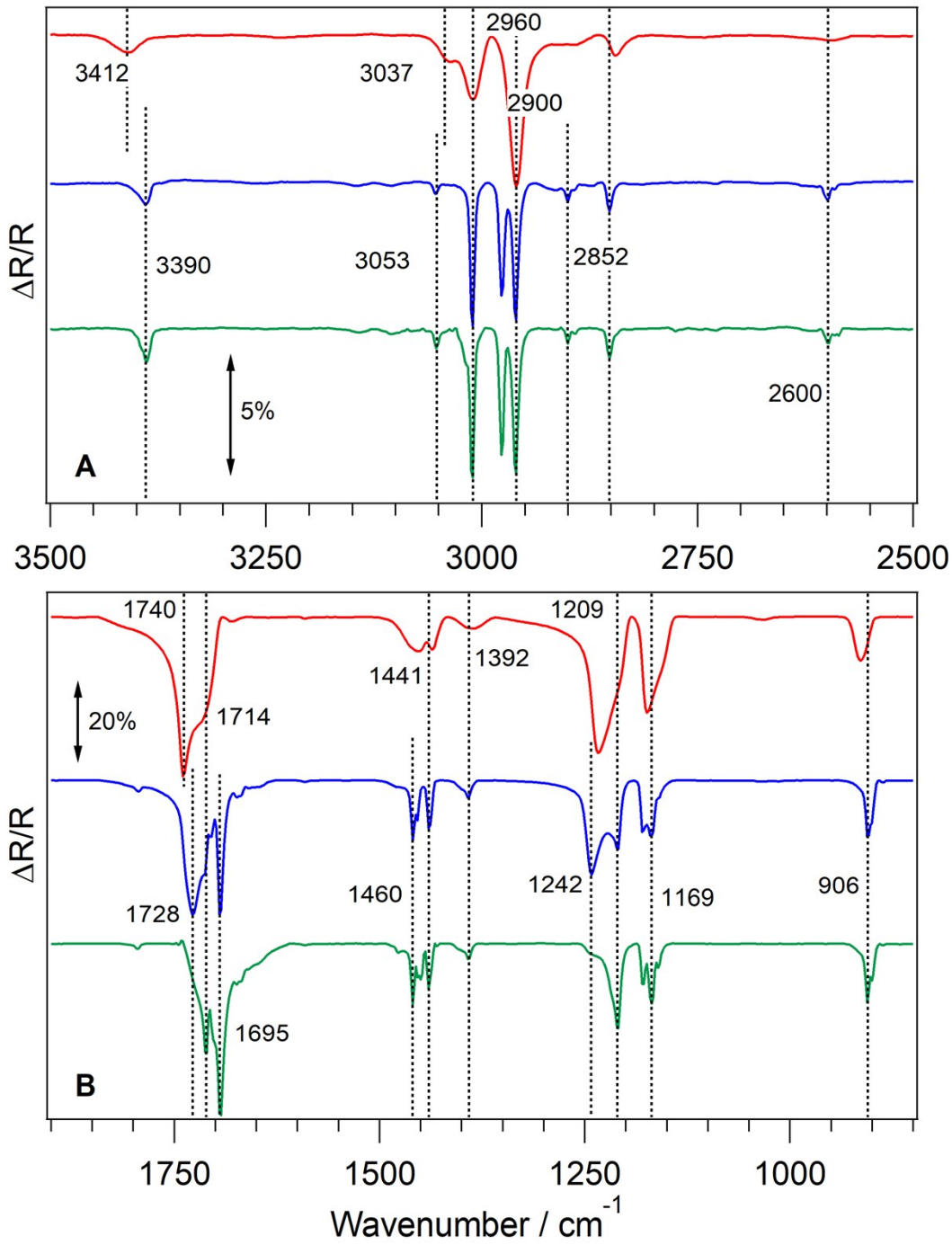

Figure E2. *RAIR spectra of 300 L_m MF deposited on HOPG at 27 K (red trace), deposited at 27 K and annealed to 100 K (blue trace) and deposited at 105 K (green trace). Band intensities and positions are shown on the figure. The two crystalline phases examined in this work are represented by the red and green traces.*

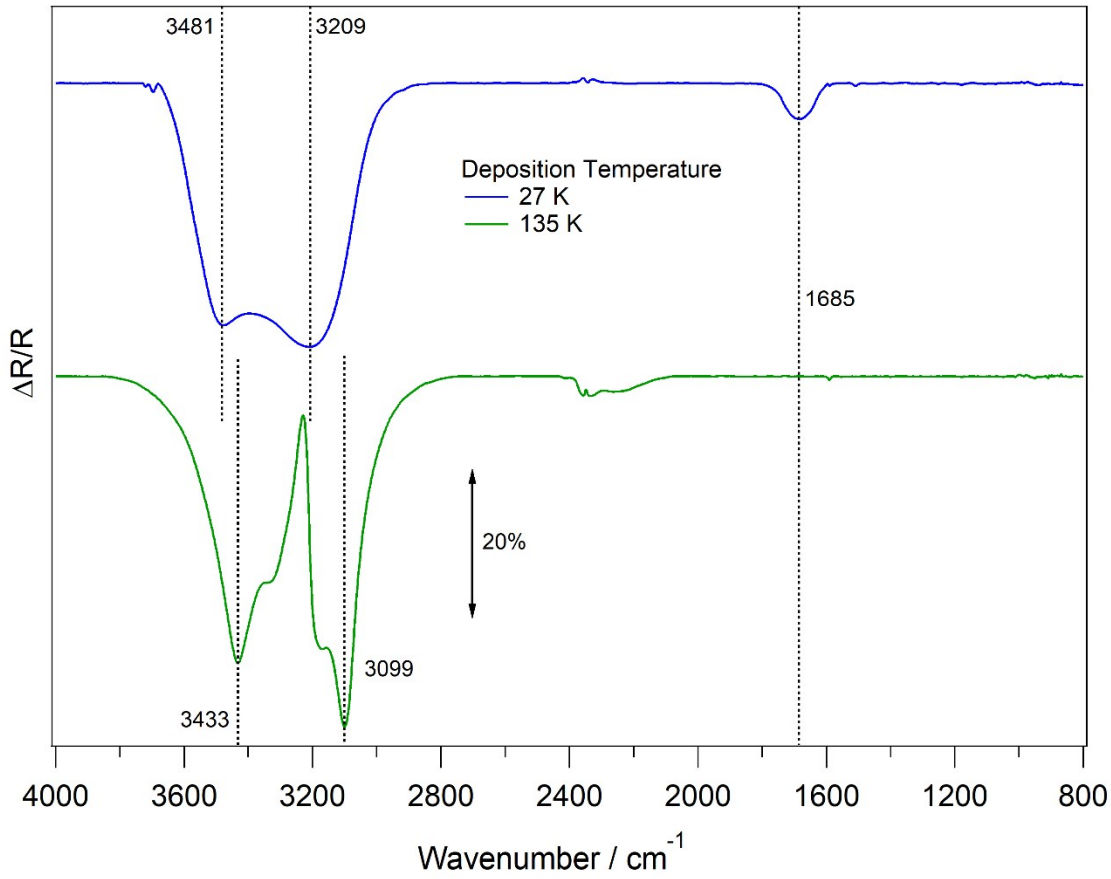

Figure E3. *RAIR spectra of 100 L_m H_2O deposited on HOPG at 27 K (blue trace) and 135 K (green trace). Band intensities and positions are shown on the figure.*

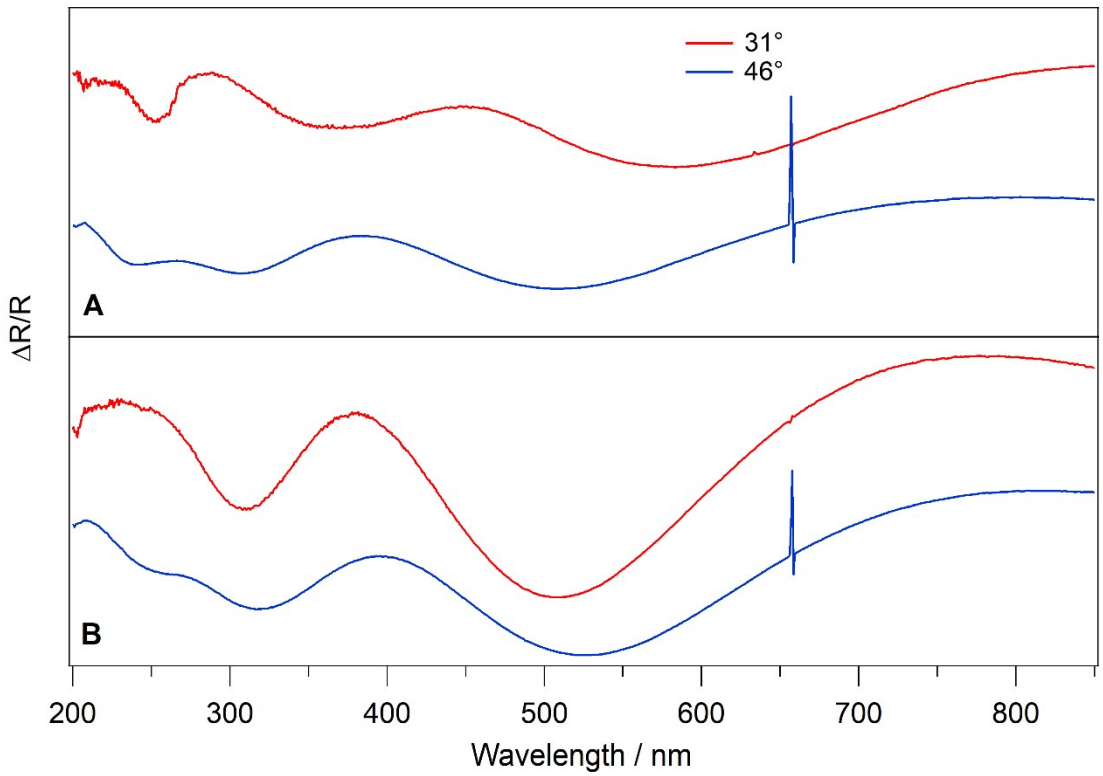

Figure E4. *UV/visible spectra of 300 L_m amorphous (A) and crystalline (B) MF at two reflection angles.*

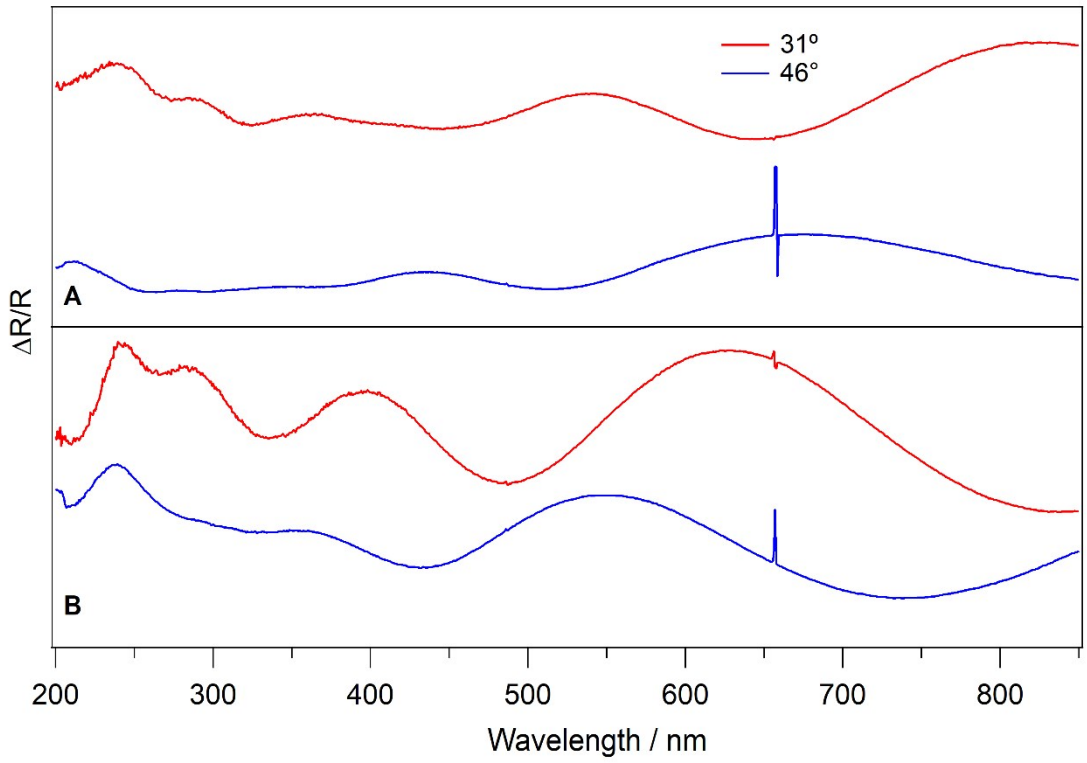

Figure E5. *UV/visible spectra of 150 L_m ASW (A) and CI (B) at two reflection angles.*

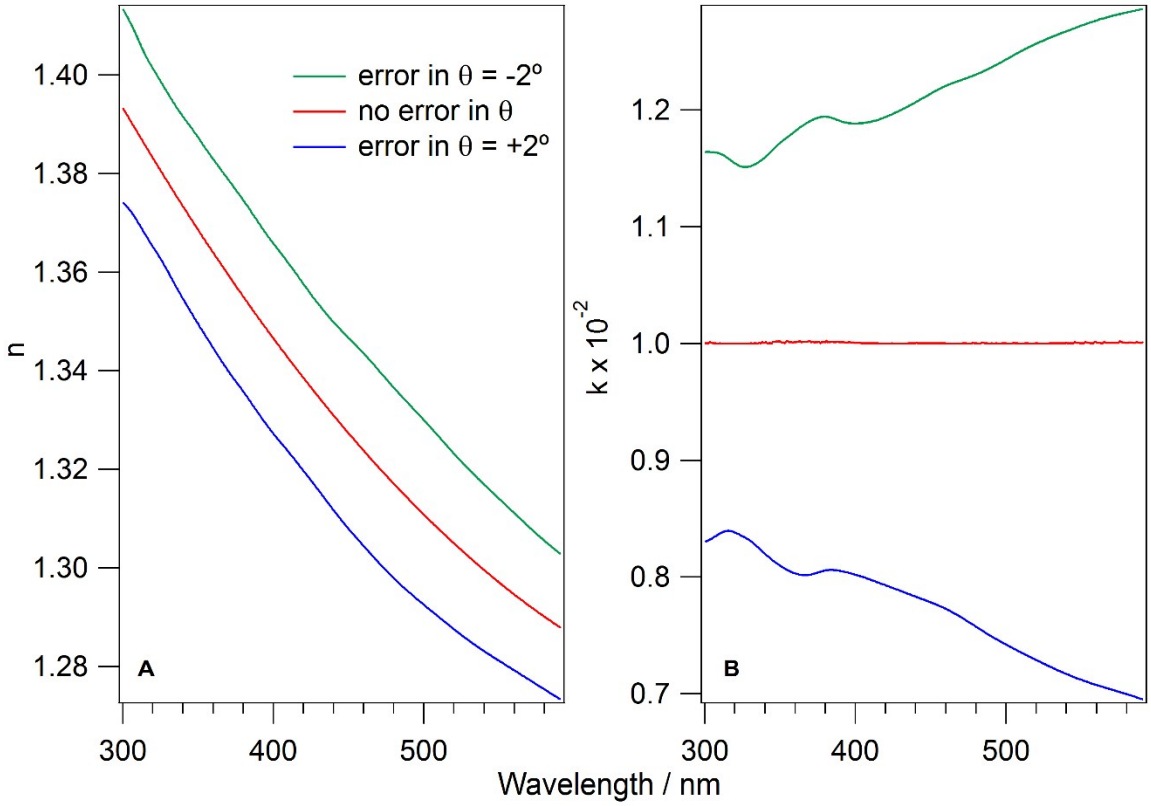

Figure E6. *Real (panel A) and complex (panel B) part of the refractive index of amorphous H_2O ice as a function of wavelength. Red traces: as shown in Figures 3B and 3C. Blue traces: the result of the code when the reflection angle error is + 2°. Green traces: the result of the code when the reflection angle error is -2°.*

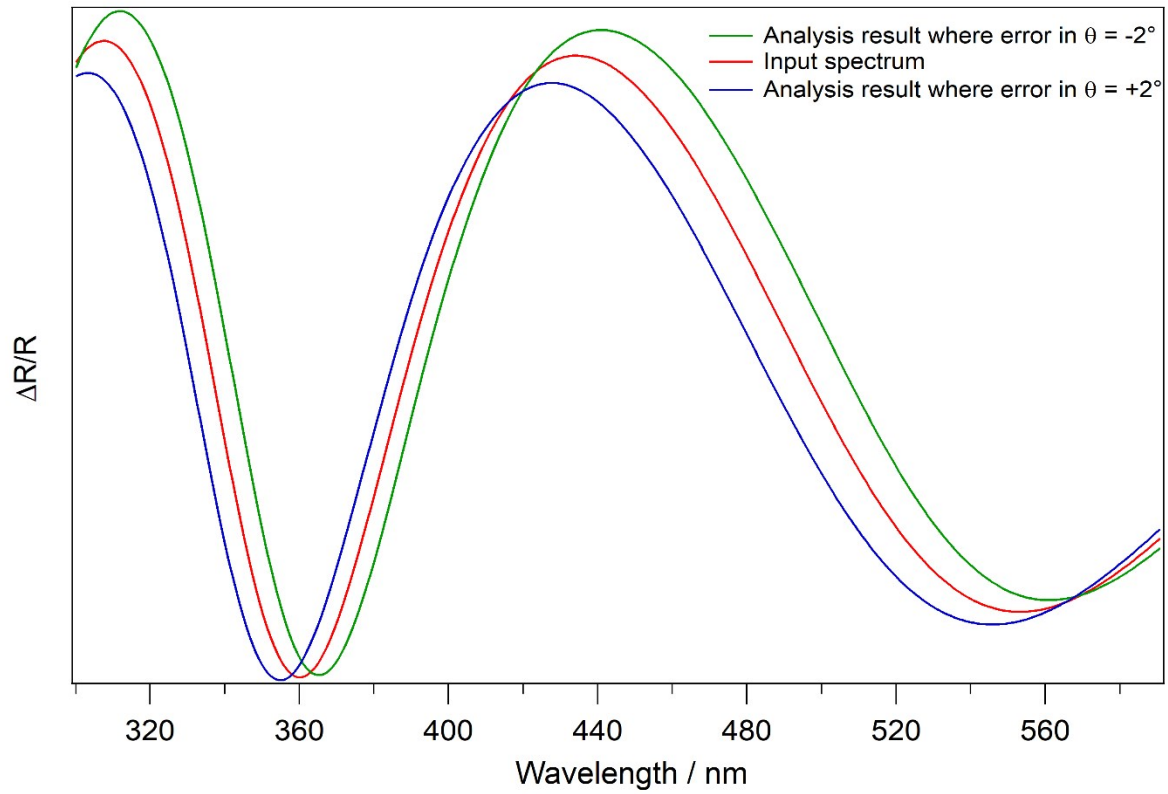

Figure E7. *The results of the code when a small error on the reflection angle is introduced.*

Table E1. *RAIRS bands of MF observed in this work using several deposition temperatures. Symbols: ν = stretching, δ = deformation, β = bending, ρ = rocking, OT = overtone, as = asymmetric, s = symmetric, (sh) = shoulder and * = tentative assignment.*

	Wavenumber / cm^{-1}		
Assignment	Deposited at 27 K	Deposited at 27 K and annealed to 100 K	Deposited at 105 K
$\nu(C{=}O)_{OT}$	3412	3390	3388
$\nu(CH_3)_{as}$	3037 (sh)	3055	3053
$\nu(CH_3)_{as}$	3010	3010	3010
ν(C-H)	2960	2978/2960	2978/2960
$*\delta(CH_3)_{as\text{-}OT}$	-	2900	2900
$*\delta(CH_3)_{s\text{-}OT}$	2845	2852	2852
$*\nu(C\text{-}O)_{OT}$	2594	2600	2600
ν(C=O)	1740	1728	1728 (sh)
ν(C=O)	1714 (sh)	1714 (sh)	1712
ν(C=O)	1678	1695	1695
$\delta(CH_3)_{as}$	1454	1460/1454 (sh)	1460/1450
$\delta(CH_3)_{s}$	1437	1441	1441
β(C-H)	1387	1392	1392
ν(C-O)	1234	1242/1209	1242 (sh)/1209
$\rho(CH_3)$	1175	1180/1169/1159 (sh)	1180/1169/1161 (sh)
ν(C-O)	914	906	906/900 (sh)

Appendix E1: *Python code used to determine the wavelength dependent complex refractive indices of ices.*

```
#Code to determine wavelength dependent complex refractive indices (n
and k) values of ices.
#Written in Python 3.5 in July/August 2018 by James Stubbing, with much
assistance from Dr. Adam Baskerville.
#The code calculates a deltaR/R value, which is then optimised by the
least squares method by comparison to
#experimental data at a range of reflection angles.


#import relevant modules
import numpy as np                                      #numpy module for
mathematical operations
from scipy.optimize import least_squares                #least squares
operation from scipy module
import pandas as pd                                     #pandas module for
reading files
import time                                             #allows the time
the program takes to run to be recorded

start = time.time()   #Starts counter to see how long code takes to run

#sets pandas to show all data in the output files
pd.set_option('display.max_rows', 10000)
pd.set_option('display.max_columns', 10000)


#list containing all experimental reflection angles
#EDIT AS REQUIRED
angle = [np.deg2rad(31), np.deg2rad(39), np.deg2rad(46),
np.deg2rad(53), np.deg2rad(58), np.deg2rad(64), np.deg2rad(68)]


#empty lists for results
#ADD AS MANY THICKNESS VALUES AND ANLGES AS REQUIRED
ntarget = []                #Output value of n
ktarget = []                #Output value of k
sim_31_1 = []               #simulated spectrum using determined n and
k values at angle of 31 degrees for first thickness
sim_39_1 = []
sim_46_1 = []
sim_53_1 = []
sim_58_1 = []
sim_64_1 = []
sim_68_1 = []
sim_31_2 = []
sim_39_2 = []
sim_46_2 = []
sim_53_2 = []
sim_58_2 = []
sim_64_2 = []
sim_68_2 = []
sim_31_3 = []
sim_39_3 = []
sim_46_3 = []
```

```
sim_53_3 = []
sim_58_3 = []
sim_64_3 = []
sim_68_3 = []
sim_31_4 = []
sim_39_4 = []
sim_46_4 = []
sim_53_4 = []
sim_58_4 = []
sim_64_4 = []
sim_68_4 = []
sim_31_5 = []
sim_39_5 = []
sim_46_5 = []
sim_53_5 = []
sim_58_5 = []
sim_64_5 = []
sim_68_5 = []
sim_31_6 = []
sim_39_6 = []
sim_46_6 = []
sim_53_6 = []
sim_58_6 = []
sim_64_6 = []
sim_68_6 = []
sim_31_7 = []
sim_39_7 = []
sim_46_7 = []
sim_53_7 = []
sim_58_7 = []
sim_64_7 = []
sim_68_7 = []
res_31_1 = []                #residual between simulated and
experimental spectra
res_39_1 = []
res_46_1 = []
res_53_1 = []
res_58_1 = []
res_64_1 = []
res_68_1 = []
res_31_2 = []
res_39_2 = []
res_46_2 = []
res_53_2 = []
res_58_2 = []
res_64_2 = []
res_68_2 = []
res_31_3 = []
res_39_3 = []
res_46_3 = []
res_53_3 = []
res_58_3 = []
res_64_3 = []
res_68_3 = []
res_31_4 = []
res_39_4 = []
res_46_4 = []
res_53_4 = []
```

```
res_58_4 = []
res_64_4 = []
res_68_4 = []
res_31_5 = []
res_39_5 = []
res_46_5 = []
res_53_5 = []
res_58_5 = []
res_64_5 = []
res_68_5 = []
res_31_6 = []
res_39_6 = []
res_46_6 = []
res_53_6 = []
res_58_6 = []
res_64_6 = []
res_68_6 = []
res_31_7 = []
res_39_7 = []
res_46_7 = []
res_53_7 = []
res_58_7 = []
res_64_7 = []
res_68_7 = []


#Reads in all the data from the formatted input file, must contain
wavelength,
#graphite n and k (Djurisic and Li, 1999, DOI 10.1063/1.369370), exptl
data for
#each angle at each thickness and each thickness value.
#HEADERS CAN BE CHANGED AS REQUIRED
exp_data = pd.read_csv('N:\Documents\Python\Input
files\EXAMPLE_FILE_NAME.csv')          #CHANGE DIRECTORY AS NEEDED
exp_lambda1 = exp_data['lambda1']                              #grabs data
from column with given header (header excluded)
exp_n21s = exp_data['n21s']
exp_k21s = exp_data['k21s']
exp_n21p = exp_data['n21p']
exp_k21p = exp_data['k21p']
exp_31_1 = exp_data['31_1']
exp_39_1 = exp_data['39_1']
exp_46_1 = exp_data['46_1']
exp_53_1 = exp_data['53_1']
exp_58_1 = exp_data['58_1']
exp_64_1 = exp_data['64_1']
exp_68_1 = exp_data['68_1']
exp_31_2 = exp_data['31_2']
exp_39_2 = exp_data['39_2']
exp_46_2 = exp_data['46_2']
exp_53_2 = exp_data['53_2']
exp_58_2 = exp_data['58_2']
exp_64_2 = exp_data['64_2']
exp_68_2 = exp_data['68_2']
exp_31_3 = exp_data['31_3']
exp_39_3 = exp_data['39_3']
exp_46_3 = exp_data['46_3']
exp_53_3 = exp_data['53_3']
```

```
exp_58_3 = exp_data['58_3']
exp_64_3 = exp_data['64_3']
exp_68_3 = exp_data['68_3']
exp_31_4 = exp_data['31_4']
exp_39_4 = exp_data['39_4']
exp_46_4 = exp_data['46_4']
exp_53_4 = exp_data['53_4']
exp_58_4 = exp_data['58_4']
exp_64_4 = exp_data['64_4']
exp_68_4 = exp_data['68_4']
exp_31_5 = exp_data['31_5']
exp_39_5 = exp_data['39_5']
exp_46_5 = exp_data['46_5']
exp_53_5 = exp_data['53_5']
exp_58_5 = exp_data['58_5']
exp_64_5 = exp_data['64_5']
exp_68_5 = exp_data['68_5']
exp_31_6 = exp_data['31_6']
exp_39_6 = exp_data['39_6']
exp_46_6 = exp_data['46_6']
exp_53_6 = exp_data['53_6']
exp_58_6 = exp_data['58_6']
exp_64_6 = exp_data['64_6']
exp_68_6 = exp_data['68_6']
exp_31_7 = exp_data['31_7']
exp_39_7 = exp_data['39_7']
exp_46_7 = exp_data['46_7']
exp_53_7 = exp_data['53_7']
exp_58_7 = exp_data['58_7']
exp_64_7 = exp_data['64_7']
exp_68_7 = exp_data['68_7']
d_1 = exp_data.at[0, 'd_1']
d_2 = exp_data.at[0, 'd_2']
d_3 = exp_data.at[0, 'd_3']
d_4 = exp_data.at[0, 'd_4']
d_5 = exp_data.at[0, 'd_5']
d_6 = exp_data.at[0, 'd_6']
d_7 = exp_data.at[0, 'd_7']



#Calculates sum of difference squared across all angles for a given
wavelength
#CHANGE ANGLES TO MATCH ABOVE LIST
def DR_R_func_1(j, d, lambda1, n21s, k21s, n21p, k21p, n11, k11):
    sum_diff_sqrd = 0
    n0 = 1
    N11 = complex(n11, k11)
    N21s = complex(n21s, k21s)
    N21p = complex(n21p, k21p)
    #print(N21s)

    for i in angle:
        theta0 = i
        if theta0 == (np.deg2rad(31)):
            if d == d_1:
                exp_value = exp_31_1[j]
```

```
    elif d == d_2:
        exp_value = exp_31_2[j]
    elif d == d_3:
        exp_value = exp_31_3[j]
    elif d == d_4:
        exp_value = exp_31_4[j]
    elif d == d_5:
        exp_value = exp_31_5[j]
    elif d == d_6:
        exp_value = exp_31_6[j]
    elif d == d_7:
        exp_value = exp_31_7[j]
elif theta0 == (np.deg2rad(39)):
    if d == d_1:
        exp_value = exp_39_1[j]
    elif d == d_2:
        exp_value = exp_39_2[j]
    elif d == d_3:
        exp_value = exp_39_3[j]
    elif d == d_4:
        exp_value = exp_39_4[j]
    elif d == d_5:
        exp_value = exp_39_5[j]
    elif d == d_6:
        exp_value = exp_39_6[j]
    elif d == d_7:
        exp_value = exp_39_7[j]
elif theta0 == (np.deg2rad(46)):
    if d == d_1:
        exp_value = exp_46_1[j]
    elif d == d_2:
        exp_value = exp_46_2[j]
    elif d == d_3:
        exp_value = exp_46_3[j]
    elif d == d_4:
        exp_value = exp_46_4[j]
    elif d == d_5:
        exp_value = exp_46_5[j]
    elif d == d_6:
        exp_value = exp_46_6[j]
    elif d == d_7:
        exp_value = exp_46_7[j]
elif theta0 == (np.deg2rad(53)):
    if d == d_1:
        exp_value = exp_53_1[j]
    elif d == d_2:
        exp_value = exp_53_2[j]
    elif d == d_3:
        exp_value = exp_53_3[j]
    elif d == d_4:
        exp_value = exp_53_4[j]
    elif d == d_5:
        exp_value = exp_53_5[j]
    elif d == d_6:
        exp_value = exp_53_6[j]
    elif d == d_7:
        exp_value = exp_53_7[j]
elif theta0 == (np.deg2rad(58)):
```

```
            if d == d_1:
                exp_value = exp_58_1[j]
            elif d == d_2:
                exp_value = exp_58_2[j]
            elif d == d_3:
                exp_value = exp_58_3[j]
            elif d == d_4:
                exp_value = exp_58_4[j]
            elif d == d_5:
                exp_value = exp_58_5[j]
            elif d == d_6:
                exp_value = exp_58_6[j]
            elif d == d_7:
                exp_value = exp_58_7[j]
        elif theta0 == (np.deg2rad(64)):
            if d == d_1:
                exp_value = exp_64_1[j]
            elif d == d_2:
                exp_value = exp_64_2[j]
            elif d == d_3:
                exp_value = exp_64_3[j]
            elif d == d_4:
                exp_value = exp_64_4[j]
            elif d == d_5:
                exp_value = exp_64_5[j]
            elif d == d_6:
                exp_value = exp_64_6[j]
            elif d == d_7:
                exp_value = exp_64_7[j]
        elif theta0 == (np.deg2rad(68)):
            if d == d_1:
                exp_value = exp_68_1[j]
            elif d == d_2:
                exp_value = exp_68_2[j]
            elif d == d_3:
                exp_value = exp_68_3[j]
            elif d == d_4:
                exp_value = exp_68_4[j]
            elif d == d_5:
                exp_value = exp_68_5[j]
            elif d == d_6:
                exp_value = exp_68_6[j]
            elif d == d_7:
                exp_value = exp_68_7[j]

        costheta1 = np.sqrt((1 - ((n0 ** 2 * ((np.sin(theta0))**2)) /
N11)))
        costheta2s = np.sqrt((1 - ((n0 ** 2 * ((np.sin(theta0))**2)) /
N21s)))
        costheta2p = np.sqrt((1 - ((n0 ** 2 * ((np.sin(theta0)) ** 2))
/ N21p)))
        deltaa = (2 * np.pi * d * N11 * costheta1) / lambda1

        r1p = ((N11 * np.cos(theta0)) - (n0 * costheta1)) / ((N11 *
np.cos(theta0)) + (n0 * costheta1))
        r1s = ((n0 * np.cos(theta0)) - (N11 * costheta1)) / ((n0 *
np.cos(theta0)) + (N11 * costheta1))
```

```
        r2p = ((N21p * costheta1) - (N11 * costheta2p)) / ((N21p *
costheta1) + (N11 * costheta2p))
        r2s = ((N11 * costheta1) - (N21s * costheta2s)) / ((N11 *
costheta1) + (N21s * costheta2s))
        r02p = ((N21p * np.cos(theta0)) - (n0 * costheta2p)) / ((N21p *
np.cos(theta0)) + (n0 * costheta2p))
        r02s = ((n0 * costheta1) - (N21s * costheta2s)) / ((N21s *
np.cos(theta0)) + (n0 * costheta2s))

        RP = abs((r1p + (r2p * np.exp(-2j*deltaa))) / (1 + (r1p * r2p *
np.exp(-2j*deltaa))))**2
        RS = abs((r1s + (r2s * np.exp(-2j*deltaa))) / (1 + (r1s * r2s *
np.exp(-2j*deltaa))))**2
        R0P = (abs(r02p))**2
        R0S = (abs(r02s))**2

        R = RP + RS
        R0 = R0P + R0S
        deltaR_over_R = (R-R0)/R0

        diff = abs(exp_value - deltaR_over_R)
        sum_diff_sqrd += (diff**2)
    return sum_diff_sqrd


#Gives simulated spectra using optimised n and k values
def DR_R_func_2(theta0, d, lambda1, n21s, k21s, n21p, k21p, n11, k11):
    n0 = 1
    N11 = complex(n11, k11)
    N21s = complex(n21s, k21s)
    N21p = complex(n21p, k21p)

    costheta1 = np.sqrt((1 - ((n0 ** 2 * ((np.sin(theta0))**2)) /
N11)))
    costheta2s = np.sqrt((1 - ((n0 ** 2 * ((np.sin(theta0))**2)) /
N21s)))
    costheta2p = np.sqrt((1 - ((n0 ** 2 * ((np.sin(theta0)) ** 2)) /
N21p)))
    deltaa = (2 * np.pi * d * N11 * costheta1) / lambda1

    r1p = ((N11 * np.cos(theta0)) - (n0 * costheta1)) / ((N11 *
np.cos(theta0)) + (n0 * costheta1))
    r1s = ((n0 * np.cos(theta0)) - (N11 * costheta1)) / ((n0 *
np.cos(theta0)) + (N11 * costheta1))
    r2p = ((N21p * costheta1) - (N11 * costheta2p)) / ((N21p *
costheta1) + (N11 * costheta2p))
    r2s = ((N11 * costheta1) - (N21s * costheta2s)) / ((N11 *
costheta1) + (N21s * costheta2s))
    r02p = ((N21p * np.cos(theta0)) - (n0 * costheta2p)) / ((N21p *
np.cos(theta0)) + (n0 * costheta2p))
    r02s = ((n0 * costheta1) - (N21s * costheta2s)) / ((N21s *
np.cos(theta0)) + (n0 * costheta2s))

    RP = abs((r1p + (r2p * np.exp(-2j*deltaa))) / (1 + (r1p * r2p *
np.exp(-2j*deltaa))))**2
    RS = abs((r1s + (r2s * np.exp(-2j*deltaa))) / (1 + (r1s * r2s *
np.exp(-2j*deltaa))))**2
    R0P = (abs(r02p))**2
```

```
    R0S = (abs(r02s))**2

    R = RP + RS
    R0 = R0P + R0S
    deltaR_over_R = (R-R0)/R0

    return deltaR_over_R


#pulls n and k as separate variables
def func_wrap(x):
    n11, k11 = x
    fx = ((DR_R_func_1(j, d_1, exp_lambda1[j], exp_n21s[j],
exp_k21s[j], exp_n21p[j], exp_k21p[j], n11, k11))
         +(DR_R_func_1(j, d_2, exp_lambda1[j], exp_n21s[j],
exp_k21s[j], exp_n21p[j], exp_k21p[j], n11, k11))
         +(DR_R_func_1(j, d_3, exp_lambda1[j], exp_n21s[j],
exp_k21s[j], exp_n21p[j], exp_k21p[j], n11, k11))
         +(DR_R_func_1(j, d_4, exp_lambda1[j], exp_n21s[j],
exp_k21s[j], exp_n21p[j], exp_k21p[j], n11, k11))
         +(DR_R_func_1(j, d_5, exp_lambda1[j], exp_n21s[j],
exp_k21s[j], exp_n21p[j], exp_k21p[j], n11, k11)))
         +(DR_R_func_1(j, d_6, exp_lambda1[j], exp_n21s[j],
exp_k21s[j], exp_n21p[j], exp_k21p[j], n11, k11))
         +(DR_R_func_1(j, d_7, exp_lambda1[j], exp_n21s[j],
exp_k21s[j], exp_n21p[j], exp_k21p[j], n11, k11)))
    return fx




#Allows iteration with float step size rather than integer #Written by
Dr Adam Baskerville
def seq(start, stop, step=1):
    n = int(round((stop - start) / float(step)))
    if n > 1:
        return ([start + step * i for i in range(n + 1)])
    else:
        return ([])


#Finds the starting guess for n and k by brute force approach for first
wavelength value
#iterates all values of n from 0.5 - 3 at a step size of 0.2 and all
values of k
#between 0 and 1 at a step size of 0.001
s = 20
for j in range(0, 1):
    #print(exp_lambda1[j])
    for n11 in seq(0.5, 3, 0.2):
        for k11 in seq(0.000, 1, 0.001):
            sol_1 = DR_R_func_1(j, d_1, exp_lambda1[j], exp_n21s[j],
exp_k21s[j], exp_n21p[j], exp_k21p[j], n11, k11)
            sol_2 = DR_R_func_1(j, d_2, exp_lambda1[j], exp_n21s[j],
exp_k21s[j], exp_n21p[j], exp_k21p[j], n11, k11)
            sol_3 = DR_R_func_1(j, d_3, exp_lambda1[j], exp_n21s[j],
exp_k21s[j], exp_n21p[j], exp_k21p[j], n11, k11)
```

```
            sol_4 = DR_R_func_1(j, d_4, exp_lambda1[j], exp_n21s[j],
exp_k21s[j], exp_n21p[j], exp_k21p[j], n11, k11)
            sol_5 = DR_R_func_1(j, d_5, exp_lambda1[j], exp_n21s[j],
exp_k21s[j], exp_n21p[j], exp_k21p[j], n11, k11)
            sol_6 = DR_R_func_1(j, d_6, exp_lambda1[j], exp_n21s[j],
exp_k21s[j], exp_n21p[j], exp_k21p[j], n11, k11)
            sol_7 = DR_R_func_1(j, d_7, exp_lambda1[j], exp_n21s[j],
exp_k21s[j], exp_n21p[j], exp_k21p[j], n11, k11)
            sol_sum = sol_1 + sol_2 + sol_3 + sol_4 + sol_5  + sol_6 +
sol_7
            if sol_sum < s:
                s = sol_sum
                n11_guess = n11
                k11_guess = k11



#Seeding and optimisation
#Takes value of n and k for previous wavelength as starting guess for
current wavelength
#Uses least squares to optimise n and k values
for j in range(0,len(exp_lambda1)):
    if j == 0:
        res_wrapped = least_squares(func_wrap, (n11_guess, k11_guess),
bounds=([0, 0], [5, 5]))     #optimisation using guess
        z = res_wrapped.x[0] + res_wrapped.x[1] * 1j
        ntarget.append(res_wrapped.x[0])
        ktarget.append(res_wrapped.x[1])
    elif j != 0:
        res_wrapped = least_squares(func_wrap, (ntarget[j-1],
ktarget[j-1]), bounds=([0, 0], [5, 5]))  #optimisation using previous
wavelength value
        z = res_wrapped.x[0] + res_wrapped.x[1] * 1j
        ntarget.append(res_wrapped.x[0])
        ktarget.append(res_wrapped.x[1])


#Calls function to give simulated spectra for each angle and calculates
residuals
for i in angle:
    theta0 = i
    for x in range(0, len(exp_lambda1)):
        if theta0 == np.deg2rad(31):
            a = DR_R_func_2(i, d_1, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_31_1.append(a)
            residual_a = (a - exp_31_1[x])**2
            res_31_1.append(residual_a)
            b = DR_R_func_2(i, d_2, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_31_2.append(b)
            residual_b = (b - exp_31_2[x])**2
            res_31_2.append(residual_b)
            c = DR_R_func_2(i, d_3, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_31_3.append(c)
            residual_c = (c - exp_31_3[x])**2
            res_31_3.append(residual_c)
```

```
            dd = DR_R_func_2(i, d_4, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_31_4.append(dd)
            residual_d = (dd - exp_31_4[x]) ** 2
            res_31_4.append(residual_d)
            ee = DR_R_func_2(i, d_5, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_31_5.append(ee)
            residual_e = (ee - exp_31_5[x]) ** 2
            res_31_5.append(residual_e)
            f = DR_R_func_2(i, d_6, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_31_6.append(f)
            residual_f = (f - exp_31_6[x]) ** 2
            res_31_6.append(residual_f)
            g = DR_R_func_2(i, d_7, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_31_7.append(g)
            residual_g = (g - exp_31_7[x]) ** 2
            res_31_7.append(residual_g)
        elif theta0 == np.deg2rad(39):
            a = DR_R_func_2(i, d_1, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_39_1.append(a)
            residual_a = (a - exp_39_1[x])**2
            res_39_1.append(residual_a)
            b = DR_R_func_2(i, d_2, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_39_2.append(b)
            residual_b = (b - exp_39_2[x])**2
            res_39_2.append(residual_b)
            c = DR_R_func_2(i, d_3, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_39_3.append(c)
            residual_c = (c - exp_39_3[x])**2
            res_39_3.append(residual_c)
            dd = DR_R_func_2(i, d_4, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_39_4.append(dd)
            residual_d = (dd - exp_39_4[x]) ** 2
            res_39_4.append(residual_d)
            ee = DR_R_func_2(i, d_5, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_39_5.append(ee)
            residual_e = (ee - exp_39_5[x]) ** 2
            res_39_5.append(residual_e)
            f = DR_R_func_2(i, d_6, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_39_6.append(f)
            residual_f = (f - exp_39_6[x]) ** 2
            res_39_6.append(residual_f)
            g = DR_R_func_2(i, d_7, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_39_7.append(g)
            residual_g = (g - exp_39_7[x]) ** 2
            res_39_7.append(residual_g)
        elif theta0 == np.deg2rad(46):
```

```
            a = DR_R_func_2(i, d_1, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_46_1.append(a)
            residual_a = (a - exp_46_1[x])**2
            res_46_1.append(residual_a)
            b = DR_R_func_2(i, d_2, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_46_2.append(b)
            residual_b = (b - exp_46_2[x])**2
            res_46_2.append(residual_b)
            c = DR_R_func_2(i, d_3, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_46_3.append(c)
            residual_c = (c - exp_46_3[x])**2
            res_46_3.append(residual_c)
            dd = DR_R_func_2(i, d_4, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_46_4.append(dd)
            residual_d = (dd - exp_46_4[x]) ** 2
            res_46_4.append(residual_d)
            ee = DR_R_func_2(i, d_5, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_46_5.append(ee)
            residual_e = (ee - exp_46_5[x]) ** 2
            res_46_5.append(residual_e)
            f = DR_R_func_2(i, d_6, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_46_6.append(f)
            residual_f = (f - exp_46_6[x]) ** 2
            res_46_6.append(residual_f)
            g = DR_R_func_2(i, d_7, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_46_7.append(g)
            residual_g = (g - exp_46_7[x]) ** 2
            res_46_7.append(residual_g)
        elif theta0 == np.deg2rad(53):
            a = DR_R_func_2(i, d_1, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_53_1.append(a)
            residual_a = (a - exp_53_1[x])**2
            res_53_1.append(residual_a)
            b = DR_R_func_2(i, d_2, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_53_2.append(b)
            residual_b = (b - exp_53_2[x])**2
            res_53_2.append(residual_b)
            c = DR_R_func_2(i, d_3, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_53_3.append(c)
            residual_c = (c - exp_53_3[x])**2
            res_53_3.append(residual_c)
            dd = DR_R_func_2(i, d_4, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_53_4.append(dd)
            residual_d = (dd - exp_53_4[x]) ** 2
            res_53_4.append(residual_d)
            ee = DR_R_func_2(i, d_5, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
```

```
            sim_53_5.append(ee)
            residual_e = (ee - exp_53_5[x]) ** 2
            res_53_5.append(residual_e)
            f = DR_R_func_2(i, d_6, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_53_6.append(f)
            residual_f = (f - exp_53_6[x]) ** 2
            res_53_6.append(residual_f)
            g = DR_R_func_2(i, d_7, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_53_7.append(g)
            residual_g = (g - exp_53_7[x]) ** 2
            res_53_7.append(residual_g)
        elif theta0 == np.deg2rad(58):
            a = DR_R_func_2(i, d_1, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_58_1.append(a)
            residual_a = (a - exp_58_1[x])**2
            res_58_1.append(residual_a)
            b = DR_R_func_2(i, d_2, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_58_2.append(b)
            residual_b = (b - exp_58_2[x])**2
            res_58_2.append(residual_b)
            c = DR_R_func_2(i, d_3, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_58_3.append(c)
            residual_c = (c - exp_58_3[x])**2
            res_58_3.append(residual_c)
            dd = DR_R_func_2(i, d_4, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_58_4.append(dd)
            residual_d = (dd - exp_58_4[x]) ** 2
            res_58_4.append(residual_d)
            ee = DR_R_func_2(i, d_5, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_58_5.append(ee)
            residual_e = (ee - exp_58_5[x]) ** 2
            res_58_5.append(residual_e)
            f = DR_R_func_2(i, d_6, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_58_6.append(f)
            residual_f = (f - exp_58_6[x]) ** 2
            res_58_6.append(residual_f)
            g = DR_R_func_2(i, d_7, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_58_7.append(g)
            residual_g = (g - exp_58_7[x]) ** 2
            res_58_7.append(residual_g)
        elif theta0 == np.deg2rad(64):
            a = DR_R_func_2(i, d_1, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_64_1.append(a)
            residual_a = (a - exp_64_1[x])**2
            res_64_1.append(residual_a)
            b = DR_R_func_2(i, d_2, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_64_2.append(b)
```

```
            residual_b = (b - exp_64_2[x])**2
            res_64_2.append(residual_b)
            c = DR_R_func_2(i, d_3, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_64_3.append(c)
            residual_c = (c - exp_64_3[x])**2
            res_64_3.append(residual_c)
            dd = DR_R_func_2(i, d_4, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_64_4.append(dd)
            residual_d = (dd - exp_64_4[x]) ** 2
            res_64_4.append(residual_d)
            ee = DR_R_func_2(i, d_5, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_64_5.append(ee)
            residual_e = (ee - exp_64_5[x]) ** 2
            res_64_5.append(residual_e)
            f = DR_R_func_2(i, d_6, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_64_6.append(f)
            residual_f = (f - exp_64_6[x]) ** 2
            res_64_6.append(residual_f)
            g = DR_R_func_2(i, d_7, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_64_7.append(g)
            residual_g = (g - exp_64_7[x]) ** 2
            res_64_7.append(residual_g)
        elif theta0 == np.deg2rad(68):
            a = DR_R_func_2(i, d_1, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_68_1.append(a)
            residual_a = (a - exp_68_1[x])**2
            res_68_1.append(residual_a)
            b = DR_R_func_2(i, d_2, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_68_2.append(b)
            residual_b = (b - exp_68_2[x])**2
            res_68_2.append(residual_b)
            c = DR_R_func_2(i, d_3, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_68_3.append(c)
            residual_c = (c - exp_68_3[x])**2
            res_68_3.append(residual_c)
            dd = DR_R_func_2(i, d_4, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_68_4.append(dd)
            residual_d = (dd - exp_68_4[x]) ** 2
            res_68_4.append(residual_d)
            ee = DR_R_func_2(i, d_5, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_68_5.append(ee)
            residual_e = (ee - exp_68_5[x]) ** 2
            res_68_5.append(residual_e)
            f = DR_R_func_2(i, d_6, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_68_6.append(f)
            residual_f = (f - exp_68_6[x]) ** 2
            res_68_6.append(residual_f)
```

```
            g = DR_R_func_2(i, d_7, exp_lambda1[x], exp_n21s[x],
exp_k21s[x], exp_n21p[x], exp_k21p[x], ntarget[x], ktarget[x])
            sim_68_7.append(g)
            residual_g = (g - exp_68_7[x]) ** 2
            res_68_7.append(residual_g)
        else:
            break



#Puts all output info into a dataframe for a file
table = pd.DataFrame({'lambda':exp_lambda1,
                      'n':ntarget,
                      'k':ktarget,
                      'sim_31_1': sim_31_1,
                      'sim_39_1': sim_39_1,
                      'sim_46_1': sim_46_1,
                      'sim_53_1': sim_53_1,
                      'sim_58_1': sim_58_1,
                      'sim_64_1': sim_64_1,
                      'sim_68_1': sim_68_1,
                      'res_31_1': res_31_1,
                      'res_39_1': res_39_1,
                      'res_46_1': res_46_1,
                      'res_53_1': res_53_1,
                      'res_58_1': res_58_1,
                      'res_64_1': res_64_1,
                      'res_68_1': res_68_1,
                      'sim_31_2': sim_31_2,
                      'sim_39_2': sim_39_2,
                      'sim_46_2': sim_46_2,
                      'sim_53_2': sim_53_2,
                      'sim_58_2': sim_58_2,
                      'sim_64_2': sim_64_2,
                      'sim_68_2': sim_68_2,
                      'res_31_2': res_31_2,
                      'res_39_2': res_39_2,
                      'res_46_2': res_46_2,
                      'res_53_2': res_53_2,
                      'res_58_2': res_58_2,
                      'res_64_2': res_64_2,
                      'res_68_2': res_68_2,
                      'sim_31_3': sim_31_3,
                      'sim_39_3': sim_39_3,
                      'sim_46_3': sim_46_3,
                      'sim_53_3': sim_53_3,
                      'sim_58_3': sim_58_3,
                      'sim_64_3': sim_64_3,
                      'sim_68_3': sim_68_3,
                      'res_31_3': res_31_3,
                      'res_39_3': res_39_3,
                      'res_46_3': res_46_3,
                      'res_53_3': res_53_3,
                      'res_58_3': res_58_3,
                      'res_64_3': res_64_3,
                      'res_68_3': res_68_3,
                      'sim_31_4': sim_31_4,
                      'sim_39_4': sim_39_4,
```

```
                        'sim_46_4': sim_46_4,
                        'sim_53_4': sim_53_4,
                        'sim_58_4': sim_58_4,
                        'sim_64_4': sim_64_4,
                        'sim_68_4': sim_68_4,
                        'res_31_4': res_31_4,
                        'res_39_4': res_39_4,
                        'res_46_4': res_46_4,
                        'res_53_4': res_53_4,
                        'res_58_4': res_58_4,
                        'res_64_4': res_64_4,
                        'res_68_4': res_68_4,
                        'sim_31_5': sim_31_5,
                        'sim_39_5': sim_39_5,
                        'sim_46_5': sim_46_5,
                        'sim_53_5': sim_53_5,
                        'sim_58_5': sim_58_5,
                        'sim_64_5': sim_64_5,
                        'sim_68_5': sim_68_5,
                        'res_31_5': res_31_5,
                        'res_39_5': res_39_5,
                        'res_46_5': res_46_5,
                        'res_53_5': res_53_5,
                        'res_58_5': res_58_5,
                        'res_64_5': res_64_5,
                        'res_68_5': res_68_5,
                        'sim_31_6': sim_31_6,
                        'sim_39_6': sim_39_6,
                        'sim_46_6': sim_46_6,
                        'sim_53_6': sim_53_6,
                        'sim_58_6': sim_58_6,
                        'sim_64_6': sim_64_6,
                        'sim_68_6': sim_68_6,
                        'res_31_6': res_31_6,
                        'res_39_6': res_39_6,
                        'res_46_6': res_46_6,
                        'res_53_6': res_53_6,
                        'res_58_6': res_58_6,
                        'res_64_6': res_64_6,
                        'res_68_6': res_68_6,
                        'sim_31_7': sim_31_7,
                        'sim_39_7': sim_39_7,
                        'sim_46_7': sim_46_7,
                        'sim_53_7': sim_53_7,
                        'sim_58_7': sim_58_7,
                        'sim_64_7': sim_64_7,
                        'sim_68_7': sim_68_7,
                        'res_31_7': res_31_7,
                        'res_39_7': res_39_7,
                        'res_46_7': res_46_7,
                        'res_53_7': res_53_7,
                        'res_58_7': res_58_7,
                        'res_64_7': res_64_7,
                        'res_68_7': res_68_7,
                        })

#Writes all outputs to a single .txt file
```

```
ffile = open('EXAMPLE_OUTPUT_FILE.txt', 'w+')       #RENAME AS NEEDED
ffile.write(str(table))
ffile.close()

#Gives code run time in seconds
end = time.time()
print("Calculation took {} s".format((end - start)))


print("\nComplex RI calculator has run. Program ended.")
```

Appendix E2. *Python code used to simulate a UV-visible spectrum from known n and k values.*

```
#Code to simulate a UV spectrum from the n and k values for an ice
#Written by James Stubbing in Python 3.7 July 2018


#Import relvant modules
import numpy as np              #numpy module for mathematical
operations
import pandas as pd             #pandas module for reading files
import time                     #allows the time the program takes to
run to be recorded

start = time.time()                     #Starts counter to see how long
code takes to run

pd.set_option('display.max_rows', 2000)         #Set panda to show all
the data
pd.set_option('display.max_columns', 2000)

#List of reflection angles
#EDIT AS REQUIRED
angle = [np.deg2rad(31), np.deg2rad(39), np.deg2rad(46),
np.deg2rad(53), np.deg2rad(58), np.deg2rad(64), np.deg2rad(68)]


#Reads in all relevant constant data
#wavelength and graphite optical parameters (Djurisic and Li, 1999, DOI
10.1063/1.369370)
#Must be stored in a .csv file
graphite_data =
pd.read_csv('N:\Documents\Python\EXAMPLE_GRAPHITE_PARAMETERS_FILE.csv')
lambda1 = graphite_data['lambda1']      #wavelength
n21s = graphite_data['n21s']            #n for graphite s polarised
k21s = graphite_data['k21s']
n21p = graphite_data['n21p']
k21p = graphite_data['k21p']


#Set thicknesses required
#EDIT AS REQUIRED
d_100 = 100
d_200 = 200
d_250 = 250
d_400 = 400
d_500 = 500


#Read in known ice n and k values
#from literature or set by user
#EDIT AS REQUIRED
adsorbate =
pd.read_csv('N:\Documents\Python\EXAMPLE_ICE_NANDK_FILE.csv')
n11 = adsorbate['n11']
k11 = adsorbate['k11']


#Produces spectra using all input data
```

```
def DR_R_func(theta0, d, lambda1, n21s, k21s, n21p, k21p, n11, k11):
    n0 = 1
    N11 = complex(n11, k11)
    N21s = complex(n21s, k21s)
    N21p = complex(n21p, k21p)


    costheta1 = np.sqrt((1 - ((n0 ** 2 * ((np.sin(theta0))**2)) /
N11)))
    costheta2s = np.sqrt((1 - ((n0 ** 2 * ((np.sin(theta0))**2)) /
N21s)))
    costheta2p = np.sqrt((1 - ((n0 ** 2 * ((np.sin(theta0)) ** 2)) /
N21p)))
    deltaa = (2 * np.pi * d * N11 * costheta1) / lambda1

    r1p = ((N11 * np.cos(theta0)) - (n0 * costheta1)) / ((N11 *
np.cos(theta0)) + (n0 * costheta1))
    r1s = ((n0 * np.cos(theta0)) - (N11 * costheta1)) / ((n0 *
np.cos(theta0)) + (N11 * costheta1))
    r2p = ((N21p * costheta1) - (N11 * costheta2p)) / ((N21p *
costheta1) + (N11 * costheta2p))
    r2s = ((N11 * costheta1) - (N21s * costheta2s)) / ((N11 *
costheta1) + (N21s * costheta2s))
    r02p = ((N21p * np.cos(theta0)) - (n0 * costheta2p)) / ((N21p *
np.cos(theta0)) + (n0 * costheta2p))
    r02s = ((n0 * costheta1) - (N21s * costheta2s)) / ((N21s *
np.cos(theta0)) + (n0 * costheta2s))

    RP = abs((r1p + (r2p * np.exp(-2j*deltaa))) / (1 + (r1p * r2p *
np.exp(-2j*deltaa))))**2
    RS = abs((r1s + (r2s * np.exp(-2j*deltaa))) / (1 + (r1s * r2s *
np.exp(-2j*deltaa))))**2
    R0P = (abs(r02p))**2
    R0S = (abs(r02s))**2

    R = RP + RS
    R0 = R0P + R0S
    deltaR_over_R = (R-R0)/R0

    #print(deltaR_over_R)
    return deltaR_over_R

#lists for output values
sim_31_100 = []             #E.G. simulated spectrum for angle of 31
degrees at first thickness
sim_39_100 = []
sim_46_100 = []
sim_53_100 = []
sim_58_100 = []
sim_64_100 = []
sim_68_100 = []
sim_31_200 = []
sim_39_200 = []
sim_46_200 = []
sim_53_200 = []
sim_58_200 = []
sim_64_200 = []
sim_68_200 = []
```

```
sim_31_250 = []
sim_39_250 = []
sim_46_250 = []
sim_53_250 = []
sim_58_250 = []
sim_64_250 = []
sim_68_250 = []
sim_31_400 = []
sim_39_400 = []
sim_46_400 = []
sim_53_400 = []
sim_58_400 = []
sim_64_400 = []
sim_68_400 = []
sim_31_500 = []
sim_39_500 = []
sim_46_500 = []
sim_53_500 = []
sim_58_500 = []
sim_64_500 = []
sim_68_500 = []


#Calls function to produce spectra
for i in angle:
    theta0 = i
    #print(theta0)
    for x in range(0, len(lambda1)):
        a = DR_R_func(i, d_100, lambda1[x], n21s[x], k21s[x], n21p[x],
k21p[x], n11[x], k11[x])
        b = DR_R_func(i, d_200, lambda1[x], n21s[x], k21s[x], n21p[x],
k21p[x], n11[x], k11[x])
        c = DR_R_func(i, d_250, lambda1[x], n21s[x], k21s[x], n21p[x],
k21p[x], n11[x], k11[x])
        e = DR_R_func(i, d_400, lambda1[x], n21s[x], k21s[x], n21p[x],
k21p[x], n11[x], k11[x])
        f = DR_R_func(i, d_500, lambda1[x], n21s[x], k21s[x], n21p[x],
k21p[x], n11[x], k11[x])
        if theta0 == np.deg2rad(33):
            sim_31_100.append(a)
            sim_31_200.append(b)
            sim_31_250.append(c)
            sim_31_400.append(e)
            sim_31_500.append(f)
        elif theta0 == np.deg2rad(37):
            sim_39_100.append(a)
            sim_39_200.append(b)
            sim_39_250.append(c)
            sim_39_400.append(e)
            sim_39_500.append(f)
        elif theta0 == np.deg2rad(48):
            sim_46_100.append(a)
            sim_46_200.append(b)
            sim_46_250.append(c)
            sim_46_400.append(e)
            sim_46_500.append(f)
        elif theta0 == np.deg2rad(51):
            sim_53_100.append(a)
```

```
            sim_53_200.append(b)
            sim_53_250.append(c)
            sim_53_400.append(e)
            sim_53_500.append(f)
        elif theta0 == np.deg2rad(60):
            sim_58_100.append(a)
            sim_58_200.append(b)
            sim_58_250.append(c)
            sim_58_400.append(e)
            sim_58_500.append(f)
        elif theta0 == np.deg2rad(62):
            sim_64_100.append(a)
            sim_64_200.append(b)
            sim_64_250.append(c)
            sim_64_400.append(e)
            sim_64_500.append(f)
        elif theta0 == np.deg2rad(70):
            sim_68_100.append(a)
            sim_68_200.append(b)
            sim_68_250.append(c)
            sim_68_400.append(e)
            sim_68_500.append(f)
        else:
            break


#Puts all output data into dataframe for a file
table = pd.DataFrame({'lambda':lambda1,
                      'sim_33_100': sim_31_100,
                      'sim_37_100': sim_39_100,
                      'sim_48_100': sim_46_100,
                      'sim_51_100': sim_53_100,
                      'sim_60_100': sim_58_100,
                      'sim_62_100': sim_64_100,
                      'sim_70_100': sim_68_100,
                      'sim_33_200': sim_31_200,
                      'sim_37_200': sim_39_200,
                      'sim_48_200': sim_46_200,
                      'sim_51_200': sim_53_200,
                      'sim_60_200': sim_58_200,
                      'sim_62_200': sim_64_200,
                      'sim_70_200': sim_68_200,
                      'sim_33_250': sim_31_250,
                      'sim_37_250': sim_39_250,
                      'sim_48_250': sim_46_250,
                      'sim_51_250': sim_53_250,
                      'sim_60_250': sim_58_250,
                      'sim_62_250': sim_64_250,
                      'sim_70_250': sim_68_250,
                      'sim_33_400': sim_31_400,
                      'sim_37_400': sim_39_400,
                      'sim_48_400': sim_46_400,
                      'sim_51_400': sim_53_400,
                      'sim_60_400': sim_58_400,
                      'sim_62_400': sim_64_400,
                      'sim_70_400': sim_68_400,
                      'sim_33_500': sim_31_500,
                      'sim_37_500': sim_39_500,
```

```
                        'sim_48_500': sim_46_500,
                        'sim_51_500': sim_53_500,
                        'sim_60_500': sim_58_500,
                        'sim_62_500': sim_64_500,
                        'sim_70_500': sim_68_500
                        })


#writes dataframe to a single .txt file
ffile = open('SIMULATIONS_EXAMPLE_FILE_NAME.txt', 'w+')
ffile.write(str(table))
ffile.close

#Shows time taken for program
end = time.time()
print("Calculation took {} s".format((end - start)))
```