Electronic Supplementary Material (ESI) for Soft Matter.
This journal is © The Royal Society of Chemistry 2020

Supplementary Information

COMPETITION OF NANOPARTICLE-INDUCED MOBILIZATION AND IMMOBILIZATION EFFECTS ON SEGMENTAL DYNAMICS OF AN EPOXY-BASED NANOCOMPOSITE

Paulina Szymoniak, Brian Richard Pauw, Xintong Qu, Andreas Schönhals

Bundesanstalt für Materialforschung und –prüfung - (BAM), Unter den Eichen 87, 12205 Berlin (Germany)

TGA measurements

The measurements were performed using a STA7000 Series Thermogravimetric Analyzer (Hitachi, Chidoya, Japan) with a microgram-level weight change detection. Samples of 8-13 mg were heated in alumina pans from 303 K to 1273 K at a heating rate of 10 K min^{-1}. Nitrogen was used as a purge gas up to approximately 873 K. In the temperature range from 873 K to 1273 K oxygen was employed in order to oxidize the material completely. The analyzer is coupled via a heated (473 K) quartz capillary with a ThermoStar GSD 329 mass spectrometer (MS) (Pfeiffer Vacuum, Asslar, Deutschland). The TGA-MS connection enables the investigation of the pyrolysis gas glow, which is transferred to MS.

Figure S1A shows the mass loss versus temperature for the pure epoxy and the PNCs with various BNP contents (EP/BNP1 – EP/BNP15). A minor weight loss of less than 1 wt % was found for all the samples at ca. 423 K, followed by a further decomposition reaction at ca. 523 K, where the mass loss is in the range from 2.5 for EP to 4 wt% for EP/BNP15, respectively. The decomposition products at 423 and 523 K were investigated with mass spectrometry. It was found that the pyrolysis gas at 423 K contained CO_2, which is a typical decomposition product of an un-reacted anhydride. At the higher temperature phenyl groups were detected, resulting from the decomposition of the un-crosslinked DGEBA monomer. The main step in the mass loss is approximately 90 wt% for EP and 70 wt% for EP/BNP15 decreasing with increasing BNPs content. The char yield is proportional to the nanoparticle concentration, as expected (see Figure S1B). The main mass loss corresponds to the degradation of the epoxy network, where the decomposition temperature (T_{dec}) is defined as half step height of the main mass loss. The inset of Figure S1A shows that T_{dec} shifts to lower temperatures with increasing filler content, from approximately 681 K for EP to 661 K for EP/BNP15. This means that the introduction of BNPs causes an earlier

decomposition of the matrix of the composites likely due to a lower crosslinking degree. The last step in the mass loss curve at approximately 900 K for all the samples corresponds to the complete oxidation of the material, because synthetic air used as a purge gas in this temperature range.

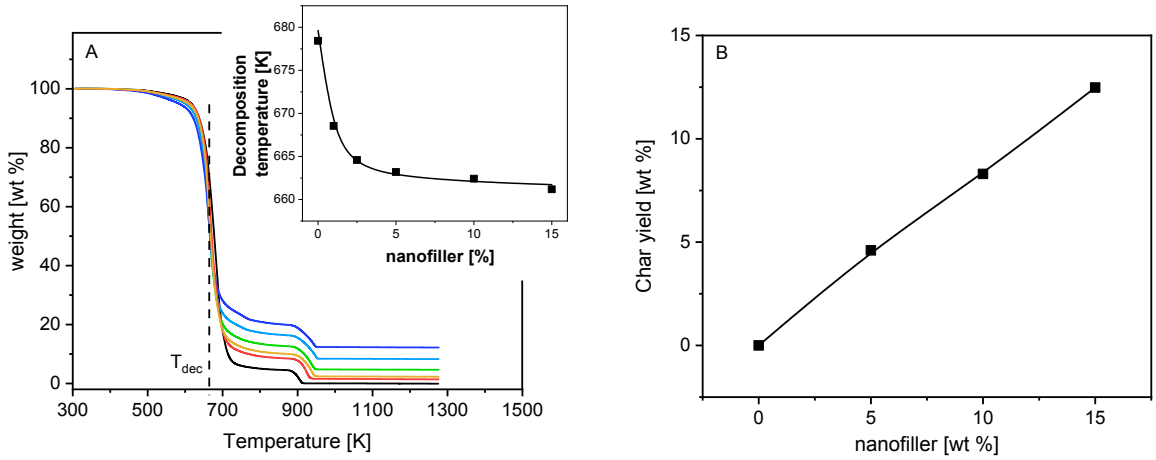

Figure S1: (A) TGA weight loss curves of anhydride-cured unfilled epoxy (EP - black) and nanocomposites filled with boehmite: 1 wt% (EP/BNP1 – red), 2.5 wt% (EP/BNP2.5 – orange), 5 wt% (EP/BNP5 – green), 10 wt% (EP/BNP10 – blue), 15 wt% (EP/BNP15 – dark blue): The inset gives the decomposition temperature as function of nanofiller content. (B) Char yield vs. concentration of the nanofiller.

Analysis of the X-ray Data

Method 1:

For each PNC, the wide-angle signal I_c beyond q=7 nm^{-1} was described by fitting a linear combination of the constituents (i.e. the pure polymer and the BNP signals, I_p and I_b, with scaling factors A_b and A_p):

$$I_c=A_bI_b + A_pI_p. \quad (2)$$

The scaling factors were determined by a least-squares optimization procedure, weighting the intensity by their respective uncertainties.

As this ignores any change in signal due to the interaction between the BNP and the polymer, the resulting volume fraction φ_b should only be considered as a rough estimate for the composition. The volume fraction is determined from the scaling factors for the polymer and BNP:

$$\phi_b = \frac{A_b}{A_b + A_p} \quad (3)$$

The mass fraction w_b in the composite is then estimated via the bulk densities for the nanoparticles and polymer ρ_b and ρ_p, respectively by

$$w_b = \frac{\phi_b\rho_b}{\phi_b\rho_b + (1 - \phi_b)\rho_p} \quad (4)$$

Note that the weight fraction w_b does not scale linearly with the volume fraction φ_b, as they denote fractions of a changing in total. An example of the fit is given in Figure S2.

Despite this being a rather inaccurate method, the resulting mass fractions do not deviate much from the formulated mass fraction, and an acceptable agreement is obtained (see Figure 2A). Of note are two caveats. Firstly, due to the geometry of the instrument, with the wide-angle detector positioned very close to the sample, any deviations in height of the sample due to bulging, thickness

or mounting effects will lead to a shift in peak position, in particular at high q. Secondly, as the wide-angle signal comes from a small beam on a single position on a non-spinning sample, a good powder average is not achieved, and the peak heights are not necessarily internally comparable (see Figure S2).

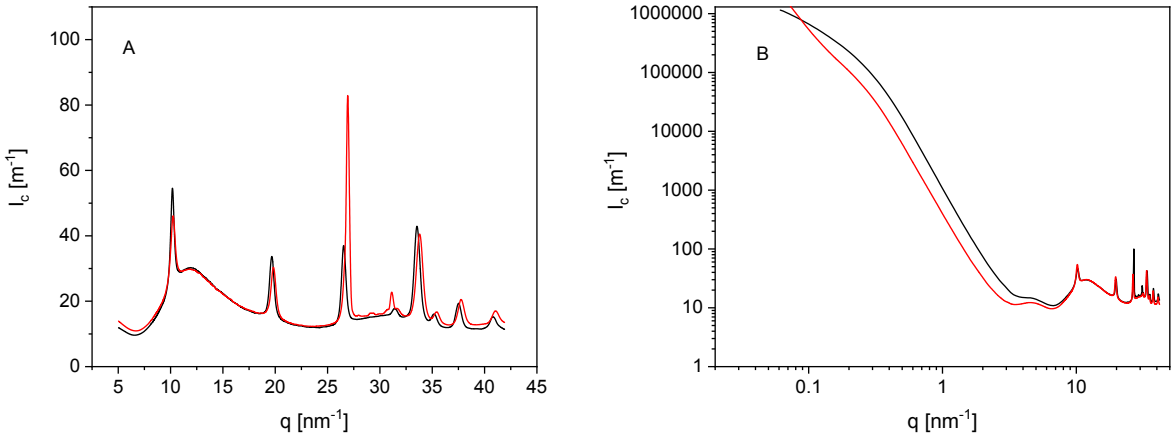

Figure S2: Analysis of X-ray data by fitting a linear combination of both constituents: Black – experimental data, red -fitted function. (A) Zoom-in in the WAXS region. (B) Full available q-range.

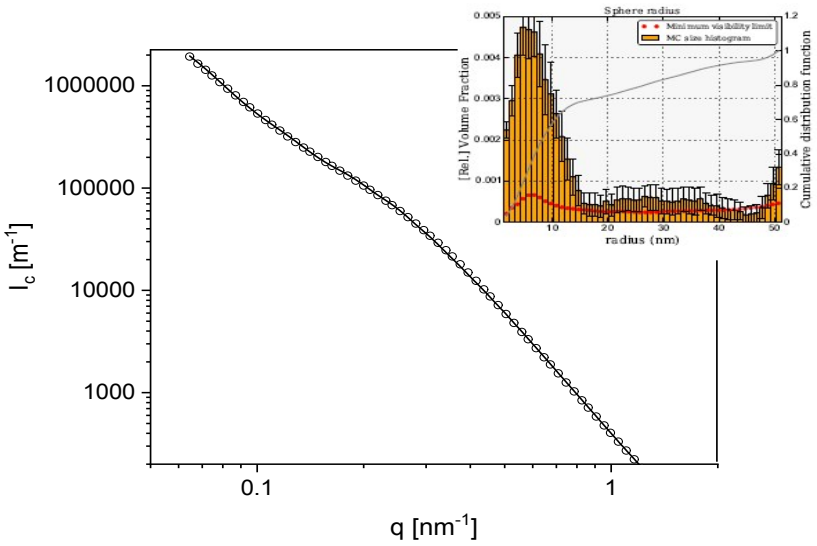

Figure S3: X-ray scattering intensity versus q: circles – experimental data, line – McSAS fit to the experimental data. The inset gives the histogram of the size distribution.

Conventional DSC

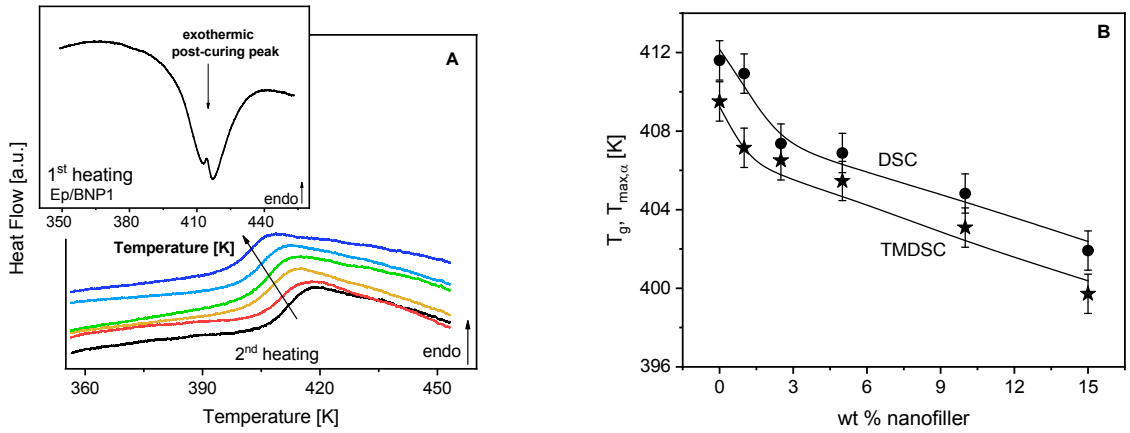

Figure S4: (A) Heat flow curves for the second heating run measured with a heating rate of 10 K min^{-1} for anhydride-cured unfilled epoxy (EP - black) and nanocomposites filled with boehmite: 1 wt% (EP/BNP1 – red), 2.5 wt% (EP/BNP2.5 – orange), 5 wt% (EP/BNP5 – green), 10 wt% (EP/BNP10 – blue), 15 wt% (EP/BNP15 – dark blue). The curves are shifted along Y-scale for clarity. The inset gives the heat flow curve from the first heating run for EP/BNP1. (B) T_g (circles) and $T_{max,\alpha}$ (stars) obtained from DSC at a heating rate of 10 K min^{-1} and TMDSC at frequency of 0.017 Hz vs. concentration of the nanoparticles. Lines are guides to the eyes.

Flash DSC of the unfilled epoxy

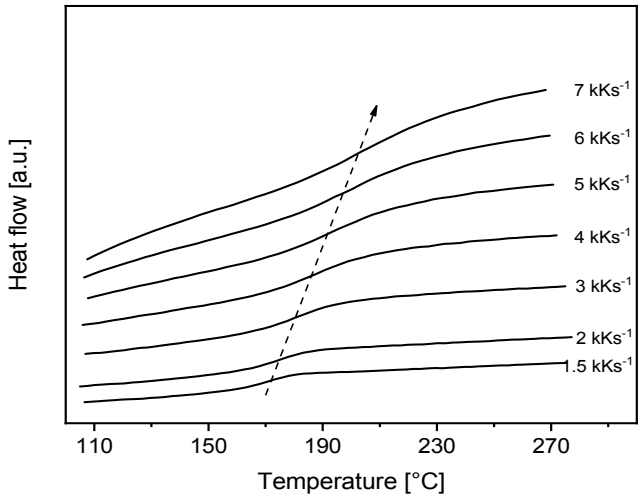

Figure S5: FSC heat flow curves for unfilled epoxy at the indicated heating rates.

Flash DSC of the Nanocomposite

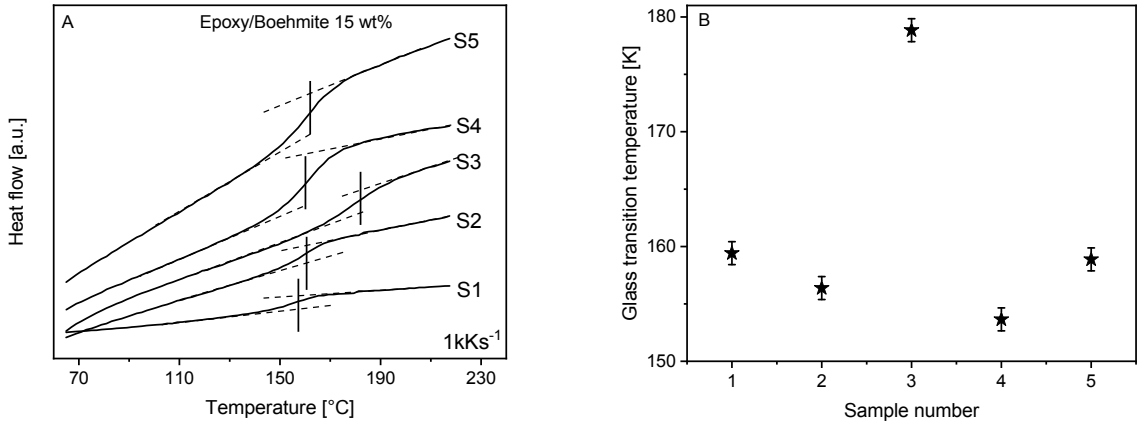

Figure S6:(A) Flash DSC heat flow curves for the epoxy/boehmite nanocomposite with 15 wt% filler content (EP/BNP15) for five consecutive samples (S1-S5) obtained from the same material measured with a heating rate of 1 kK s^{-1}. The heat flow curves were shifted along the y-scale for the sake of clarity. (B) Glass transition temperature values obtained from Flash DSC for the different samples S1-S5.

Examples of the Analysis of the Dielectric Measurements

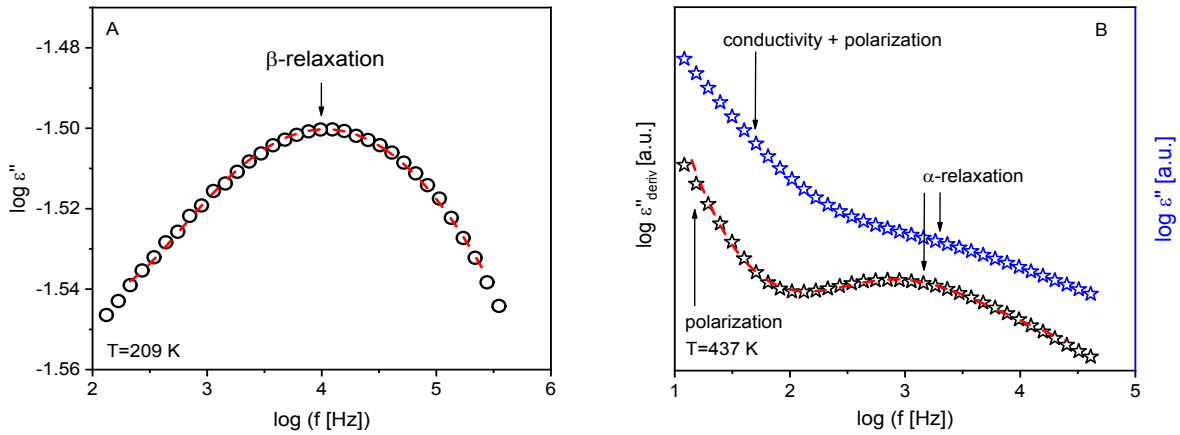

Figure S7: Frequency dependence of (A) ε'' and (B) ε'' and ε_{deriv}'' for pure epoxy for the second heating run at (A) 209 K and (B) 437 K (blue stars - ε''; black stars - ε_{deriv}''). Red dashed lines are fits of the Havriliak-Negami equation (Eqn. 4) and of the derivative of Havriliak-Negami function (Eqn. 7) to the corresponding data, respectively.

Activation energy of the β-relaxation

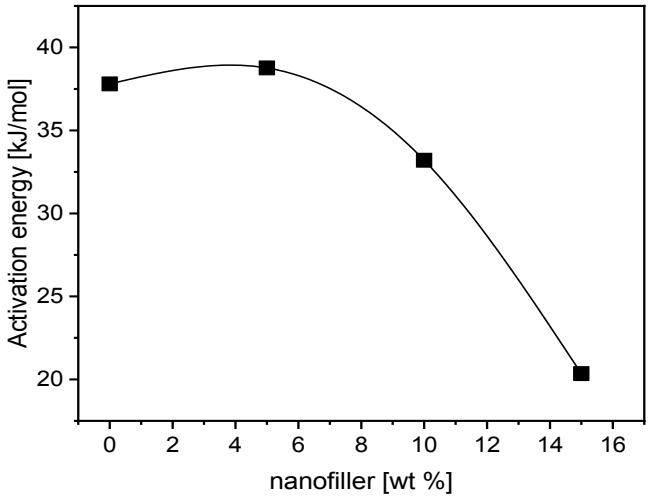

Figure S8: Activation energy of the β-relaxation versus nanoparticle content.

Python Code for the analysis of the X-ray data

[1] # Python 3 initialization:

```
import pandas, glob
import numpy as np
import h5py
import os
from pathlib import Path
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
from IPython.display import display, Math, Latex
matplotlib.style.use('bmh')
# linear regression (and uncertainty estimation) comes from statsmodels:
import statsmodels.api as sm
# for standard scattering models:
# change this one to whereever you have the sasmodels installed:
import sys
sys.path.append('/Users/brian/Code/sasmodels/')
import sasmodels # using commit January 14, 2020, SHA1:␣
,!ea22ff693b7b815e0f7bdb8ec9acac4b517f34e5
import sasmodels.core
import sasmodels.direct_model
# for beam center linear regression:
from scipy.stats import linregress
# for beam profile plots:
from pylab import cm
from matplotlib.colors import LogNorm
# for data merging:
from statsmodels.stats.weightstats import DescrStatsW
from findLinearComination import findLinearCombination
```

[2]

```
# for data reading:
# codify this into something that can read and concatenate multiple nexus␣
,!datasets for datamerge:
def readList(filenamelist,
eMin = 0.01, outputRanges = False):
"""
Q10: multiply Q with 10, to shift from inverse Angstrom to inverse nm
IMult: Multiply intensity with given factor. 100.0 will move from inverse␣
,!cm to inverse m.
"""
df, dfrange = readDat(filenamelist[0], eMin = eMin, outputRanges = True,␣
,!Q10 = True)
ranges = [dfrange]
for filename in filenamelist[1:]:
df2, dfrange = readDat(filename, eMin = eMin, outputRanges = True, Q10␣
,!= True)
df = df.append(df2, ignore_index = True)
ranges.append(dfrange)
if outputRanges:
return df, ranges
```

```
else:
return df
# reads merged datafiles in three-column format
def readDat(filename,
eMin = 0.01,
CSVOpts = None, outputRanges = False,
Q10 = False, IMult = 1.):
"""
Q10: multiply Q with 10, to shift from inverse Angstrom to inverse nm
IMult: Multiply intensity with given factor. 100.0 will move from inverse␣
,!cm to inverse m.
"""
CSVArgs = {"sep" : '\t',
"skipinitialspace" : True,
"skip_blank_lines" : True,
"skiprows" : 0,
"engine": "python",
# "nrows" : 1024,
"header" : None,
"names" : ['Q','I', 'IError'],
"decimal": ",",
"index_col" : False}

if CSVOpts is not None:
CSVArgs.update(CSVOpts)
df = pandas.read_csv(filename, **CSVArgs)
if not "IError" in df:
df["IError"] = eMin * df["I"]
df["IError"].clip(lower=eMin * df["I"])
if Q10:
df.Q *= 10.
df["QError"]= df.Q * 0.01
df.I *= IMult
df.IError *= IMult
ranges = [df.Q.min(), df.Q.max()]
if not outputRanges:
return df
else:
return df, ranges
def mergyMagic(df, nBin = 500, qBounds = None, unweighted = False):
if qBounds is None:
qMin, qMax = df.Q.min(), df.Q.max()
else:
qMin, qMax = np.min(qBounds), np.max(qBounds)
# define weighted standard error on the mean:
def SEMw(x, w):
# function adapted from: https://stats.stackexchange.com/questions/
,!25895/computing-standard-error-in-weighted-mean-estimation
# citing: The main reference is this paper, by Donald F. Gatz and␣
,!Luther Smith, where 3 formula based estimators are compared with bootstrap␣
,!results. The best approximation to the bootstrap result comes from Cochran␣
,!(1977):
# side note: this provides rubbish results.
n = len(w)
xWbar = np.average(x, weights = w)
```

```
wbar = np.mean(w)
out = (
n / ((n - 1) * np.sum(w)**2)
* (np.sum((w * x - wbar * xWbar)**2)
- 2 * xWbar * np.sum((w - wbar) * (w * x - wbar * xWbar))
+ xWbar**2 * np.sum((w - wbar)**2))
)
return out

# pass through everything outside the bin range
# prepare bin edges:
binEdges = np.logspace(np.log10(qMin), np.log10(qMax), num = nBin + 1)
binDat = pandas.DataFrame(data = {
"Q" : np.full(nBin, np.nan), # mean Q
"I" : np.full(nBin, np.nan), # mean intensity
"IStd" : np.full(nBin, np.nan), # standard deviation of the mean␣
,!intensity
"ISEM" : np.full(nBin, np.nan), # standard error on mean of the mean␣
,!intensity (maybe, but weighted is hard.)
"ISEMw" : np.full(nBin, np.nan), # weighted standard error on mean of␣
,!the mean intensity
"IError": np.full(nBin, np.nan), # Propagated errors of the intensity
"IE" : np.full(nBin, np.nan), # Combined error estimate of the␣
,!intensity
"QStd" : np.full(nBin, np.nan), # standard deviation of the mean Q
"QSEM" : np.full(nBin, np.nan), # standard deviation of the mean Q
# "QError": np.full(nBin, np.nan), # Propagated errors on the mean Q
"QE" : np.full(nBin, np.nan), # Combined errors on the mean Q
})
# add a little to the end to ensure the last datapoint is captured:
binEdges[-1] = binEdges[-1] + 1e-3 * (binEdges[-1] - binEdges[-2])
# now do the binning per bin.
for binN in range(len(binEdges) - 1):
# this almost works, except for the very last datapoint.
# Perhaps shift the last edge a little outward?
dfRange = df.query('{} <= Q < {}'.format(binEdges[binN],␣
,!binEdges[binN+1])).copy()
if len(dfRange) == 0:
# print("ping")
pass
elif len(dfRange) == 1:
# might not be necessary to do this..
# print("pong")
# can't do stats on this:
binDat.Q.loc[binN] = float(dfRange.Q)
binDat.I.loc[binN] = float(dfRange.I)
binDat.IError.loc[binN] = float(dfRange.IError)
binDat.IStd.loc[binN] = float(dfRange.IError)
binDat.ISEM.loc[binN] = float(dfRange.IError)
binDat.ISEMw.loc[binN] = float(dfRange.IError)
binDat.IE.loc[binN] = float(dfRange.IError)
binDat.QStd.loc[binN] = float(dfRange.QError)

binDat.QE.loc[binN] = binDat.QSEM.loc[binN]
else:#
```

```
    dfRange.IError.clip_lower(dfRange.I * 0.01) # clip to minimum␣
    ,!uncertainty
    # dfRange["wt"] = np.abs((dfRange.I / dfRange.IError)**2) # inverse␣
    ,!relative weight per point
    dfRange["wt"] = np.abs(dfRange.I / (dfRange.IError**2)) # inverse␣
    ,!relative weight per point
    if unweighted:
    dfRange["wt"] = np.ones(dfRange.I.shape) # np.abs(dfRange.I /␣
    ,!(dfRange.IError**2)) # inverse relative weight per point
    # dfRange.wt /= dfRange.wt.max() # normalization, probably not␣
    ,!necessary
    DSI = DescrStatsW(dfRange.I, weights = dfRange.wt)
    DSQ = DescrStatsW(dfRange.Q, weights = dfRange.wt)
    binDat.Q.loc[binN] = DSQ.mean
    binDat.I.loc[binN] = DSI.mean
    binDat.IError.loc[binN] = np.sqrt(((dfRange.wt * dfRange.
    ,!IError)**2).sum()) / dfRange.wt.sum()
    binDat.IStd.loc[binN] = DSI.std
    # following suggestion regarding V1/V2 from: https://groups.google.
    ,!com/forum/#!topic/medstats/H4SFKPBDAAM
    binDat.ISEM.loc[binN] = DSI.std * np.sqrt((dfRange.wt**2).sum() /␣
    ,!(dfRange.wt.sum())**2)
    binDat.ISEMw.loc[binN] = SEMw(dfRange.I, dfRange.wt)
    binDat.IE.loc[binN] = np.max([binDat.ISEM[binN], binDat.
    ,!IError[binN], DSI.mean * 1e-2])
    binDat.QStd.loc[binN] = DSQ.std
    binDat.QSEM.loc[binN] = DSQ.std * np.sqrt((dfRange.wt**2).sum() /␣
    ,!(dfRange.wt.sum())**2)
    binDat.QE.loc[binN] = binDat.QSEM.loc[binN]
    # remove empty bins
    binDat.dropna(thresh = 4, inplace = True)
    return binDat
```

Section 1: Testing the file reading methods

```
[3]:  filename = Path('data/20170831_00014_WAXS_0005.dat')
      dset = readDat(filename, Q10 = True, eMin = 0.01)
[4]:  dset.plot(x='Q', y="I", yerr = "IError", logx = True, logy = True )
[4]:  <matplotlib.axes._subplots.AxesSubplot at 0x1a2510dc90>
```

Section 2: can we read, combine, and plot, all the data?

```
[68]: filestarts = []
      # find all the data files in the data directory
      filenames = sorted(Path('data/').glob("*SAXS*000*.dat"))
      for fname in filenames:
      # define an identifier for each dataset (SAXS+WAXS)
      '20170831_00014': 'EP unfilled',
      '20171020_00026': 'Boehmite nanofiller',
      '20170831_00019': 'EP + 1 wt.% BNP',
      '20170831_00024': 'EP + 2.5 wt.% BNP',
      '20170831_00029': 'EP + 5 wt.% BNP',
```

```
'20170831_00034': 'EP + 10 wt.% BNP',
'20170831_00039': 'EP + 15 wt.% BNP',
})
massfracs = {}
massfracs.update({
'20170831_00014': 0,
'20171020_00026': 0.66,
'20170831_00019': 0.01,
'20170831_00024': 0.025,
'20170831_00029': 0.05,
'20170831_00034': 0.1,
'20170831_00039': 0.15,
})
sampleFilestarts = ['20170831_00039', '20170831_00034', '20170831_00029',␣
,!'20170831_00024', '20170831_00019']
filestarts.append(fname.stem.rsplit("_",2)[0])
# set the relevant dataset titles based on the first part of the filenames
titles = {}
titles.update({

'20170831_00014': 'EP unfilled',
'20171020_00026': 'Boehmite nanofiller',
'20170831_00019': 'EP + 1 wt.% BNP',
'20170831_00024': 'EP + 2.5 wt.% BNP',
'20170831_00029': 'EP + 5 wt.% BNP',
'20170831_00034': 'EP + 10 wt.% BNP',
'20170831_00039': 'EP + 15 wt.% BNP',
})
massfracs = {}
massfracs.update({
'20170831_00014': 0,
'20171020_00026': 0.66,
'20170831_00019': 0.01,
'20170831_00024': 0.025,
'20170831_00029': 0.05,
'20170831_00034': 0.1,
'20170831_00039': 0.15,
})
sampleFilestarts = ['20170831_00039', '20170831_00034', '20170831_00029',␣
        ,!'20170831_00024', '20170831_00019']
```

[6]:
```
# read these files in pairs, and merge the data using datamerge (weighted by␣
,!uncertainty):
bindats = {}
for fstart in filestarts:
# del df, binDat, binDat_range1, binDat_range2
df, dfRanges = readList(sorted(Path('data/').glob(f"{fstart}*AXS_00*.
,!dat")), outputRanges = True)
df.sort_values('Q', inplace = True)
# uncomment this to set a specific min Q and max Q between which data is␣
,!merged, other
qCrossover = 8
binDat_range1 = mergyMagic(df, nBin = 100, qBounds = [df.Q.min(),␣
,!qCrossover])
binDat_range2 = mergyMagic(df, nBin = 300, qBounds = [qCrossover, df.Q.
,!max()])
```

```
binDat=pandas.concat([binDat_range1, binDat_range2], ignore_index=True)
binDat.to_csv(f'data/{fstart}_concatDat.dat', columns = ['Q', 'I', 'IE'],␣
,/sep = '\t', index = False, header = False)
bindats[fstart] = binDat
```

[7]:
```
# let's plot this data in WAXS and overall
fh, ((ah1, ah2)) = plt.subplots(nrows = 1, ncols = 2, figsize = [12,5])

for fstart in filestarts:
if fstart == '20171020_00026':
continue
binDat = bindats[fstart]
ah1 = binDat.plot("Q", "I", yerr = "IE", xerr = "QE", logx = True, logy =␣
,/True, zorder = 1, ms = 2, ax = ah1, lw = 1, label = titles[fstart], fmt =␣
,/'-', alpha = 1)
ah2 = binDat.plot("Q", "I", yerr = "IE", xerr = "QE", logx = True, logy =␣
,/True, zorder = 1, ms = 2, ax = ah2, lw = 1, label = titles[fstart], fmt =␣
,/'-', alpha = 1)
# filename = Path('data/20170831_00039_SAXS_00005.dat')
# dsetSaxs0p05 = readDat(filename, Q10 = True, eMin = 0.01, IMult = 0.05)
bgnd0p01 = bindats['20171020_00026'].copy()
bgnd0p01.I *= 0.0005
bgnd0p01.IE *= 0.0005
ah2 = bgnd0p01.plot("Q", "I", yerr = "IE", logx = True, logy = True, zorder =␣
,/1, ms = 2, ax = ah2, lw = 1, label = 'Boehmite / 2000', fmt = '--', alpha =␣
,/1)
plt.sca(ah1)
plt.xlim([7, 40])
plt.ylim([10, 50])
plt.xscale('linear')
plt.yscale('linear')
plt.tight_layout()
plt.savefig('allPlots.pdf')
```

[8]:
```
# '20170831_00014': 'EP unfilled',
# '20171020_00026': 'Boehmite nanofiller',
# '20170831_00019': 'EP + 1 wt.% BNP',
# '20170831_00024': 'EP + 2.5 wt.% BNP',
# '20170831_00029': 'EP + 5 wt.% BNP',
# '20170831_00034': 'EP + 10 wt.% BNP',
# '20170831_00039': 'EP + 15 wt.% BNP',
# densities:
densityEP= 1.21 #[g/cm³]
# EP/BNP15: 1,33 [g/cm³]
densityBNP= 3.01 #[g/cm³]
```

[69]:
```
# match the measured PNC data by a linear combination of polymer and boehmite␣
,/signals, and plot this
resultDf = pandas.DataFrame(
index = sampleFilestarts,
columns = ['title', 'boehmiteVolFracXRD', 'boehmiteMassFracXRD',␣
,/'agglomerateVolFracSAXS']
)
for fstart in sampleFilestarts:
resultDf.loc[fstart, 'inputMassfrac'] = massfracs[fstart]
```

```
for fstart in sampleFilestarts:
matchData = bindats[fstart]
dset1 = bindats['20170831_00014']
dset2 = bindats['20171020_00026']
FLC = findLinearCombination(dset1, dset2, matchData,
initialGuess = [1.,1.],
qmin = 7, qmax = np.inf)
        I1, E1, I2, E2, If, Ef = FLC.readScaledDatasets(FLC.sc)
# estimate volume fractions of boehmite in polymer
vf = FLC.sc[1]/(FLC.sc[0]+FLC.sc[1])
# translate to mass fraction:
mf = vf * densityBNP / (vf * densityBNP + (1 - vf) * densityEP)
# save for later
resultDf.loc[fstart, 'boehmiteVolFracXRD'] = float(vf)
resultDf.loc[fstart, 'boehmiteMassFracXRD'] = float(mf)
resultDf.loc[fstart, 'title'] = titles[fstart]
# start plotting:
# right-hand figure
fh, ((ah1, ah2)) = plt.subplots(nrows = 1, ncols = 2, figsize = [12,5])
plt.sca(ah2)
ah2 = matchData.plot("Q", "I", yerr = "IE", xerr = "QE", logx = True, logy␣
,!= True, zorder = 1, ms = 2, ax = ah2, lw = 1, label = titles[fstart], fmt =␣
,!'-', alpha = 1)

plt.errorbar(matchData.Q, I1 + I2, np.sqrt(E1 **2 + E2 **2), label =␣
,!'linear combination of ALOOH + Polymer')
plt.legend()
plt.xlabel('q (nm$^{-1}$)')
plt.ylabel('I (m$^{-1}$ sr$^{-1}$)')
# left-hand figure
plt.sca(ah1)
ah1 = matchData.plot("Q", "I", yerr = "IE", xerr = "QE", logx = True, logy␣
,!= True, zorder = 1, ms = 2, ax = ah1, lw = 1, label = titles[fstart], fmt =␣
,!'-', alpha = 1)
plt.errorbar(matchData.Q, I1 + I2, np.sqrt(E1 **2 + E2 **2), label =␣
,!'linear combination of ALOOH + Polymer')
plt.xlim([7, 40])
plt.ylim([10, 100])
plt.xscale('linear')
plt.yscale('linear')
plt.xlabel('q (nm$^{-1}$)')
plt.ylabel('I (m$^{-1}$ sr$^{-1}$)')
# set title and save figure
fh.suptitle(f'Boehmite volume fraction based on $7 \leq q \leq 40$: {vf*100:
,!0.02f} vol.% (maybe), \n multiplied by BNP/EP density ratio: {mf * 100:0.
,!02f} wt.% (maybe) ')
plt.legend()
plt.savefig(f'{titles[fstart]}_linearCombination.pdf')
# save the SAXS data for fitting with McSAS:
SAXSsubset = matchData.loc[matchData.Q < 3.]
SAXSsubset.to_csv(f'{fstart}_SAXSonly.csv', columns = ['Q', 'I', 'IE'], sep␣
,!= '\t', header = False, index = False)
```

[43]:
```
# export SAXS data for nanofiller only
fstart = '20171020_00026' # boehmite nanofiller
```

```
matchData = bindats[fstart]
SAXSsubset = matchData.loc[matchData.Q < 3.]
SAXSsubset.to_csv(f'{fstart}_SAXSonly.csv', columns = ['Q', 'I', 'IE'], sep =␣
,/'\t', header = False, index = False)
```

Section 3 Analysis of McSAS fits; extracting agglomerate fraction

[71]:
```
for fstart in sampleFilestarts:
    # find the directory where the McSAS results live, by searching for the␣
    ,/date separators:
    resultDf.loc[fstart, 'mcsasStatsFile'] = sorted(Path('.').
    ,/glob(f'{fstart}*-*-*_*-*-*/{fstart}*stats_radius.dat'))[0]
    # note, for this to work, the third and fourth regions of the McSAS␣
    ,/analysis should be rmin < 15, and 15> rmax, respectively.
    data = pandas.read_csv(resultDf.loc[fstart, 'mcsasStatsFile'], sep = ' ',␣
    ,/skipinitialspace= True)
    volFracNonAgg = data.loc[2, 'totalValue']
    volFracAgg = data.loc[3, 'totalValue']
    resultDf.loc[fstart, 'agglomerateVolFracSAXS'] = volFracAgg / (volFracAgg +␣
    ,/volFracNonAgg)
    vf = (volFracAgg + volFracNonAgg)
    mf = vf * densityBNP / (vf * densityBNP + (1 - vf) * densityEP)
    resultDf.loc[fstart, 'boehmiteVolFracSAXS'] = vf
    resultDf.loc[fstart, 'boehmiteMassFracSAXS'] = mf
```

[105]:
```
resultDf.loc[:,('title', 'inputMassfrac', 'boehmiteMassFracSAXS',␣
,/'boehmiteMassFracXRD', 'agglomerateVolFracSAXS')].to_csv('results.csv', sep␣
,/= ';')
resultDf.loc[:,('title', 'inputMassfrac', 'boehmiteMassFracSAXS',␣
,/'boehmiteMassFracXRD', 'agglomerateVolFracSAXS')]
```

[105]:
```
                                    title inputMassfrac boehmiteMassFracSAXS \
20170831_00039 EP + 15 wt.% BNP 0.150 0.137293
20170831_00034 EP + 10 wt.% BNP 0.100 0.094820
20170831_00029 EP + 5 wt.% BNP 0.050 0.047526
20170831_00024 EP + 2.5 wt.% BNP 0.025 0.022976
20170831_00019 EP + 1 wt.% BNP 0.010 0.009507
                boehmiteMassFracXRD agglomerateVolFracSAXS
20170831_00039 0.167756 0.302275
20170831_00034 0.102336 0.331270
20170831_00029 0.041031 0.349372
20170831_00024 0.016601 0.353067
20170831_00019 0.006178 0.376432
```

[73]:
```
resultDf.loc[:,'agglomerateVolFracSAXS'] = resultDf.loc[:
,/,'agglomerateVolFracSAXS'].astype(float)
resultDf.loc[:,'boehmiteMassFracXRD'] = resultDf.loc[:,'boehmiteMassFracXRD'].
,/astype(float)
```

[104]:
```
fh, (ah1, ah2) = plt.subplots(nrows = 1, ncols = 2, figsize = [12, 6], sharex =␣
,/True)
plt.sca(ah1)
ah1 = resultDf.plot(x = 'inputMassfrac', y = 'boehmiteMassFracXRD', ax = ah1,␣
,/label = 'method 1 (XRD)', alpha = 1, kind = 'scatter')
ah1 = resultDf.plot(x = 'inputMassfrac', y = 'boehmiteMassFracSAXS', ax = ah1,␣
,/label = 'method 2 (SAXS)', alpha = 1, kind = 'scatter', color = 'red')
plt.xlabel('input mass fraction')
plt.ylabel('boehmite mass fraction')
```

```
plt.plot([0, 0.15], [0, 0.15], label = 'ideal', color = 'black', linewidth = 1,␣
,/linestyle = '--')
plt.legend()
plt.sca(ah2)
ah2 = resultDf.plot(x = 'inputMassfrac', y = 'agglomerateVolFracSAXS', ax =␣
,/ah2, label = 'method 2 (SAXS)', alpha = 1, kind = 'scatter', color = 'green')
plt.xlabel('input mass fraction')
plt.ylabel('agglomerate fraction')
plt.savefig('fractions.pdf')
```