Following the structural changes of triolein films during lipolysis

Ben A. Humphreys,[a,b,#,] Philipp Gutfreund,[c] Andrew R. McCluskey,[d,e,f] Tomas Arnold,[g,h] Jesper Vind,[i] and Tommy Nylander [a,b,j,k]*

[a]*Physical Chemistry, Department of Chemistry, Lund University, P.O. Box 124, S-221 00 Lund, Sweden.*
[b]*NanoLund, Lund University, SE-221 00 Lund, Sweden.*
[c]*Institut Laue-Langevin, 71 Avenue des Martyrs, CS20156, 38042 Grenoble, France.*
[d]*Centre for Computational Chemistry, School of Chemistry, University of Bristol, Cantock's Close, Bristol BS8 1TS, U.K.*
[e]*European Spallation Source ERIC, Data Management and Software Centre, Asmussens Allé 305, DK-2800 Kongens Lyngby, Denmark.*
[f] *Diamond Light Source, Harwell Campus, Didcot, OX11 0DE, U.K.*
[g]*European Spallation Source ERIC, P.O. Box 176, SE-221 00 Lund, Sweden.*
[h]*ISIS Pulsed Neutron and Muon Source, STFC, Harwell Science and Innovation Campus, OX11 OQX, U.K.*
[i]*Novonesis A/S, Biologiens Vej 2, 2800 Kgs. Lyngby, Denmark.*
[j]*LINXS Institute of advanced Neutron and X-ray Science, Mesongatan 4, 224 84 Lund, Sweden.*
[k]*School of Chemical Engineering and Translational Nanobioscience Research Center, Sungkyunkwan University, Suwon, Republic of Korea.*

[#]*Now at Institut Laue-Langevin, 71 Avenue des Martyrs, CS20156, 38042 Grenoble, France*
**Corresponding author Ben A. Humphreys, email: humphreys@ill.fr, phone: +33 457428331*

Silicon wafer (SE experiments) and silicon block (NR experiments) cleaning protocol.

The silicon wafer or block was plasma cleaned for 10 min then rinsed with ethanol followed by copious Milli-Q water. The wafer or block was then subjected to a basic and acidic clean; both performed at 80 °C for 10 min. The basic mixture was a NH_4OH:H_2O_2:water solution and the acid mixture, HCl:H_2O_2:water, both solutions in the respective ratio of 1:1:5 v/v. Surfaces were rinsed with copious water after basic and acidic exposure. Wafer was then sonicated in Milli-Q water followed by ethanol (10 min each) before being stored in ethanol until required. Silicon blocks were stored under Milli-Q water and used within 24 hours of cleaning.

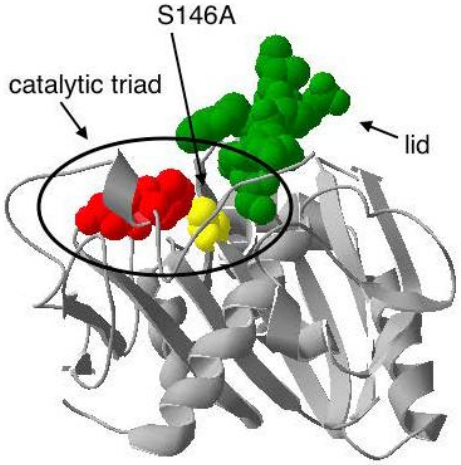


Figure S1: Molecular structure of TLL highlighting the moieties involved in the lid opening process

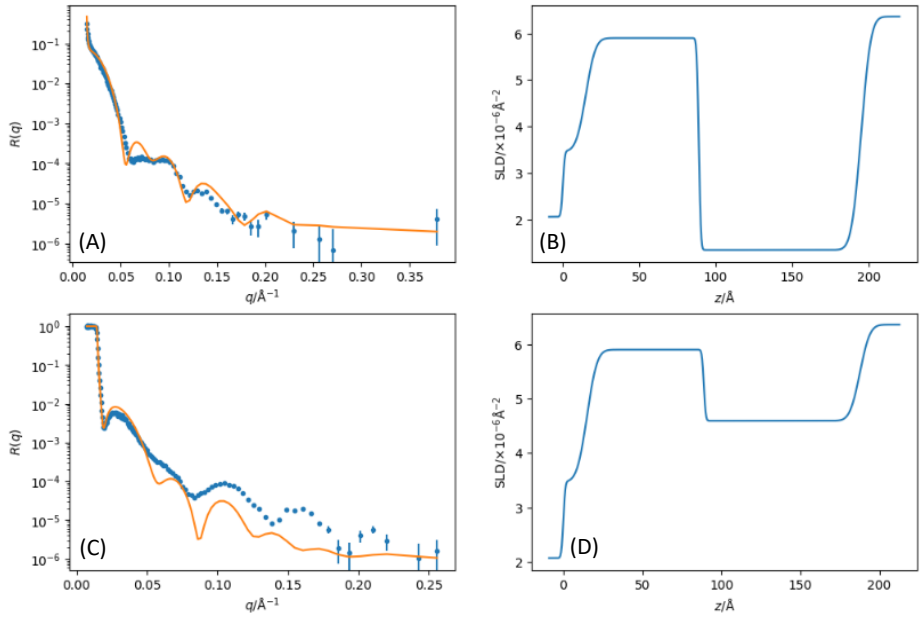


Figure S2: fitted data and SLD profiles for the pD 7.0 initial (A & B) and final (C & D) measurements for a single-slab triolein/product model.

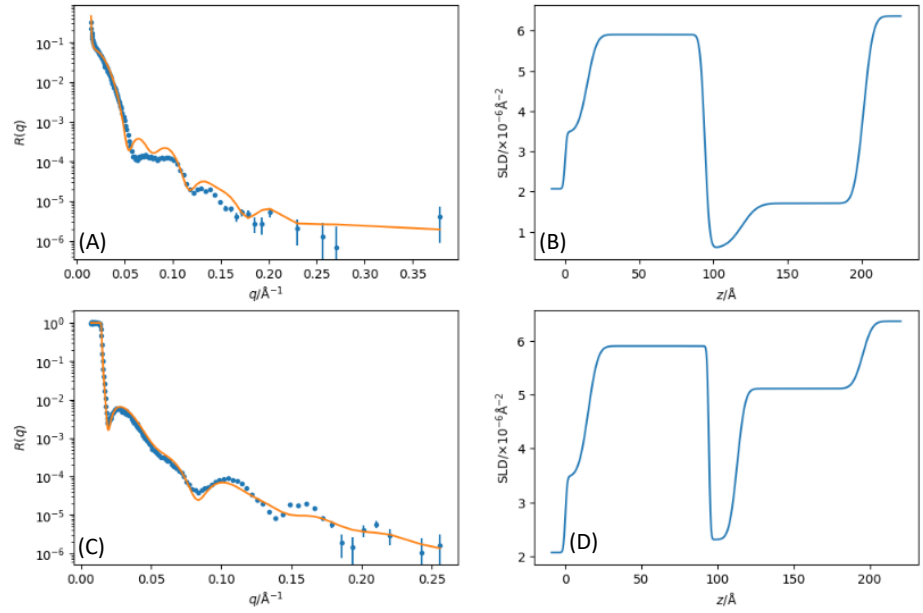


Figure S3: fitted data and SLD profiles for the pD 7.0 initial (A & B) and final (C & D) measurements for a two-slab triolein/product model.

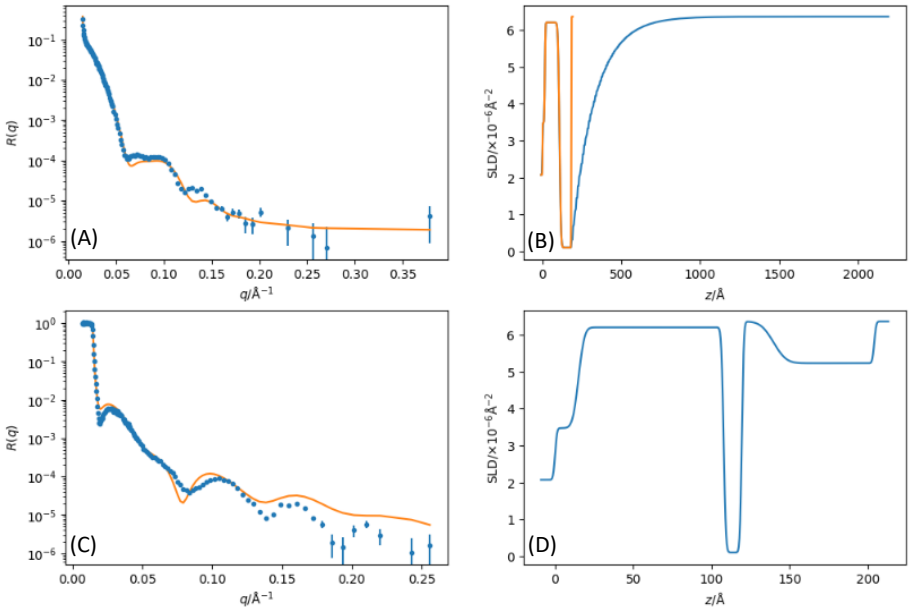


Figure S4: fitted data and SLD profiles for the pD 7.0 initial (A & B) and final (C & D) measurements where a dual model, single-slab triolein (orange line) plus an exponential decay (blue line) was used for the initial measurement while a simple three-slab model was used as the product model.

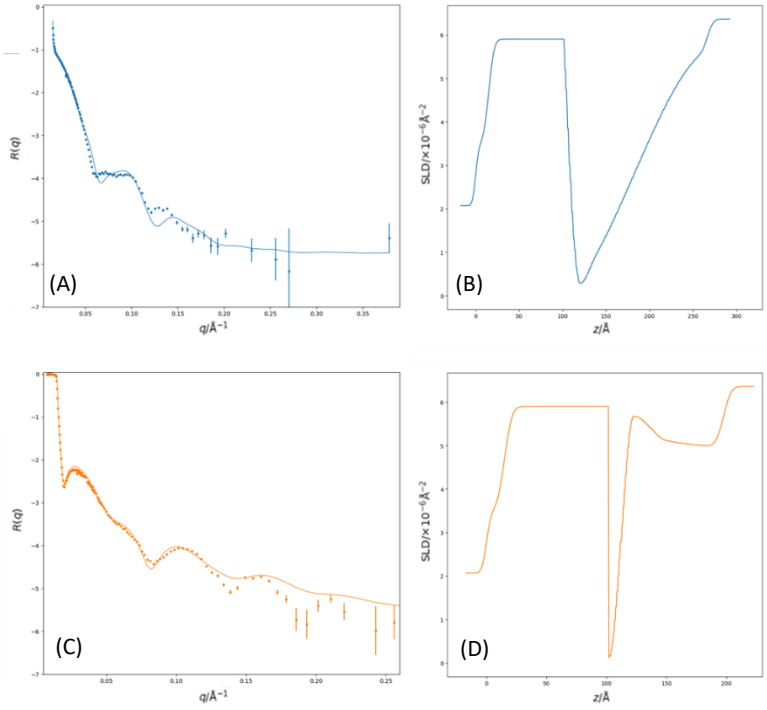


Figure S5: fitted data and SLD profiles for the pD 7.0 initial (A & B) and final (C & D) measurements for a 4-spline triolein/product model.

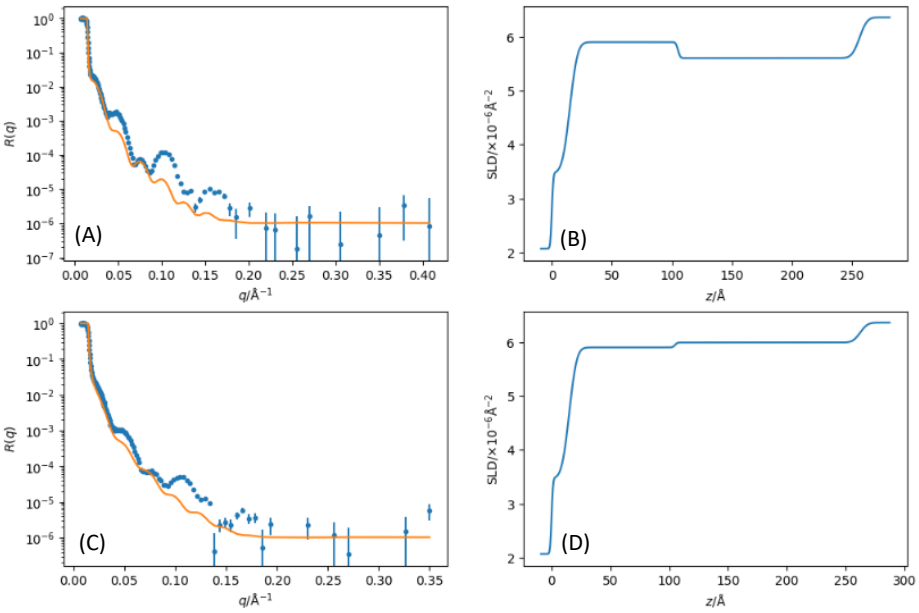


Figure S6: fitted data and SLD profiles for the pD 8.5 initial (A & B) and final (C & D) measurements for a single-slab triolein/product model.

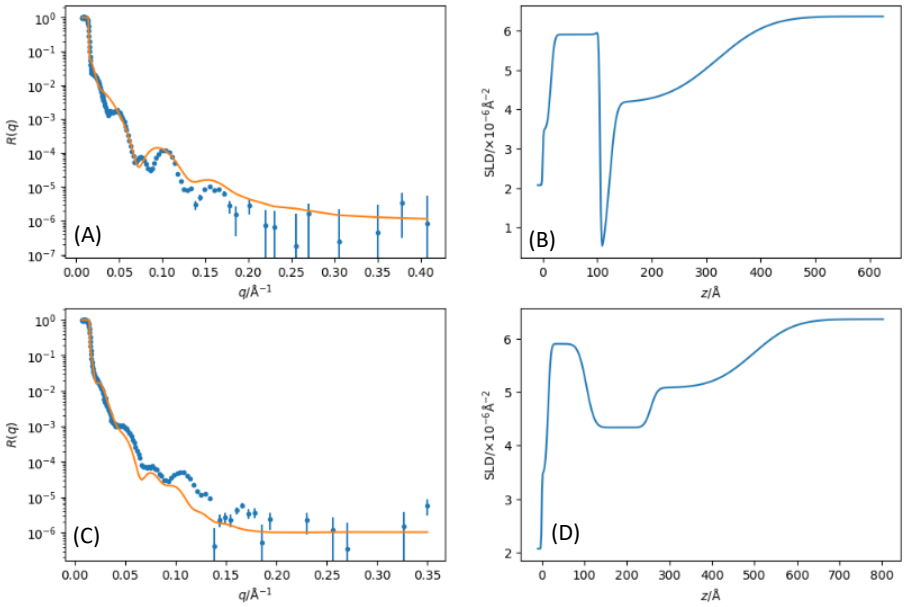


Figure S7: fitted data and SLD profiles for the pD 8.5 initial (A & B) and final (C & D) measurements for a two-slab triolein/product model.

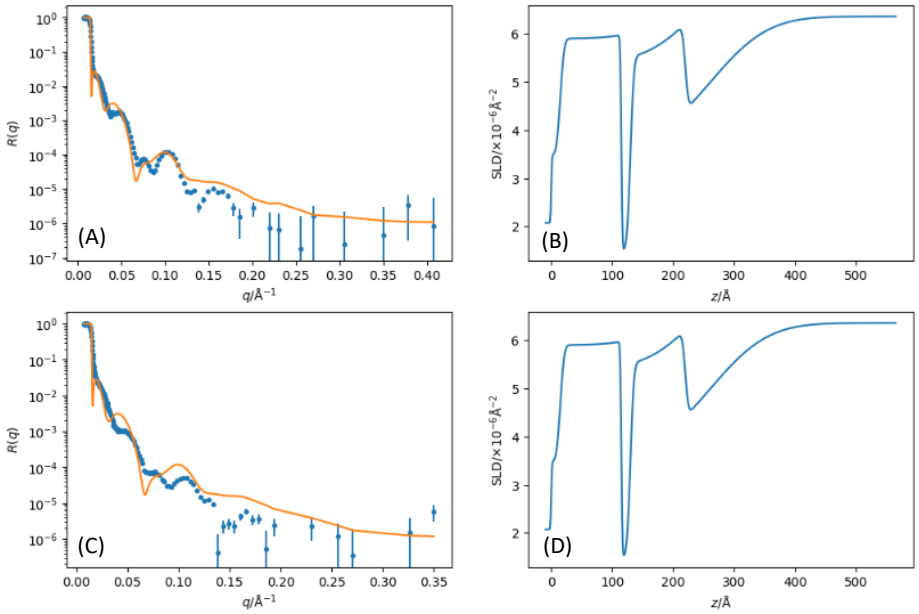


Figure S8: fitted data and SLD profiles for the pD 8.5 initial (A & B) and final (C & D) measurements for a three-slab triolein/product model.

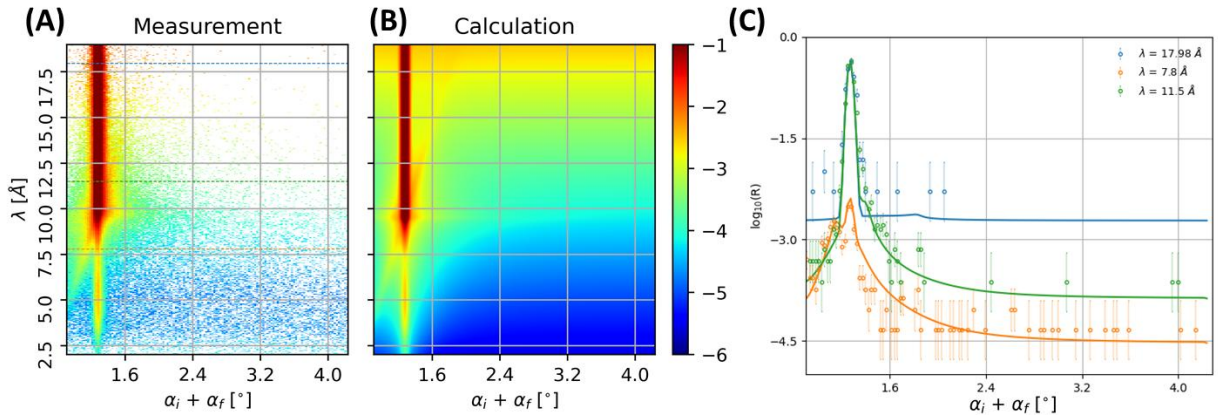


Figure S9: (A) Measurement and (B) DWBA calculation (logarithmic colour scale) for the final pD 7.0 sample after 4 hours of TLL exposure in ($\alpha_i + \alpha_f$, λ) space. (C) The corresponding 1D cuts at different wavelengths indicated in the legend.

This alternative OSS model for the system at pD 7.0 after digestion, assumes a film consisting of 2 µm aggregates floating on top of a thin product layer at the dPS interface, perforated with 10 µm solvent-filled holes. In this model however, the size and position of the water filled holes in the aggregate layer and layer at the dPS interface are now correlated. For this to be achieved, this model required the SLD of the floating aggregate layer to be ~3×10^{-6} Å^{-2}, implying that this layer was aggregated TLL, not the oleic acid produced in the reaction. While this model provides a good fit for the OSS data, it was deemed unrealistic due to the very low concentration of TLL used for the digestion not being sufficient to be visible in the NR measurements. Any accumulation of TLL at the triolein/solvent interface should also be observed in the inactive mutant TLL SE results, however no appreciable increase in thickness was observed. Furthermore, our previous study on this system utilised ATR-FTIR measurements to determine the species present after triolein lipolysis and no signal was observed from the extremely small amount of TLL present.

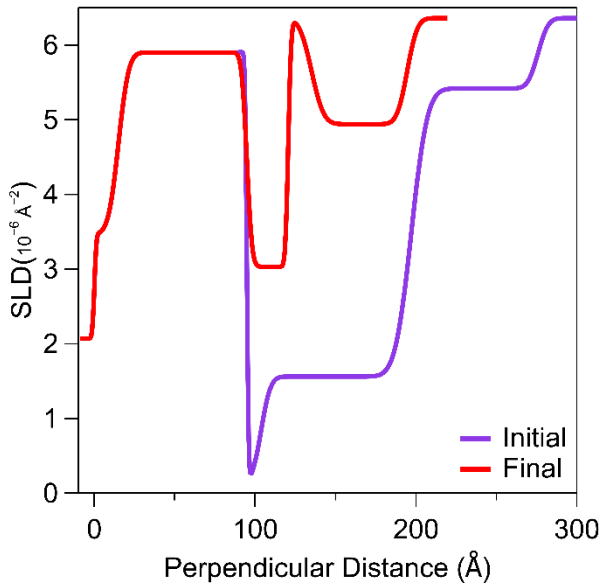


Figure S10: SLD profiles for the pD 7.0 initial (purple line) and final (red line) models. Initial and final SLD's were produced from co-refined models with linked parameters for the silicon, silicon oxide, dPS, and solvent.

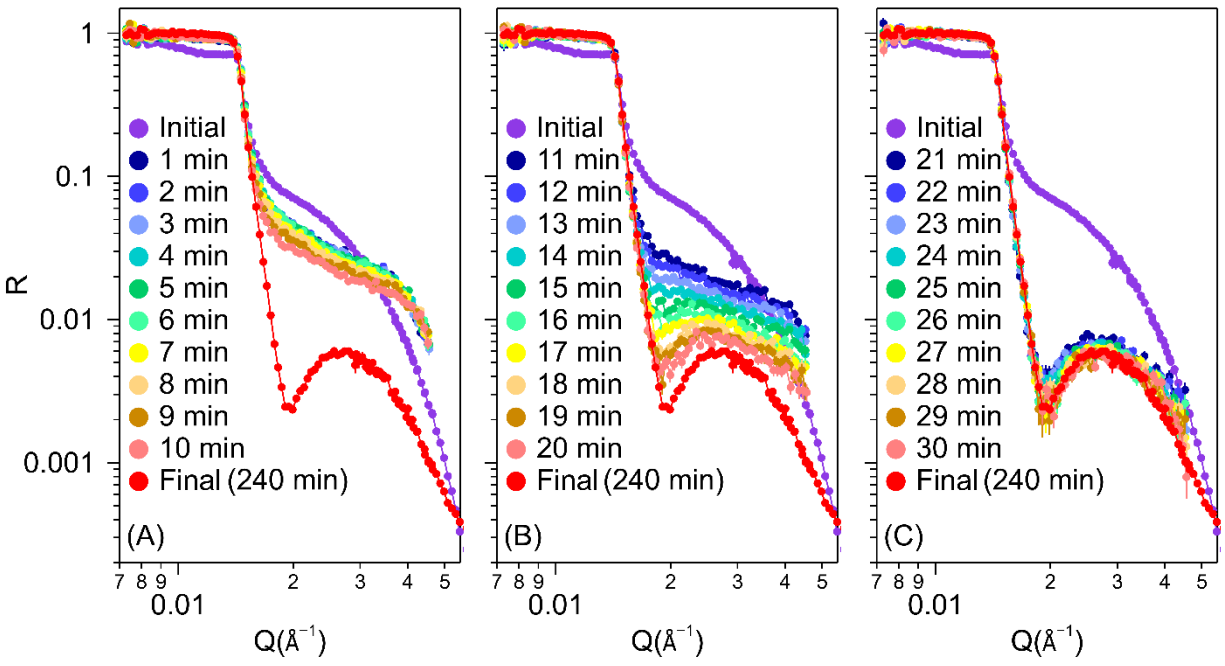


Figure S11: Measured reflectivity profiles following the kinetics of the lipolysis of the triolein film in pD 7.0 solvent after the introduction of 2 ppm TLL for (A) 1 to 10 minutes, (B) 11 to 20 minutes and (C) 21 to 30 minutes. Lines to guide the eye.

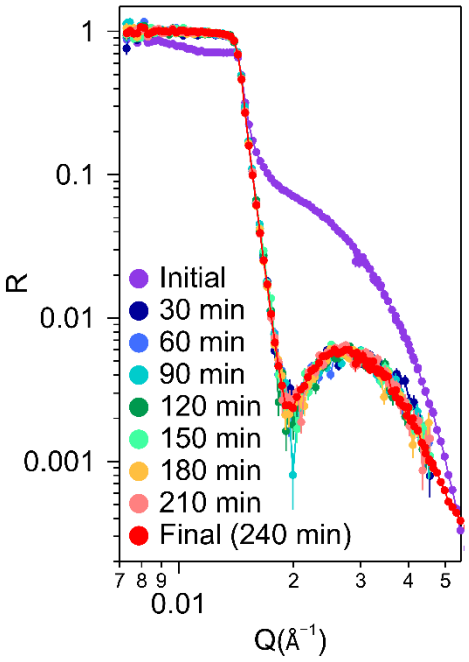


Figure S12: Measured reflectivity profiles for pD 7.0 experiment following the kinetics of the lipolysis of the triolein film after the introduction of 2 ppm TLL from 30 to 240 minutes. Lines to guide the eye.

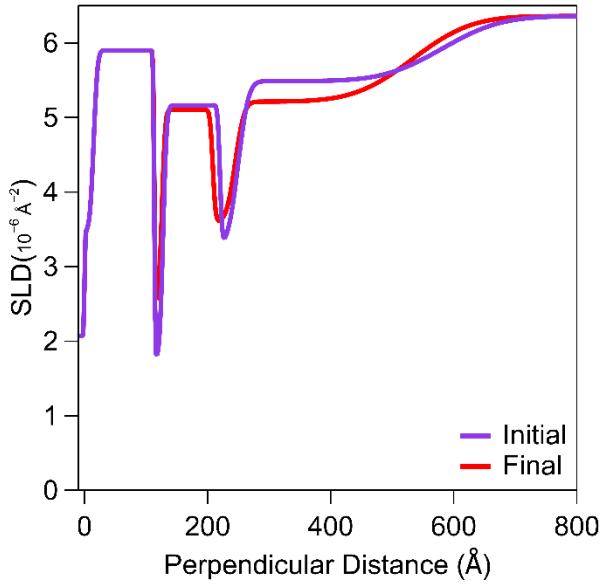


Figure S13: SLD profiles for the pD 8.5 initial and final fitted models. Initial and final SLD's were produced from co-refined models with linked parameters for the silicon, silicon oxide, dPS, and solvent.

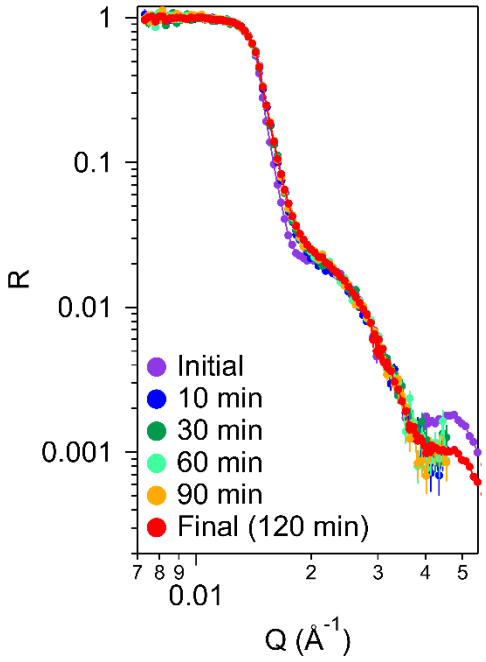


Figure S14: Measured reflectivity profiles for pD 8.5 experiment following the kinetics of the lipolysis of the triolein film after the introduction of 2 ppm TLL from 10 to 120 minutes. Lines to guide the eye.

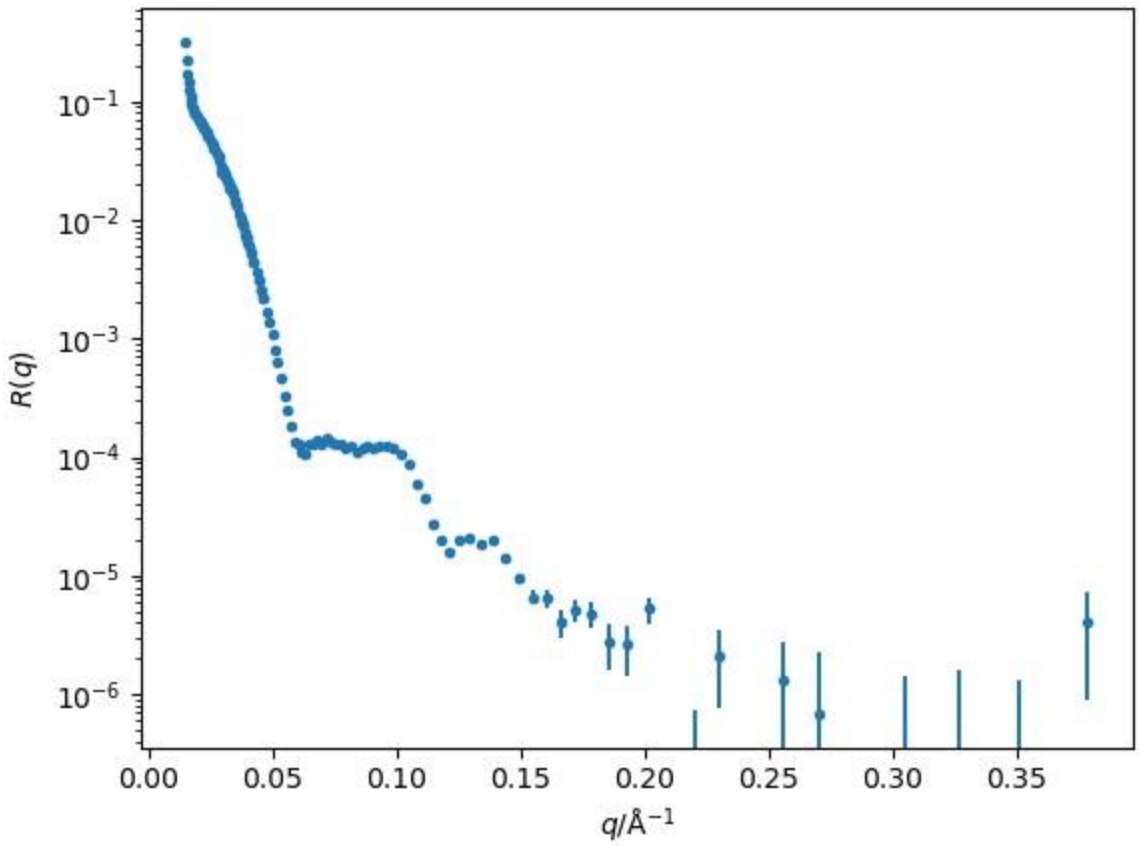
Jypyter Lab notebook for pD 7.0 modelling

In [1]:
```
from refnx.dataset import ReflectDataset
from refnx.reflect import SLD, MixedReflectModel, ReflectModel, Structure from refnx.analysis
import Parameter, Objective, Transform, CurveFitter, GlobalObje

import numpy as np
import matplotlib.pyplot as plt
```

1. Analysis of pure and digested triolein data

The aim of this work is to analyse data from pure triolein and triolein that has been digested by some of the lipase enzyme. For this analysis there are two datasets to consider, the first is the data for the pure triolein on a deuterated polystyrene layer.

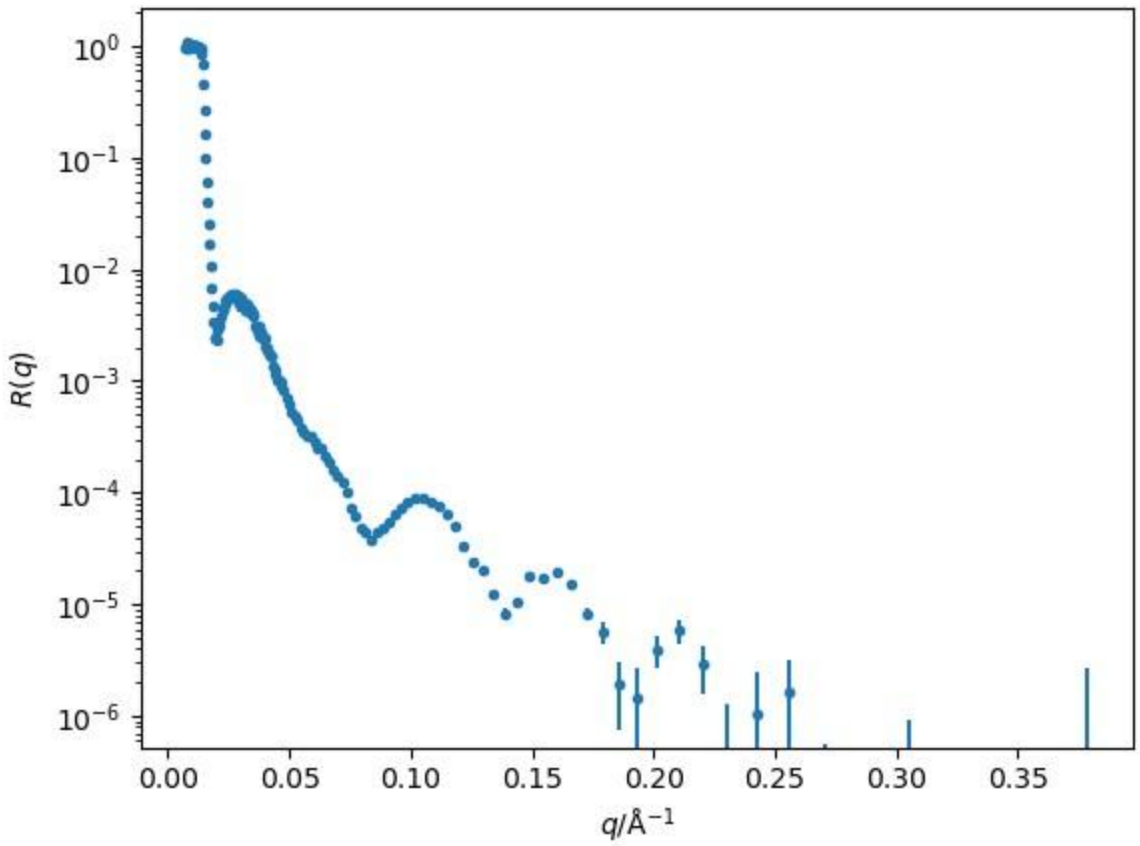
In [2]:
```
pure_data = ReflectDataset('./data/734396_734397_reducedQ.mft')
plt.errorbar(pure_data.x, pure_data.y, pure_data.y_err, marker='.', ls='')
plt.xlabel('$q$/Å$^{-1}$') plt.ylabel('$R(q)$') plt.yscale('log') plt.show()
```



The second dataset is the same layer following the introduction of lipase, which will act to digest the triolein.

In [3]:

```
dig_data = ReflectDataset('./data/734878_734879.mft')
plt.errorbar(dig_data.x, dig_data.y, dig_data.y_err, marker='.', ls='')
plt.xlabel('$q$/Å$^{-1}$') plt.ylabel('$R(q)$')


plt.yscale('log') plt.show()
```



So that the fitting will work, it is necessary that only data where the reflectivity is greater than zero is included.

In [4]:

```
pure_data = ReflectDataset((pure_data.x[pure_data.y > 0],
pure_data.y[pure_data.y > 0], pure_data.y_err[pure_data.y > 0],
pure_data.x_err[pure_data.y > 0])) dig_data =
ReflectDataset((dig_data.x[dig_data.y > 0], dig_data.y[dig_data.y
> 0], dig_data.y_err[dig_data.y > 0], dig_data.x_err[dig_data.y >
0]))
```

1.1 Model for the pure triolein system

First, let's construct the model that will be used for the analysis of the pure triolein. The model that is being used can be described as:

Silicon

SiO2 (15 Å)

Polystyrene (~80 Å)

Triolein layer 1 + solvent (~20 Å)

Triolein layer 2 + solvent (~90 Å)

Triolein layer 3 + solvent (~80 Å)

D2O

Above, there is a superphase, in the form of the silicon/SiO2/polystyrene layers and 3 triolein layers with increasing degrees of hydration before entering the D2O subphase.

It is now possible to define the materials for the pure triolein system. The SLD for triolien layers ranges from 0.1 (pure triolein) to 6.36 (pure solvent).

In [5]:
```
si = SLD(2.07, name='Si') SiO2 =
SLD(3.47, name='SiO2')
polystyrene = SLD(5.9, name='Polystyrene')
triolein1 = SLD(0.1, name='Triolein1') triolein2 =
SLD(1.5, name='Triolein2') triolein3 = SLD(5.4,
name='Triolein3') d2o = SLD(6.36, name='D2O')
```

Follow this, the simple layers can be defined and some constraints introduced.

In [6]:
```
si_layer = si(0, 0)
SiO2_layer = SiO2(15, 1)
polystyrene_layer = polystyrene(80, 5) triolein_layer1 =
triolein1(20, 0)
triolein_layer1.rough.setp(constraint=triolein_layer1.thick * 0.1)
#triolein_layer1.rough.setp(vary=True, bounds=(0.5, 1.5))
triolein_layer2 = triolein2(90, 0)
triolein_layer2.rough.setp(constraint=triolein_layer2.thick * 0.05)
#triolein_layer2.rough.setp(vary=True, bounds=(0.5, 1.5))
triolein_layer3 = triolein3(80, 0)
triolein_layer3.rough.setp(constraint=triolein_layer3.thick * 0.1) d2o_layer = d2o(0, 5)
```

We have now defined the necessary layers to construct the model.

In [7]:
```
pure_triolein = [si_layer, SiO2_layer, polystyrene_layer, triolein_layer1,
                 triolein_layer2, triolein_layer3, d2o_layer]
```

We can then constructure a model for this system.

In [8]:
```
pure_model = ReflectModel(Structure(pure_triolein), bkg=1.8e-6,
                          dq=pure_data.x_err)
```

Then the variable parameters for this model can be set and their bounds defined. Parameter, Minimum, Maximum, Thickness of polystyrene layer, 70, 90. Thickness of triolein layer1, 5, 25. SLD triolein layer1, 0.1, 1.0. Thickness of triolein layer2, 70, 120. SLD triolein layer2, 1.0, 2.0. Thickness of triolein layer3, 50, 100. SLD triolein layer3, 4.5, 6.36.

In [9]:
```
polystyrene_layer.thick.vary = True
polystyrene_layer.thick.bounds = (70, 90)
triolein_layer1.thick.vary = True
triolein_layer1.thick.bounds = (5, 25)
triolein1.real.vary = True  triolein1.real.bounds =
(0.1, 1) triolein_layer2.thick.vary = True
triolein_layer2.thick.bounds = (70, 120)
triolein2.real.vary = True  triolein2.real.bounds =
(1, 2) triolein_layer3.thick.vary = True
triolein_layer3.thick.bounds = (50, 100)
triolein3.real.vary = True  triolein3.real.bounds =
(4.5, 6.36)
```

1.2 Model for the digested triolein system

The next model that we need is that for the digested triolein system:

Silicon

SiO2 (15 Å)

Polystyrene (~80 Å)

Triolein layer 1 + solvent (~20 Å)

D2O (~25 Å)

Triolein layer 2 + solvent (~60 Å)

D2O

The layers of triolein will include some volume fraction of D2O and the silicon and polystyrene layers will be the same layers as for the pure system (thereby constraining the model over two datasets).

Therefore, we define three new layers as shown below.

In [10]:
```
triolein_1 = SLD(3, name='Triolein_1') triolein_layer_1 =
triolein_1(20, 2)
triolein_layer_1.rough.setp(constraint=triolein_layer_1.thick * 0.1) d2o_inter_layer = d2o(25,
0)
d2o_inter_layer.rough.setp(constraint=d2o_inter_layer.thick * 0.1) triolein_2 =
SLD(4, name='Triolein_2') triolein_layer_2 = triolein_2(60, 0)
triolein_layer_2.rough.setp(constraint=triolein_layer_2.thick * 0.1)
```

The digested system is then constructed from the appropriate layers and the model is created.

In [11]:
```
dig_layers = [si_layer, SiO2_layer, polystyrene_layer, triolein_layer_1,          d2o_inter_layer,
triolein_layer_2, d2o_layer]
```

In [12]:
```
dig_model = ReflectModel(Structure(dig_layers), bkg=1e-6,
dq=dig_data.x_err)
```

Then the variable parameters for this model can be set and their bounds defined. Parameter, Minimum, Maximum, Thickness of triolein layer_1, 20, 40. SLD triolein layer_1, 2.5, 4.0. Thickness of D2O inter layer, 5, 40.
Thickness of triolein layer_2, 50, 80. SLD triolein layer_2, 4.0, 6.36.

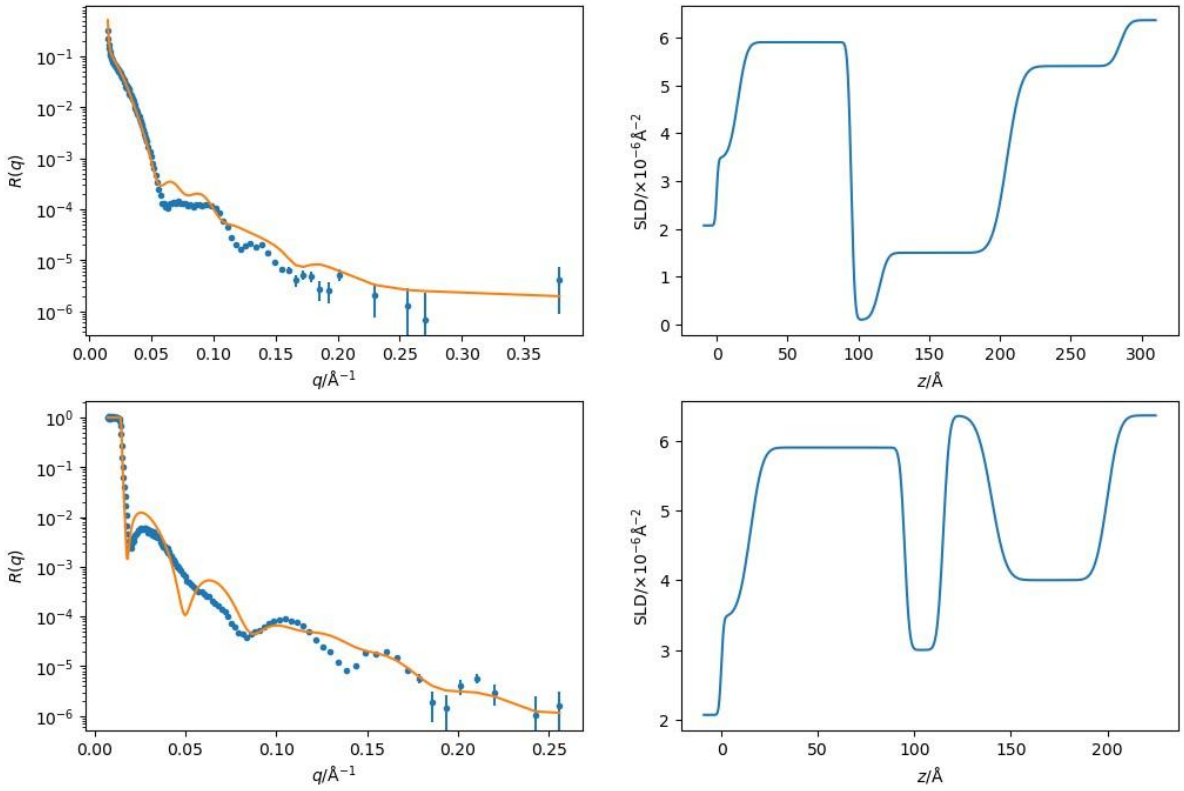
In [13]:
```
triolein_layer_1.thick.vary = True
triolein_layer_1.thick.bounds = (20, 40)
triolein_1.real.vary = True  triolein_1.real.bounds =
(2.5, 4.0) d2o_inter_layer.thick.vary = True
d2o_inter_layer.thick.bounds = (5, 40)
triolein_layer_2.thick.vary = True
triolein_layer_2.thick.bounds = (50, 80)
triolein_2.real.vary = True  triolein_2.real.bounds =
(4.0, 6.36)
```

we can then plot the results for the initial guesses

In [14]:

```
fig, ax = plt.subplots(2, 2, figsize=(12, 8)) ax[0, 0].errorbar(pure_data.x, pure_data.y,
pure_data.y_err, marker='.', ls='') ax[0, 0].plot(pure_data.x, pure_model(pure_data.x,
x_err=pure_data.x_err), zorder=  ax[0, 0].set_xlabel('$q$/Å$^{-1}$') ax[0, 0].set_ylabel('$R(q)$')
ax[0, 0].set_yscale('log')

ax[0, 1].plot(*pure_model.structure.sld_profile()) ax[0,
1].set_xlabel('$z$/Å')
ax[0, 1].set_ylabel(r'SLD/$\times10^{-6}$Å$^{-2}$')

ax[1, 0].errorbar(dig_data.x, dig_data.y, dig_data.y_err, marker='.', ls='') ax[1,
0].plot(dig_data.x, dig_model(dig_data.x, x_err=dig_data.x_err), zorder=10) ax[1,
0].set_xlabel('$q$/Å$^{-1}$') ax[1, 0].set_ylabel('$R(q)$') ax[1, 0].set_yscale('log')
ax[1, 1].plot(*dig_model.structure.sld_profile()) ax[1,
1].set_xlabel('$z$/Å')
ax[1, 1].set_ylabel(r'SLD/$\times10^{-6}$Å$^{-2}$')
```

Text(0, 0.5, 'SLD/$\\times10^{-6}$Å^{-2}') Out[14]:



1.3 Optimisation of the models

The objective functions, which bring together the data and the model, are then created. These two functions are combined with the GlobalObjective.

In [15]:
```
pure_objective = Objective(pure_model, pure_data, transform=Transform('logY')) dig_objective =
        Objective(dig_model, dig_data, transform=Transform('logY'))
```

In [16]:
```
objective = GlobalObjective(objectives=[pure_objective, dig_objective])
```

Finally, we perform the optimisation with a least squares algorithm.

In [17]:
```
fitter = CurveFitter(objective, nwalkers=128) fitter.fit('least_squares')
```

Out[17]:

message: `ftol` termination condition is satisfied.
success: True status: 2

fun: [-6.289e+01 -2.683e+01 ... -9.320e-02 3.185e-01] x: [7.982e+01 8.752e+00 1.000e-01 9.350e+01 1.562e+00

7.876e+01 5.418e+00 2.575e+01 3.031e+00 1.449e+01

5.944e+01 4.940e+00] cost: 13824.415337979448

jac: [[-4.722e-03 -1.225e+00 ... 0.000e+00 0.000e+00]

[-1.363e-01 -2.297e+00 ... 0.000e+00 0.000e+00]

...

[-3.537e-03 0.000e+00 ... -1.122e-02 7.030e-02]

[9.625e-03 0.000e+00 ... -4.028e-03 1.072e-01]] grad: [7.244e-03 4.502e-03 4.779e+01 5.423e-03 3.257e-03

2.377e-03 2.279e-02 2.402e-03 -6.831e-04 2.614e-04

7.141e-04 -7.524e-03] optimality: 0.127463937410604 active_mask: [0 0 -1 0 0 0 0 0 0 0 0 0] nfev: 11 njev: 11

covar: [[1.580e-02 -1.398e-01 ... 1.552e-02 3.608e-04]

[-1.398e-01 3.471e+00 ... -1.374e-01 -3.193e-03]

...

[1.552e-02 -1.374e-01 ... 1.341e-01 2.074e-03]

[3.608e-04 -3.193e-03 ... 2.074e-03 6.575e-05]] stderr: [1.257e-01 1.863e+00 3.516e-01 1.712e+00 6.246e-03

3.513e-01 7.273e-03 2.413e-01 2.220e-02 1.869e-01

3.662e-01 8.109e-03] we can then check the goodness of fit (chi squared value) and plot the results In [18]:
```
objective.chisqr()
```

```
fig, ax = plt.subplots(2, 2, figsize=(12, 8))
ax[0, 0].errorbar(pure_data.x, pure_data.y, pure_data.y_err, marker='.', ls='') ax[0,
0].plot(pure_data.x, pure_model(pure_data.x, x_err=pure_data.x_err), zorder=  ax[0,
0].set_xlabel('$q$/Å$^{-1}$') ax[0, 0].set_ylabel('$R(q)$') ax[0, 0].set_yscale('log')

ax[0, 1].plot(*pure_model.structure.sld_profile()) ax[0,
1].set_xlabel('$z$/Å')
```

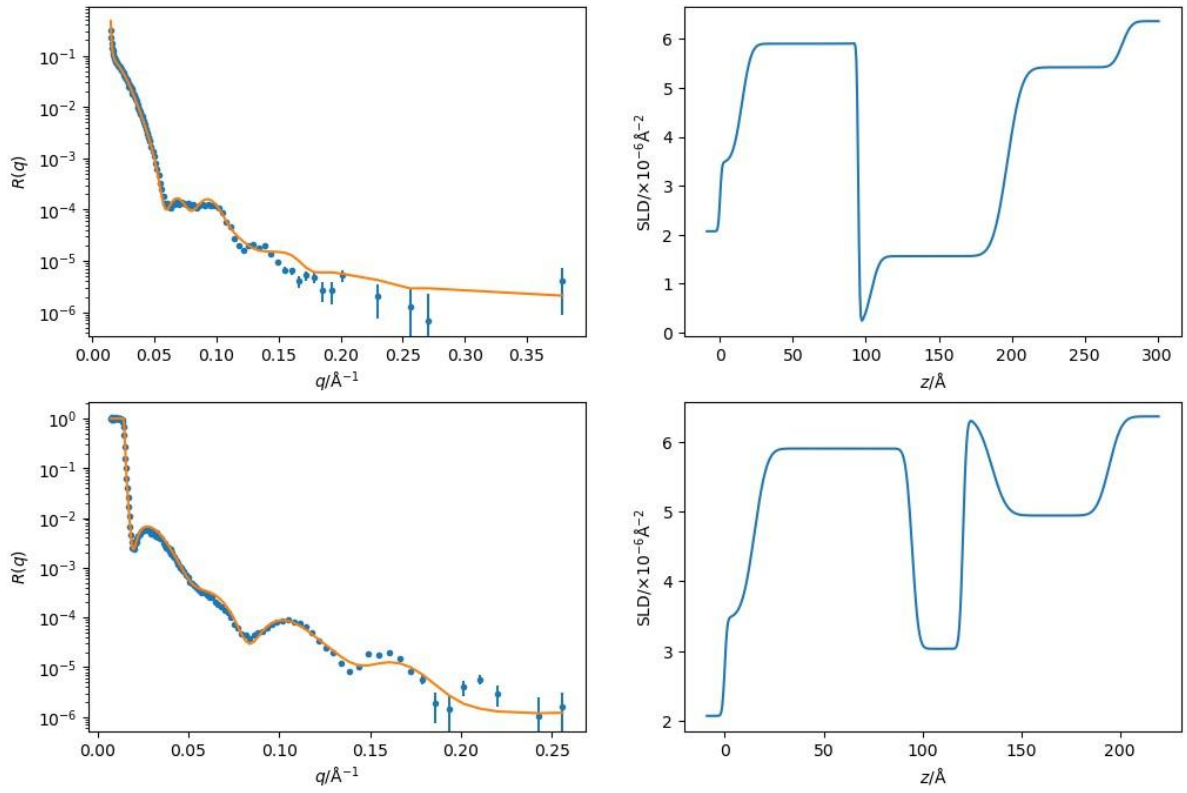
Out[18]: 27648.830675958896

In [19]:

1

```
ax[0, 1].set_ylabel(r'SLD/$\times10^{-6}$Å$^{-2}$')

ax[1, 0].errorbar(dig_data.x, dig_data.y, dig_data.y_err, marker='.', ls='') ax[1,
0].plot(dig_data.x, dig_model(dig_data.x, x_err=dig_data.x_err), zorder=10) ax[1,
0].set_xlabel('$q$/Å$^{-1}$') ax[1, 0].set_ylabel('$R(q)$') ax[1, 0].set_yscale('log')
ax[1, 1].plot(*dig_model.structure.sld_profile()) ax[1,
1].set_xlabel('$z$/Å')
ax[1, 1].set_ylabel(r'SLD/$\times10^{-6}$Å$^{-2}$')
```

Text(0, 0.5, 'SLD/$\\times10^{-6}$Å^{-2}') Out[19]:



In [20]:
```
print(pure_objective.varying_parameters())
```

```
_____________________________________________________________________________

Parameters:     None

<Parameter:'Polystyrene - thick', value=79.8201 +/- 0.126, bounds=[70.0, 90.0]>

<Parameter:'Triolein1 - thick', value=8.75241 +/- 1.86 , bounds=[5.0, 25.0]>

<Parameter:'Triolein1 - sld', value=0.1 +/- 0.352, bounds=[0.1, 1.0]>

<Parameter:'Triolein2 - thick', value=93.5045 +/- 1.71 , bounds=[70.0, 120.0]>

<Parameter:'Triolein2 - sld', value=1.56159 +/- 0.00625, bounds=[1.0, 2.0]>

<Parameter:'Triolein3 - thick', value=78.7551 +/- 0.351, bounds=[50.0, 100.0]>

<Parameter:'Triolein3 - sld', value=5.4181 +/- 0.00727, bounds=[4.5, 6.36]>
```

In [21]:
```
print(dig_objective.varying_parameters())
```

```
_____________________________________________________________________________

Parameters:     None

<Parameter:'Polystyrene - thick', value=79.8201 +/- 0.126, bounds=[70.0, 90.0]>

<Parameter:'Triolein_1 - thick', value=25.752 +/- 0.241, bounds=[20.0, 40.0]>

<Parameter:'Triolein_1 - sld', value=3.03105 +/- 0.0222, bounds=[2.5, 4.0]>

<Parameter: 'D2O - thick' , value=14.4913 +/- 0.187, bounds=[5.0, 40.0]>

<Parameter:'Triolein_2 - thick', value=59.4435 +/- 0.366, bounds=[50.0, 80.0]>

<Parameter:'Triolein_2 - sld', value=4.94041 +/- 0.00811, bounds=[4.0, 6.36]> followed by

optimisation with a differential evolution algorithm.
```

In [22]:
```
fitter = CurveFitter(objective, nwalkers=128) fitter.fit('differential_evolution')
```

```
43it [05:08,  7.17s/it]
 message: Optimization terminated successfully.
```

Out[22]:
```
 success: True     fun: 12869.940298952517

       x: [ 7.982e+01  8.752e+00  1.000e-01  9.350e+01  1.562e+00

            7.875e+01       5.418e+00       2.575e+01       3.031e+00       1.449e+01

5.945e+01  4.940e+00]     nit: 43     nfev: 10390

   jac: [ 8.240e-02 -1.517e-01  4.809e+01 -1.124e-01  8.386e-02          -3.311e-02 -

1.177e-01 -1.213e-01  2.310e-02 -2.701e-01

           -1.031e-01 -8.458e-02]
```

covar: [[1.580e-02 -1.398e-01 ... 1.553e-02 3.608e-04] [-1.398e-01 3.471e+00 ... -1.375e-01 -3.193e-03]

...

[1.553e-02 -1.375e-01 ... 1.342e-01 2.074e-03]

[3.608e-04 -3.193e-03 ... 2.074e-03 6.575e-05]] stderr: [1.257e-01 1.863e+00 3.516e-01 1.712e+00 6.246e-03

3.513e-01 7.273e-03 2.414e-01 2.221e-02 1.869e-01 3.663e-01 8.109e-03] check the goodness of fit (chi squared value) and plot the results again

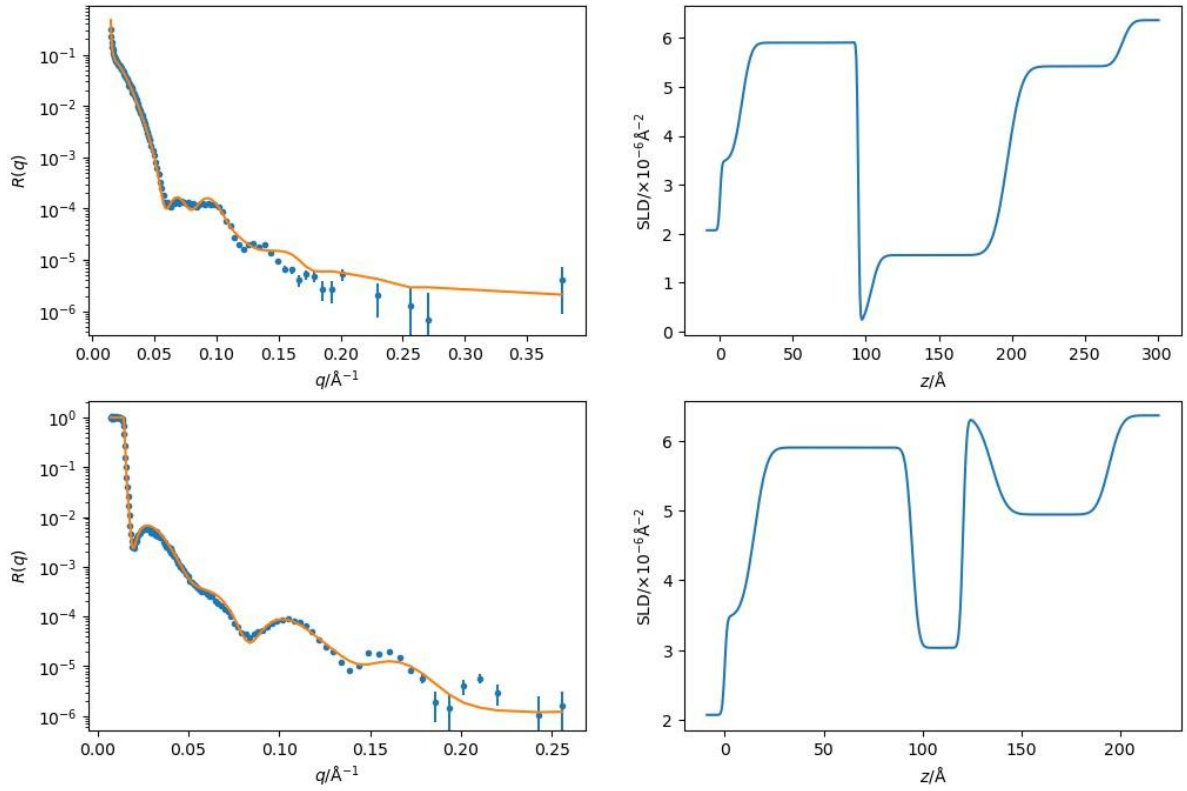
In [23]:
```
objective.chisqr()
```

```
fig, ax = plt.subplots(2, 2, figsize=(12, 8)) ax[0, 0].errorbar(pure_data.x, pure_data.y,
pure_data.y_err, marker='.', ls='') ax[0, 0].plot(pure_data.x, pure_model(pure_data.x,
x_err=pure_data.x_err), zorder= ax[0, 0].set_xlabel('$q$/Å$^{-1}$') ax[0, 0].set_ylabel('$R(q)$')
ax[0, 0].set_yscale('log')

ax[0, 1].plot(*pure_model.structure.sld_profile()) ax[0,
1].set_xlabel('$z$/Å')
ax[0, 1].set_ylabel(r'SLD/$\times10^{-6}$Å$^{-2}$')

ax[1, 0].errorbar(dig_data.x, dig_data.y, dig_data.y_err, marker='.', ls='') ax[1,
0].plot(dig_data.x, dig_model(dig_data.x, x_err=dig_data.x_err), zorder=10) ax[1,
0].set_xlabel('$q$/Å$^{-1}$') ax[1, 0].set_ylabel('$R(q)$') ax[1, 0].set_yscale('log')
ax[1, 1].plot(*dig_model.structure.sld_profile()) ax[1,
1].set_xlabel('$z$/Å')
ax[1, 1].set_ylabel(r'SLD/$\times10^{-6}$Å$^{-2}$')
```

Out[23]: 27648.832255684563

In [24]:

Text(0, 0.5, 'SLD/$\\times10^{-6}$Å^{-2}') Out[24]:



The fitted parameters for each model are printed below.

In [25]:
```
print(pure_objective.varying_parameters())
```

Parameters: None

<Parameter:'Polystyrene - thick', value=79.8204 +/- 0.126, bounds=[70.0, 90.0]>

<Parameter:'Triolein1 - thick', value=8.75216 +/- 1.86 , bounds=[5.0, 25.0]>

<Parameter:'Triolein1 - sld', value=0.1 +/- 0.352, bounds=[0.1, 1.0]>

<Parameter:'Triolein2 - thick', value=93.504 +/- 1.71 , bounds=[70.0, 120.0]>

<Parameter:'Triolein2 - sld', value=1.5616 +/- 0.00625, bounds=[1.0, 2.0]>

<Parameter:'Triolein3 - thick', value=78.7523 +/- 0.351, bounds=[50.0, 100.0]>

<Parameter:'Triolein3 - sld', value=5.41805 +/- 0.00727, bounds=[4.5, 6.36]>

In [26]:
```
print(dig_objective.varying_parameters())
```

Parameters: None

<Parameter:'Polystyrene - thick', value=79.8204 +/- 0.126, bounds=[70.0, 90.0]>

<Parameter:'Triolein_1 - thick', value=25.7482 +/- 0.241, bounds=[20.0, 40.0]>

<Parameter:'Triolein_1 - sld', value=3.0307 +/- 0.0222, bounds=[2.5, 4.0]>

<Parameter: 'D2O - thick' , value=14.4865 +/- 0.187, bounds=[5.0, 40.0]>

<Parameter:'Triolein_2 - thick', value=59.4473 +/- 0.366, bounds=[50.0, 80.0]>

<Parameter:'Triolein_2 - sld', value=4.94039 +/- 0.00811, bounds=[4.0, 6.36]> We can then plot the fitted models against the data to assess the agreement.

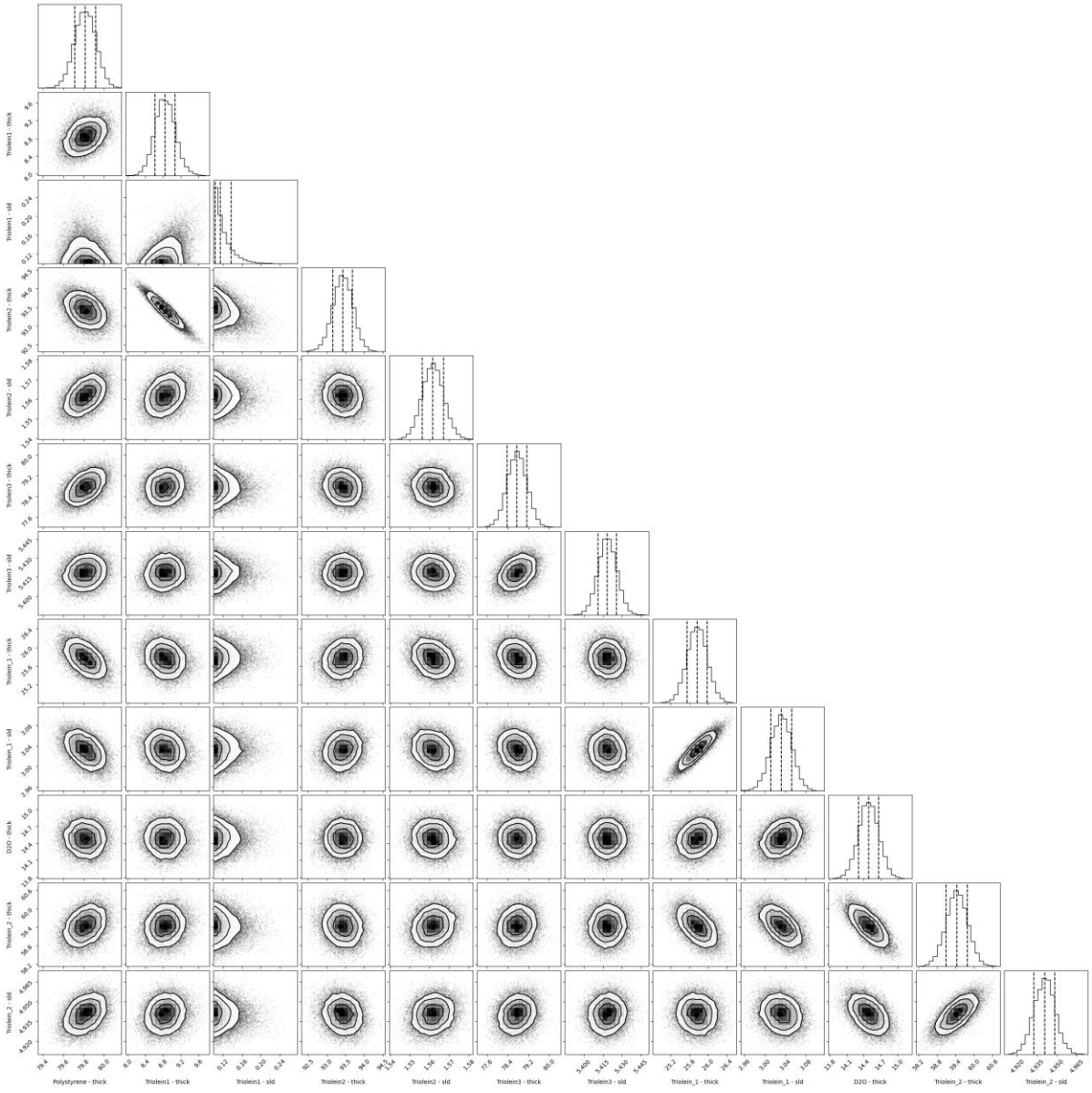
To assess the correlation between the different parameters, Markov chain Monte Carlo sampling is used.

```
In [27]: fitter.sample(1000) fitter.reset() res =
         fitter.sample(200,              nthin=10)
         100%|
```

```
| 1000/1000 [14:08<00:00,  1.18it/s]
100%|
| 2000/2000 [20:18<00:00,  1.64it/s]
```

We visualise this with a corner plot.

```
In [28]: objective.corner() plt.show()
```



check the goodness of fit (chi squared value) and plot the results again

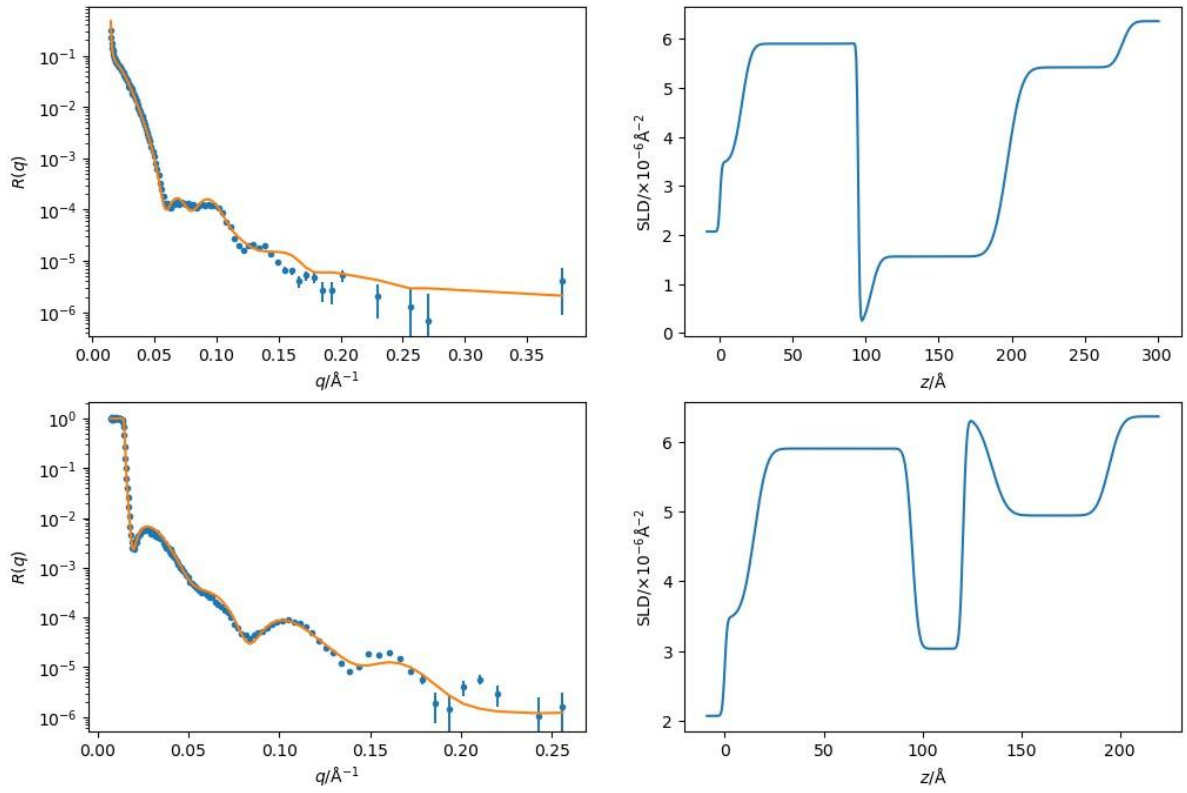
In [29]: objective.chisqr()

Out[29]: 27650.223778191896

In [30]:

```
fig, ax = plt.subplots(2, 2, figsize=(12, 8))
ax[0, 0].errorbar(pure_data.x, pure_data.y, pure_data.y_err, marker='.', ls='') ax[0,
0].plot(pure_data.x, pure_model(pure_data.x, x_err=pure_data.x_err), zorder=  ax[0,
0].set_xlabel('$q$/Å$^{-1}$') ax[0, 0].set_ylabel('$R(q)$') ax[0, 0].set_yscale('log')

ax[0, 1].plot(*pure_model.structure.sld_profile()) ax[0,
1].set_xlabel('$z$/Å')


ax[0, 1].set_ylabel(r'SLD/$\times10^{-6}$Å$^{-2}$')

ax[1, 0].errorbar(dig_data.x, dig_data.y, dig_data.y_err, marker='.', ls='') ax[1,
0].plot(dig_data.x, dig_model(dig_data.x, x_err=dig_data.x_err), zorder=10) ax[1,
0].set_xlabel('$q$/Å$^{-1}$') ax[1, 0].set_ylabel('$R(q)$') ax[1, 0].set_yscale('log')
ax[1, 1].plot(*dig_model.structure.sld_profile()) ax[1,
1].set_xlabel('$z$/Å')
ax[1, 1].set_ylabel(r'SLD/$\times10^{-6}$Å$^{-2}$')
```

Text(0, 0.5, 'SLD/$\\times10^{-6}$Å^{-2}') Out[30]:



In [31]: `print(pure_objective.varying_parameters())`

Parameters: None

<Parameter:'Polystyrene - thick', value=79.8155 +/- 0.105, bounds=[70.0, 90.0]>

<Parameter:'Triolein1 - thick', value=8.84281 +/- 0.229, bounds=[5.0, 25.0]>

<Parameter:'Triolein1 - sld', value=0.114052 +/- 0.017, bounds=[0.1, 1.0]>

<Parameter:'Triolein2 - thick', value=93.428 +/- 0.266, bounds=[70.0, 120.0]>

<Parameter:'Triolein2 - sld', value=1.56174 +/- 0.00544, bounds=[1.0, 2.0]>

<Parameter:'Triolein3 - thick', value=78.7576 +/- 0.381, bounds=[50.0, 100.0]>

<Parameter:'Triolein3 - sld', value=5.41851 +/- 0.00737, bounds=[4.5, 6.36]>

In [32]:
```
print(dig_objective.varying_parameters())
```

Parameters: None

<Parameter:'Polystyrene - thick', value=79.8155 +/- 0.105, bounds=[70.0, 90.0]>

<Parameter:'Triolein_1 - thick', value=25.7625 +/- 0.217, bounds=[20.0, 40.0]>

<Parameter:'Triolein_1 - sld', value=3.03188 +/- 0.0203, bounds=[2.5, 4.0]>

<Parameter: 'D2O - thick' , value=14.4862 +/- 0.178, bounds=[5.0, 40.0]>

<Parameter:'Triolein_2 - thick', value=59.4424 +/- 0.349, bounds=[50.0, 80.0]>

<Parameter:'Triolein_2 - sld', value=4.94072 +/- 0.00794, bounds=[4.0, 6.36]>

In [33]:
```
with open('C:/Users/humphreys/My Drive/refnx/2021 ILL Figaro TLL experiment/CSV/pH7
    np.savetxt(fh, np.array(pure_objective.model.structure.sld_profile()).T, delimi
with open('C:/Users/humphreys/My Drive/refnx/2021 ILL Figaro TLL experiment/CSV/pH7
save_arr_pure = np.array([pure_data.x, pure_data.y, pure_data.y_err, pure_model
np.savetxt(fh, save_arr_pure.T, delimiter=',', header='Q, measured R, measured
```

In [34]:
```
with open('C:/Users/humphreys/My Drive/refnx/2021 ILL Figaro TLL experiment/CSV/pH7
np.savetxt(fh, np.array(dig_objective.model.structure.sld_profile()).T, delimit
with open('C:/Users/humphreys/My Drive/refnx/2021 ILL Figaro TLL experiment/CSV/pH7
save_arr_dig = np.array([dig_data.x, dig_data.y, dig_data.y_err, dig_model(dig_  np.savetxt(fh
save_arr_dig.T, delimiter=',', header='Q, measured R, measured R
```

In [ ]:
```

```

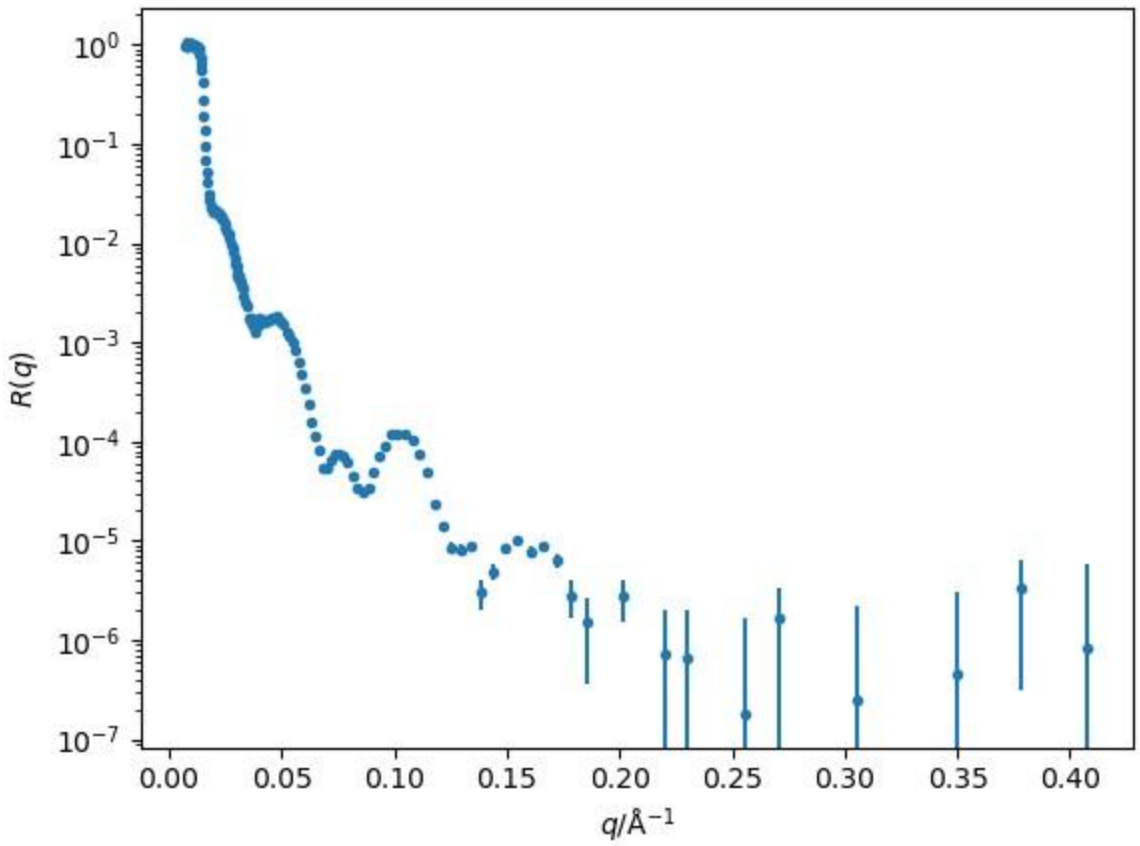
Jypyter Lab notebook for pD 8.5 modelling

In [1]:
```
from refnx.dataset import ReflectDataset
from refnx.reflect import SLD, MixedReflectModel, ReflectModel, Structure from
refnx.analysis import Parameter, Objective, Transform, CurveFitter, GlobalObje

import numpy as np
import matplotlib.pyplot as plt
```

1. Analysis of pure and digested triolein data

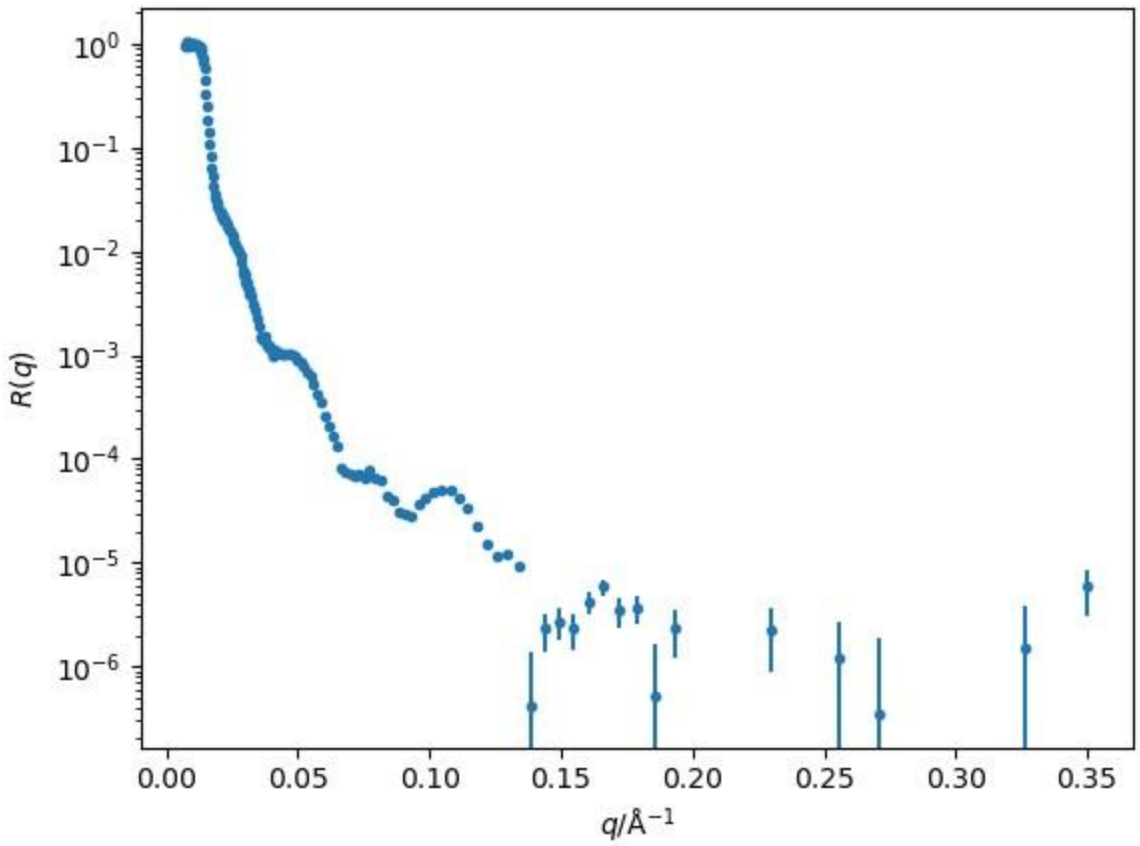
The aim of this work is to analyse data from pure triolein and triolein that has been digested by some of the lipase enzyme. For this analysis there are two datasets to consider, the first is the data for the pure triolein on a polystyrene layer.

In [2]:
```
pure_data = ReflectDataset('./data/735480_735481.mft')
plt.errorbar(pure_data.x, pure_data.y, pure_data.y_err, marker='.', ls='')
plt.xlabel('$q$/Å$^{-1}$') plt.ylabel('$R(q)$') plt.yscale('log')
plt.show()
```



The second dataset is the same layer following the introduction of lipase, which will act to digest the triolein.

In [3]:
```
dig_data = ReflectDataset('./data/735722_735723.mft')
plt.errorbar(dig_data.x, dig_data.y, dig_data.y_err, marker='.', ls='')
plt.xlabel('$q$/Å$^{-1}$') plt.ylabel('$R(q)$')



plt.yscale('log')
plt.show()
```



So that the fitting will work, it is necessary that only data where the reflectivity is greater than zero is included.

In [4]:

```
pure_data = ReflectDataset((pure_data.x[pure_data.y > 0],
pure_data.y[pure_data.y > 0], pure_data.y_err[pure_data.y
> 0], pure_data.x_err[pure_data.y > 0])) dig_data =
ReflectDataset((dig_data.x[dig_data.y > 0],
dig_data.y[dig_data.y > 0], dig_data.y_err[dig_data.y >
0], dig_data.x_err[dig_data.y > 0]))
```

1.1 Model for the pure triolein system

First, let's construct the model that will be used for the analysis of the pure triolein. The model that is being used can be described as:

Silicon

SiO2 (15 Å)

Polystyrene (~100 Å)

Triolein layer 1 + solvent (~15 Å)

Triolein layer 2 + solvent (~90 Å)

Triolein layer 3 + solvent (~40 Å)

Triolein layer 4 + solvent (~400 Å)

D2O

Above, there is a superphase, in the form of the silicon/SiO2/polystyrene layers and 4 triolein layers with increasing degrees of hydration before entering the D2O subphase.

It is now possible to define the materials for the pure triolein system. The SLD for triolien layers ranges from 0.1 (pure triolein) to 6.36 (pure solvent).

```
In [5]: si = SLD(2.07, name='Si') SiO2 = SLD(3.47,
         name= 'SiO2') polystyrene = SLD(5.9,
         name='Polystyrene') triolein1 = SLD(1.4,
         name='Triolein1') triolein2 = SLD(5.4,
         name='Triolein2') triolein3 = SLD(3.6,
         name='Triolein3') triolein4 = SLD(5.6,
         name='Triolein4') d2o = SLD(6.36,
         name='D2O')
```

Follow this, the simple layers can be defined and some constraints introduced.

In [6]:
```
si_layer = si(0, 0)
SiO2_layer = SiO2(15,1)
polystyrene_layer = polystyrene(100, 5) triolein_layer1
= triolein1(15, 2)
triolein_layer1.rough.setp(constraint=triolein_layer1.thick * 0.1) triolein_layer2
= triolein2(90, 2) triolein_layer2.rough.setp(constraint=triolein_layer2.thick *
0.05) triolein_layer3 = triolein3(40, 0)
triolein_layer3.rough.setp(constraint=triolein_layer3.thick * 0.1)
triolein_layer4 = triolein4(400, 10)
triolein_layer4.rough.setp(constraint=triolein_layer4.thick * 0.04) d2o_layer
= d2o(0, 75)
```

We have now defined the necessary layers to construct the model.

In [7]:
```
pure_triolein = [si_layer, SiO2_layer, polystyrene_layer, triolein_layer1,
triolein_layer2, triolein_layer3, triolein_layer4, d2o_layer]
```

We can then constructure a model for this system.

In [8]:
```
pure_model = ReflectModel(Structure(pure_triolein), bkg=1e-6,
dq=pure_data.x_err)
```

Then the variable parameters for this model can be set and their bounds defined. Parameter, Minimum, Maximum, Thickness of polystyrene layer, 90, 110. Thickness of triolein layer1, 5, 30. SLD triolein layer1, 0.1, 6.36. Thickness of triolein layer2, 60, 120. SLD triolein layer2, 0.1, 6.36. Thickness of triolein layer3, 10, 50. SLD triolein layer3, 0.1, 6.36. Thickness of triolein layer4, 200, 500. SLD triolein layer4, 0.1, 6.36.

In [9]:
```
polystyrene_layer.thick.vary = True
polystyrene_layer.thick.bounds = (90, 110)
triolein_layer1.thick.vary = True triolein_layer1.thick.bounds
= (5, 30)

triolein1.real.vary = True
triolein1.real.bounds = (0.1, 6.36)
triolein_layer2.thick.vary = True
triolein_layer2.thick.bounds = (60, 120)
triolein2.real.vary = True
triolein2.real.bounds = (0.1, 6.36)
triolein_layer3.thick.vary = True
triolein_layer3.thick.bounds = (10, 50)
triolein3.real.vary = True
triolein3.real.bounds = (0.1, 6.36)
triolein_layer4.thick.vary = True
triolein_layer4.thick.bounds = (200, 500)
triolein4.real.vary = True
triolein4.real.bounds = (0.1, 6.36)
```

1.2 Model for the digested triolein system

The next model that we need is that for the digested triolein system. This model consists of four layers of the triolein with the SLD alowed to vary for each layer:

Silicon

SiO2 (15 Å)

Polystyrene (~100 Å)

Triolein layer 1 + solvent (~15 Å)

Triolein layer 2 + solvent (~90 Å)

Triolein layer 3 + solvent (~40 Å)

Triolein layer 4 + solvent (~400 Å)

D2O

The layers of triolein will include some volume fraction of D2O and the silicon and polystyrene layers will be the same layers as for the pure system (thereby constraining the model over two datasets).

Therefore, we define four new layers as shown below.

In [10]:
```
triolein_1 = SLD(1.4, name='Triolein_1') triolein_2
= SLD(5.4, name='Triolein_2') triolein_3 = SLD(3.6,
name='Triolein_3') triolein_4 = SLD(5.6,
name='Triolein_4') triolein_layer_1 =
triolein_1(15, 2)
triolein_layer_1.rough.setp(constraint=triolein_layer_1.thick * 0.1)
triolein_layer_2 = triolein_2(90, 2)
triolein_layer_2.rough.setp(constraint=triolein_layer_2.thick * 0.05)
triolein_layer_3 = triolein_3(40, 0)
```

```
triolein_layer_3.rough.setp(constraint=triolein_layer_3.thick * 0.1)
triolein_layer_4 = triolein_4(400, 10)
triolein_layer_4.rough.setp(constraint=triolein_layer_4.thick * 0.04)
```

The digested system is then constructed from the appropriate layers and the model is created.

In [11]:
```
dig_layers = [si_layer, SiO2_layer, polystyrene_layer, triolein_layer_1,
triolein_layer_2, triolein_layer_3, triolein_layer_4, d2o_layer]
```

In [12]:
```
dig_model = ReflectModel(Structure(dig_layers), bkg=1e-6,
dq=dig_data.x_err)
```

Then the
variable parameters for this model can be set and their bounds defined. Parameter, Minimum, Maximum, Thickness of triolein layer_1, 5, 30. SLD triolein layer_1, 0.1, 6.36. Thickness of triolein layer_2, 60, 120. SLD triolein layer_2, 0.1, 6.36. Thickness of triolein layer_3, 10, 50. SLD triolein layer_3, 0.1, 6.36. Thickness of triolein layer_4, 200, 500. SLD triolein layer_4, 0.1, 6.36.

In [13]:
```
triolein_layer_1.thick.vary = True
triolein_layer_1.thick.bounds = (5, 30)
triolein_1.real.vary = True triolein_1.real.bounds
= (0.1, 6.36) triolein_layer_2.thick.vary = True
triolein_layer_2.thick.bounds = (60, 120)
triolein_2.real.vary = True triolein_2.real.bounds
= (0.1, 6.36) triolein_layer_3.thick.vary = True
triolein_layer_3.thick.bounds = (10, 50)
triolein_3.real.vary = True triolein_3.real.bounds
= (0.1, 6.36) triolein_layer_4.thick.vary = True
triolein_layer_4.thick.bounds = (200, 500)
triolein_4.real.vary = True triolein_4.real.bounds
= (0.1, 6.36)
```

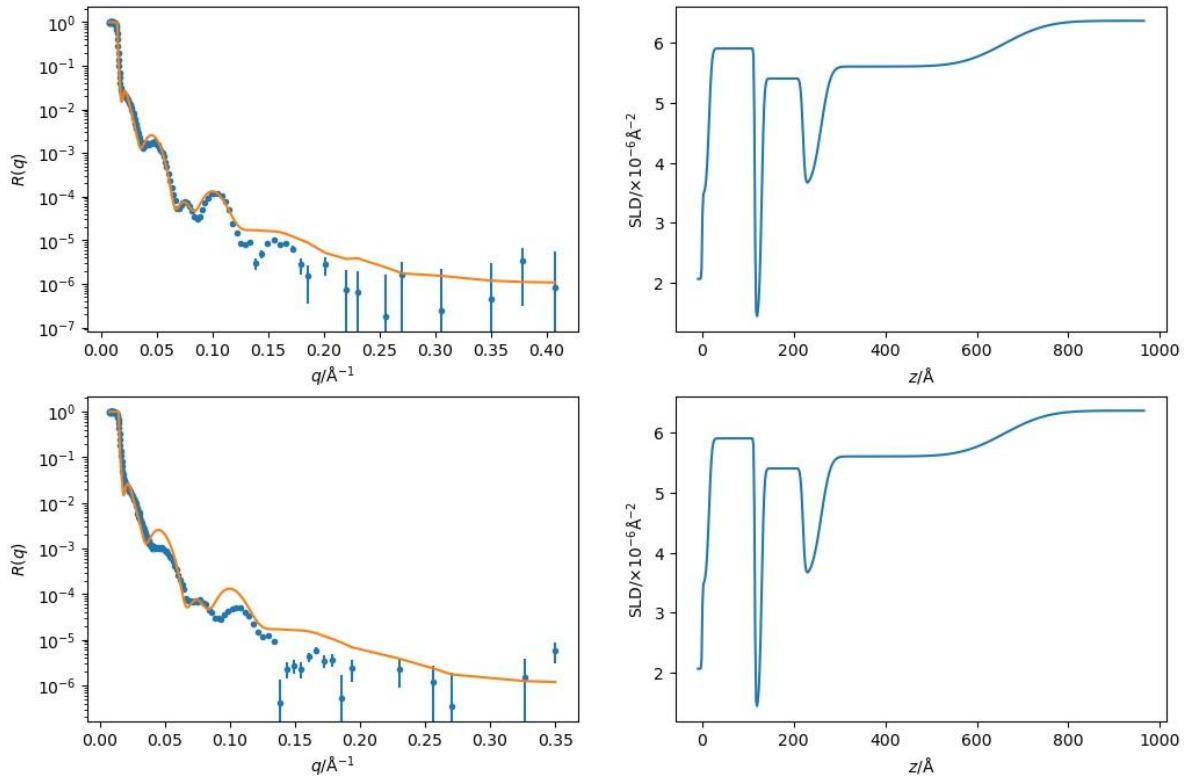
we can then check results for the initial guesses

In [14]:

```
fig, ax = plt.subplots(2, 2, figsize=(12, 8)) ax[0, 0].errorbar(pure_data.x,
pure_data.y, pure_data.y_err, marker='.', ls='') ax[0, 0].plot(pure_data.x,
pure_model(pure_data.x, x_err=pure_data.x_err), zorder= ax[0,
0].set_xlabel('$q$/Å$^{-1}$') ax[0, 0].set_ylabel('$R(q)$') ax[0,
0].set_yscale('log')

ax[0, 1].plot(*pure_model.structure.sld_profile()) ax[0,
1].set_xlabel('$z$/Å')
ax[0, 1].set_ylabel(r'SLD/$\times10^{-6}$Å$^{-2}$')

ax[1, 0].errorbar(dig_data.x, dig_data.y, dig_data.y_err, marker='.', ls='') ax[1,
0].plot(dig_data.x, dig_model(dig_data.x, x_err=dig_data.x_err), zorder=10) ax[1,
0].set_xlabel('$q$/Å$^{-1}$') ax[1, 0].set_ylabel('$R(q)$') ax[1,
0].set_yscale('log')
ax[1, 1].plot(*dig_model.structure.sld_profile()) ax[1,
1].set_xlabel('$z$/Å')
ax[1, 1].set_ylabel(r'SLD/$\times10^{-6}$Å$^{-2}$')
```

Out[14]:

```
Text(0, 0.5, 'SLD/$\\times10^{-6}$Å$^{-2}$')
```



1.3 Optimisation of the models

The objective functions, which bring together the data and the model, are then created.

These two functions are combined with the GlobalObjective.

In [15]:
```
pure_objective = Objective(pure_model, pure_data, transform=Transform('logY'))
dig_objective = Objective(dig_model, dig_data, transform=Transform('logY'))
```

In [16]:
```
objective = GlobalObjective(objectives=[pure_objective, dig_objective])
```

Finally, we perform the optimisation with a least squares algorithm.

In [17]:
```
fitter = CurveFitter(objective, nwalkers=128) fitter.fit('least_squares')
```

Out[17]:
```
     message: `ftol` termination condition is satisfied.
     success: True       status: 2
         fun: [-8.371e-01  1.080e+00 ...  1.255e-01  3.451e+00]
           x: [ 9.840e+01  1.437e+01 ...  2.886e+02  5.212e+00]        cost:
36204.78680522492             jac: [[ 1.105e-08  1.006e-07 ...
0.000e+00  0.000e+00]                 [ 0.000e+00  0.000e+00 ...
0.000e+00  0.000e+00]
                ...
                [ 1.935e-03  0.000e+00 ...  2.027e-07 -3.256e-06]
                [-8.339e-03  0.000e+00 ... -1.032e-06  1.150e-05]]
        grad:  [  1.767e-02    6.976e-03  ...    1.033e-02    1.745e-02]
        optimality:  0.9150212712167116    active_mask:  [0  0  ...  0  0]
        nfev: 22         njev: 22
       covar: [[ 5.919e-02 -1.340e-01 ...  3.926e-02  2.413e-04]
        [-1.340e-01  3.923e-01 ... -8.885e-02 -5.460e-04]
        ...
                [ 3.926e-02 -8.885e-02 ...  9.658e-01  2.687e-03]
                [ 2.413e-04 -5.460e-04 ...  2.687e-03  1.740e-05]]
stderr: [ 2.433e-01  6.263e-01 ...  9.827e-01  4.171e-03] we can then check
the goodness of fit (chi squared value) and plot the results In [18]:
objective.chisqr()
```

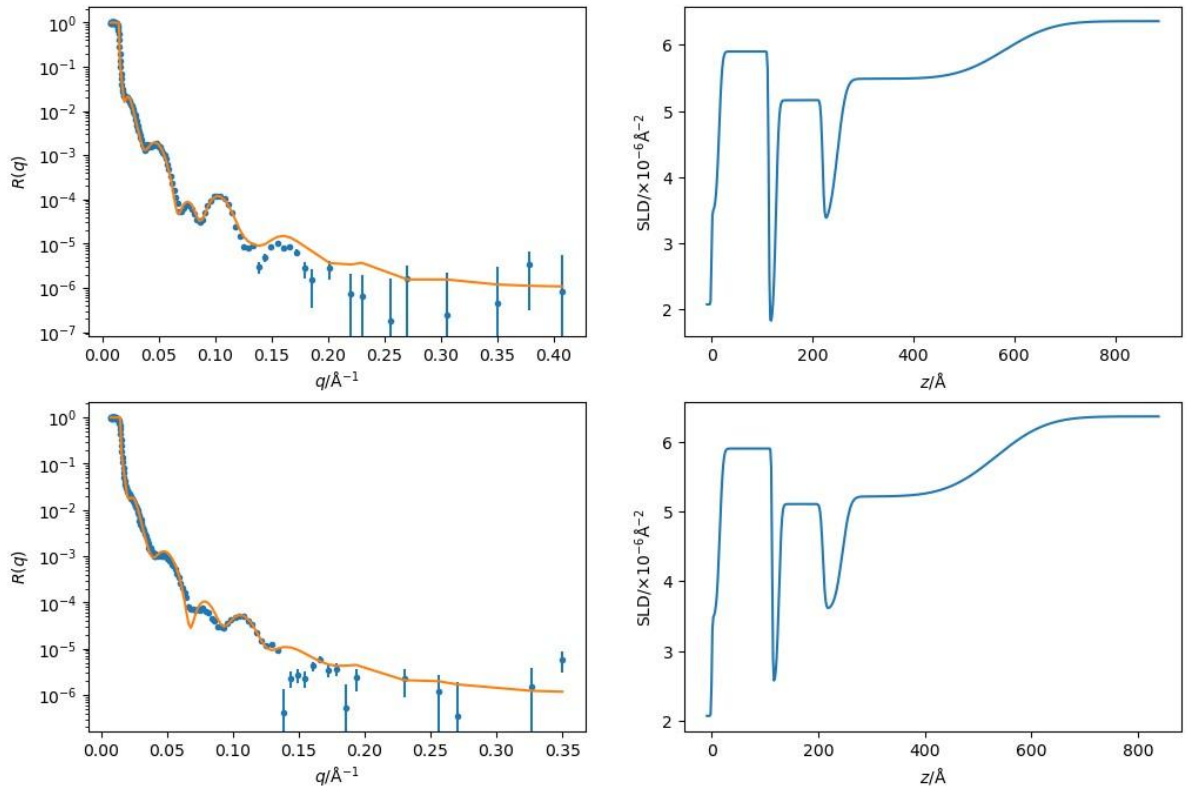
Out[18]:
```
72409.57361044985
```

In [19]:

```
fig, ax = plt.subplots(2, 2, figsize=(12, 8))
ax[0, 0].errorbar(pure_data.x, pure_data.y, pure_data.y_err, marker='.', ls='')
ax[0, 0].plot(pure_data.x, pure_model(pure_data.x, x_err=pure_data.x_err), zorder=
ax[0, 0].set_xlabel('$q$/Å$^{-1}$') ax[0, 0].set_ylabel('$R(q)$') ax[0,
0].set_yscale('log')
#for s in pure_model.structures:
#    ax[0, 1].plot(*s.sld_profile())
ax[0, 1].plot(*pure_model.structure.sld_profile()) ax[0,
1].set_xlabel('$z$/Å')
ax[0, 1].set_ylabel(r'SLD/$\times10^{-6}$Å$^{-2}$')

ax[1, 0].errorbar(dig_data.x, dig_data.y, dig_data.y_err, marker='.', ls='') ax[1,
0].plot(dig_data.x, dig_model(dig_data.x, x_err=dig_data.x_err), zorder=10) ax[1,
0].set_xlabel('$q$/Å$^{-1}$') ax[1, 0].set_ylabel('$R(q)$') ax[1,
0].set_yscale('log')
ax[1, 1].plot(*dig_model.structure.sld_profile()) ax[1,
1].set_xlabel('$z$/Å')
ax[1, 1].set_ylabel(r'SLD/$\times10^{-6}$Å$^{-2}$')
```

Out[19]:

```
Text(0, 0.5, 'SLD/$\\times10^{-6}$Å$^{-2}$')
```



In [20]:
```
print(pure_objective.varying_parameters())
```

```
_____________________________________________________________________________
Parameters:      None
<Parameter:'Polystyrene - thick', value=98.4023 +/- 0.243, bounds=[90.0, 110.0]>
```

```
<Parameter:'Triolein1 - thick', value=14.3742 +/- 0.626, bounds=[5.0, 30.0]>
<Parameter:'Triolein1 - sld', value=1.75214 +/- 0.146, bounds=[0.1, 6.36]>
<Parameter:'Triolein2 - thick', value=92.3809 +/- 0.401, bounds=[60.0, 120.0]>
<Parameter:'Triolein2 - sld', value=5.16197 +/- 0.00824, bounds=[0.1, 6.36]>
<Parameter:'Triolein3 - thick', value=29.5931 +/- 0.61 , bounds=[10.0, 50.0]>
<Parameter:'Triolein3 - sld', value=3.26512 +/- 0.0363, bounds=[0.1, 6.36]>
<Parameter:'Triolein4 - thick', value=331.4 +/- 1.14 , bounds=[200.0, 500.0]>
<Parameter:'Triolein4 - sld', value=5.48745 +/- 0.00332, bounds=[0.1, 6.36]>
```

```
In [21]: print(dig_objective.varying_parameters())
```

```
_____________________________________________________________________________
Parameters:      None
<Parameter:'Polystyrene - thick', value=98.4023 +/- 0.243, bounds=[90.0, 110.0]>
<Parameter:'Triolein_1 - thick', value=12.7317 +/- 0.691, bounds=[5.0, 30.0]>
<Parameter:'Triolein_1 - sld', value=2.53195 +/- 0.143, bounds=[0.1, 6.36]>
<Parameter:'Triolein_2 - thick', value=82.4665 +/- 0.528, bounds=[60.0, 120.0]>
<Parameter:'Triolein_2 - sld', value=5.10493 +/- 0.00977, bounds=[0.1, 6.36]>
<Parameter:'Triolein_3 - thick', value=36.5208 +/- 0.558, bounds=[10.0, 50.0]>
<Parameter:'Triolein_3 - sld', value=3.59302 +/- 0.0188, bounds=[0.1, 6.36]>
<Parameter:'Triolein_4 - thick', value=288.567 +/- 0.983, bounds=[200.0, 500.0]>
<Parameter:'Triolein_4 - sld', value=5.21231 +/- 0.00417, bounds=[0.1, 6.36]>
```

followed by optimisation with a differential evolution algorithm.

```
In [22]: fitter = CurveFitter(objective, nwalkers=128) fitter.fit('differential_evolution')
```

```
85it [04:26,  3.13s/it]
 message: Optimization terminated
successfully. Out[22]:  success: True      fun:
35488.52158289981
       x: [ 9.810e+01  1.378e+01 ...  2.812e+02  5.197e+00]
nit: 85     nfev: 36942    covar: [[ 5.682e-02 -1.345e-01 ...
4.106e-02  2.855e-04]           [-1.345e-01  4.145e-01 ... -
9.718e-02 -6.757e-04]
           ...
           [ 4.106e-02 -9.718e-02 ...  9.301e-01  2.497e-03]
           [ 2.855e-04 -6.757e-04 ...  2.497e-03  1.693e-05]]
stderr: [ 2.384e-01  6.438e-01 ...  9.644e-01  4.115e-03] check
```

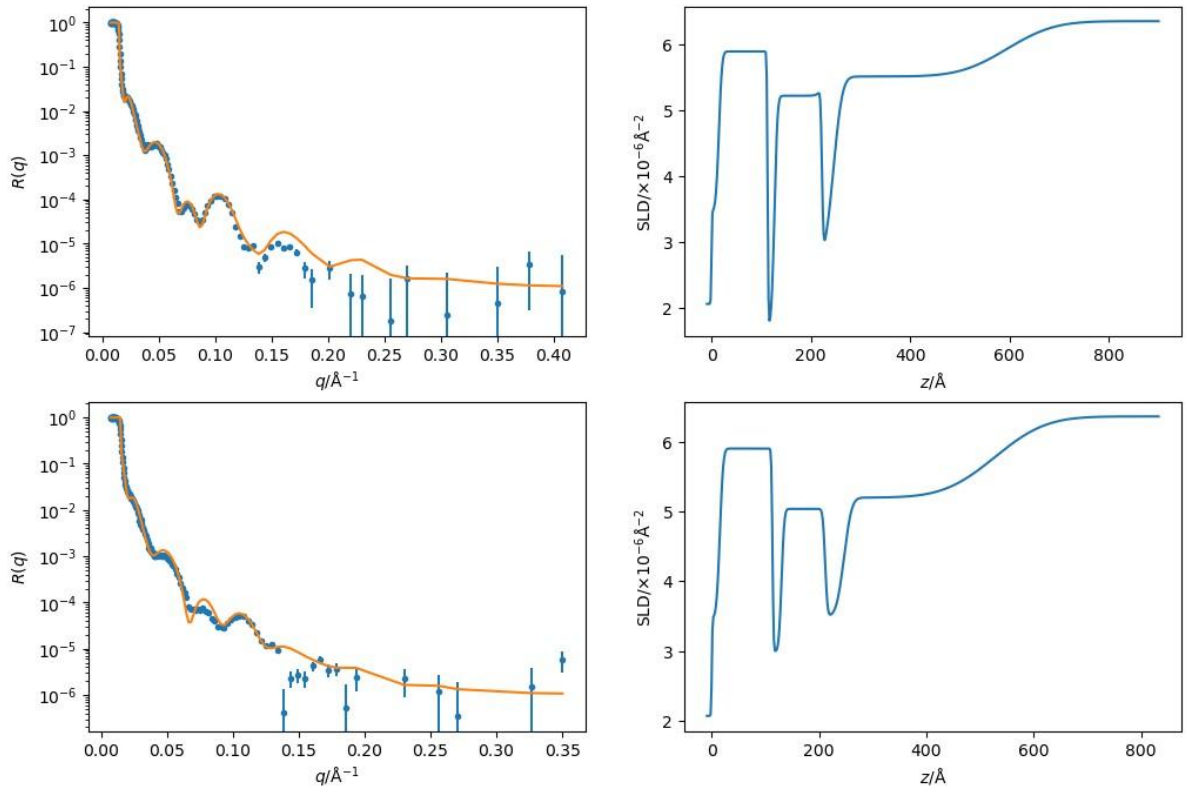
the goodness of fit (chi squared value) and plot the results again

```
In [23]: objective.chisqr()
Out[23]: 72975.45635881409
In [24]:
```

```
fig, ax = plt.subplots(2, 2, figsize=(12, 8))
ax[0, 0].errorbar(pure_data.x, pure_data.y, pure_data.y_err, marker='.', ls='')
ax[0, 0].plot(pure_data.x, pure_model(pure_data.x, x_err=pure_data.x_err), zorder=
ax[0, 0].set_xlabel('$q$/Å$^{-1}$') ax[0, 0].set_ylabel('$R(q)$') ax[0,
0].set_yscale('log')

ax[0, 1].plot(*pure_model.structure.sld_profile()) ax[0,
1].set_xlabel('$z$/Å')
ax[0, 1].set_ylabel(r'SLD/$\times10^{-6}$Å$^{-2}$')

ax[1, 0].errorbar(dig_data.x, dig_data.y, dig_data.y_err, marker='.', ls='') ax[1,
0].plot(dig_data.x, dig_model(dig_data.x, x_err=dig_data.x_err), zorder=10) ax[1,
0].set_xlabel('$q$/Å$^{-1}$') ax[1, 0].set_ylabel('$R(q)$') ax[1,
0].set_yscale('log') ax[1, 1].plot(*dig_model.structure.sld_profile()) ax[1,
1].set_xlabel('$z$/Å')
ax[1, 1].set_ylabel(r'SLD/$\times10^{-6}$Å$^{-2}$')
```

Out[24]:

```
Text(0, 0.5, 'SLD/$\\times10^{-6}$Å$^{-2}$')
```



The fitted parameters for each model are printed below.

In [25]:
```
print(pure_objective.varying_parameters())
```

```
_____________________________________________________________________________
Parameters:      None
```

```
<Parameter:'Polystyrene - thick', value=98.0951 +/- 0.238, bounds=[90.0, 110.0]>
<Parameter:'Triolein1 - thick', value=13.7847 +/- 0.644, bounds=[5.0, 30.0]>
<Parameter:'Triolein1 - sld', value=1.73958 +/- 0.161, bounds=[0.1, 6.36]>
<Parameter:'Triolein2 - thick', value=95.602 +/- 0.401, bounds=[60.0, 120.0]>
<Parameter:'Triolein2 - sld', value=5.22821 +/- 0.00792, bounds=[0.1, 6.36]>
<Parameter:'Triolein3 - thick', value=23.1772 +/- 0.618, bounds=[10.0, 50.0]>
<Parameter:'Triolein3 - sld', value=2.72929 +/- 0.0626, bounds=[0.1, 6.36]>
<Parameter:'Triolein4 - thick', value=349.395  +/- 1.2 , bounds=[200.0, 500.0]>
<Parameter:'Triolein4 - sld', value=5.52011 +/- 0.00332, bounds=[0.1, 6.36]>
```

In [26]:
```
print(dig_objective.varying_parameters())
```

```
________________________________________________________________________________
Parameters:      None
<Parameter:'Polystyrene - thick', value=98.0951 +/- 0.238, bounds=[90.0, 110.0]>
<Parameter:'Triolein_1 - thick', value=16.8849 +/- 0.642, bounds=[5.0, 30.0]>
<Parameter:'Triolein_1 - sld', value=2.99186 +/- 0.0799, bounds=[0.1, 6.36]>
<Parameter:'Triolein_2 - thick', value=81.2873 +/- 0.464, bounds=[60.0, 120.0]>
<Parameter:'Triolein_2 - sld', value=5.03518 +/- 0.00979, bounds=[0.1, 6.36]>
<Parameter:'Triolein_3 - thick', value=35.1871 +/- 0.585, bounds=[10.0, 50.0]>
<Parameter:'Triolein_3 - sld', value=3.49519 +/- 0.0217, bounds=[0.1, 6.36]>
<Parameter:'Triolein_4 - thick', value=281.236 +/- 0.964, bounds=[200.0, 500.0]>
<Parameter:'Triolein_4 - sld', value=5.19695 +/- 0.00411, bounds=[0.1, 6.36]> We
```

can then plot the fitted models against the data to assess the agreement.

To assess the correlation between the different parameters, Markov chain Monte Carlo sampling is used.

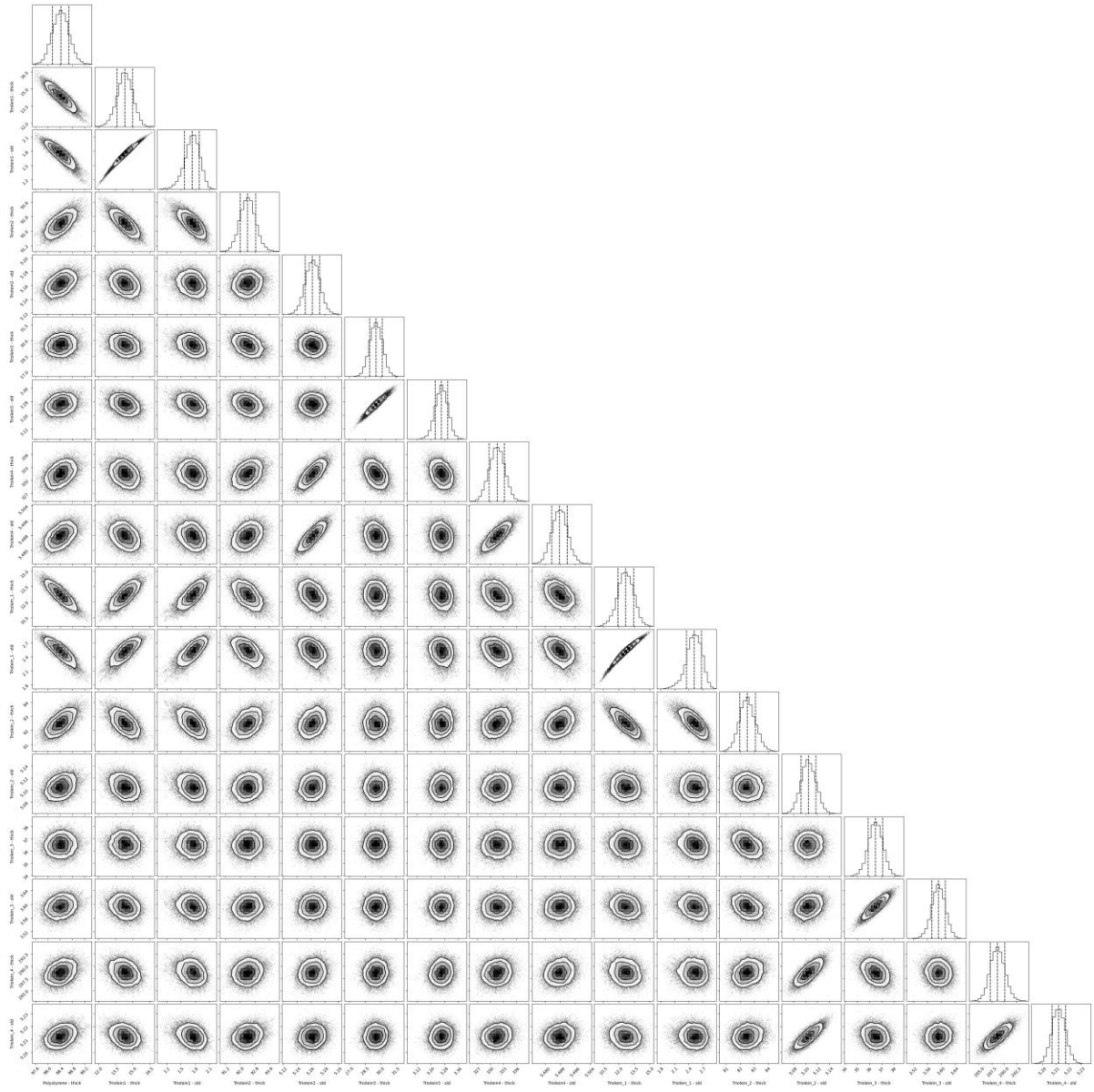
In [27]:
```
fitter.sample(1000)
fitter.reset()
```

```
res = fitter.sample(200, nthin=10)
```

```
100%|████████████████████████████████████████████████████████████████████████
| 1000/1000 [09:01<00:00,  1.85it/s]
100%|████████████████████████████████████████████████████████████████████████
| 2000/2000 [29:29<00:00,  1.13it/s]
```

We visualise this with a corner plot.

In [28]:
```
objective.corner()
plt.show()
```



In [29]:
```
objective.chisqr()
```
Out[29]: 72409.62144748759

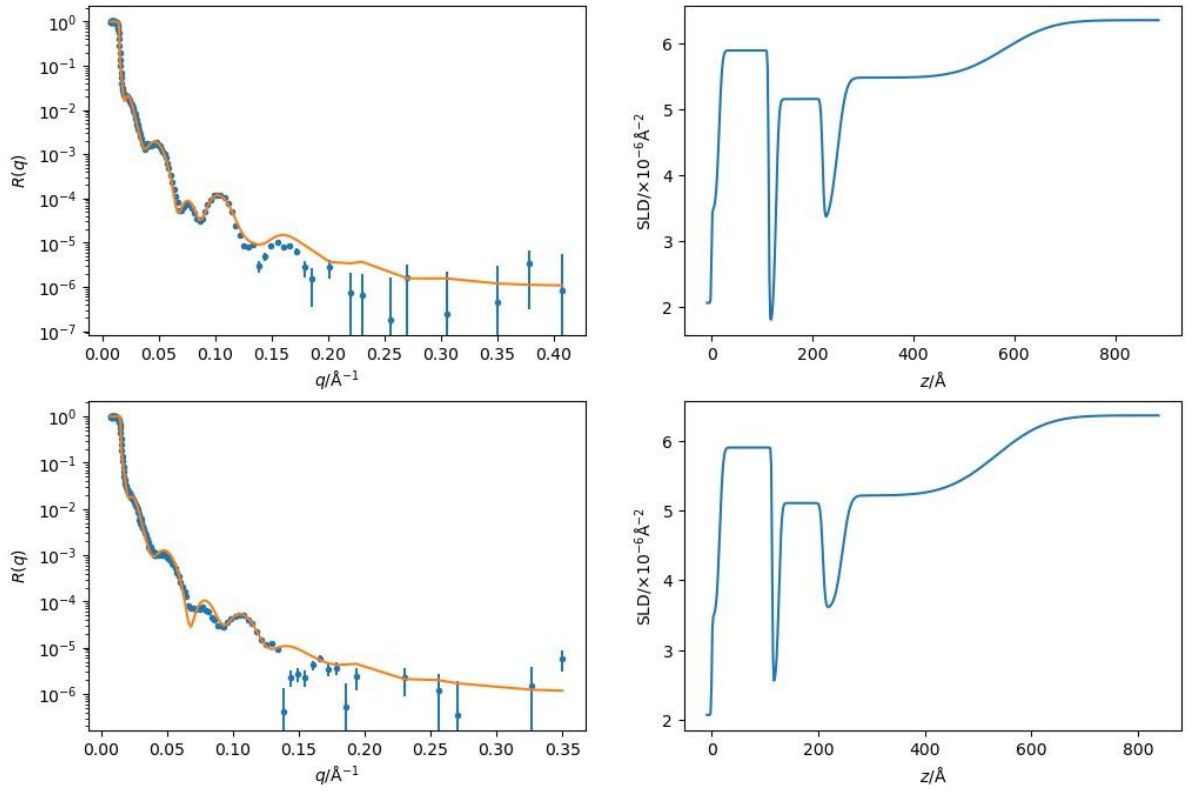
Then replot the data

In [30]:
```
fig, ax = plt.subplots(2, 2, figsize=(12, 8))
ax[0, 0].errorbar(pure_data.x, pure_data.y, pure_data.y_err, marker='.', ls='')
ax[0, 0].plot(pure_data.x, pure_model(pure_data.x, x_err=pure_data.x_err), zorder=1
ax[0, 0].set_xlabel('$q$/Å$^{-1}$') ax[0, 0].set_ylabel('$R(q)$') ax[0,
0].set_yscale('log')
```

```
ax[0, 1].plot(*pure_model.structure.sld_profile()) ax[0,
1].set_xlabel('$z$/Å')
ax[0, 1].set_ylabel(r'SLD/$\times10^{-6}$Å$^{-2}$')

ax[1, 0].errorbar(dig_data.x, dig_data.y, dig_data.y_err, marker='.', ls='') ax[1,
0].plot(dig_data.x, dig_model(dig_data.x, x_err=dig_data.x_err), zorder=10) ax[1,
0].set_xlabel('$q$/Å$^{-1}$') ax[1, 0].set_ylabel('$R(q)$') ax[1,
0].set_yscale('log')
ax[1, 1].plot(*dig_model.structure.sld_profile()) ax[1,
1].set_xlabel('$z$/Å')
ax[1, 1].set_ylabel(r'SLD/$\times10^{-6}$Å$^{-2}$')
```

```
Text(0, 0.5, 'SLD/$\\times10^{-6}$Å$^{-2}$')
```

Out[30]:



In [31]: `print(pure_objective.varying_parameters())`

```
_____________________________________________________________________________
Parameters:      None
<Parameter:'Polystyrene - thick', value=98.4265 +/- 0.272, bounds=[90.0, 110.0]>
<Parameter:'Triolein1 - thick', value=14.3239 +/- 0.684, bounds=[5.0, 30.0]>
<Parameter:'Triolein1 - sld', value=1.74193 +/- 0.159, bounds=[0.1, 6.36]>
<Parameter:'Triolein2 - thick', value=92.4154 +/- 0.428, bounds=[60.0, 120.0]>
<Parameter:'Triolein2 - sld', value=5.16252 +/- 0.0103, bounds=[0.1, 6.36]>
<Parameter:'Triolein3 - thick', value=29.5561 +/- 0.616, bounds=[10.0, 50.0]>
<Parameter:'Triolein3 - sld', value=3.26317 +/- 0.0365, bounds=[0.1, 6.36]>
<Parameter:'Triolein4 - thick', value=331.516 +/- 1.79 , bounds=[200.0, 500.0]>
```

```
<Parameter:'Triolein4 - sld', value=5.48767 +/- 0.00419, bounds=[0.1, 6.36]>
```

In [32]:
```
print(dig_objective.varying_parameters())
```

```
Parameters:      None
<Parameter:'Polystyrene - thick', value=98.4265 +/- 0.272, bounds=[90.0, 110.0]>
<Parameter:'Triolein_1 - thick', value=12.6445 +/- 0.765, bounds=[5.0, 30.0]>
<Parameter:'Triolein_1 - sld', value=2.51381 +/- 0.16 , bounds=[0.1, 6.36]>
<Parameter:'Triolein_2 - thick', value=82.5155 +/- 0.564, bounds=[60.0, 120.0]>
<Parameter:'Triolein_2 - sld', value=5.10486 +/- 0.0123, bounds=[0.1, 6.36]>
<Parameter:'Triolein_3 - thick', value=36.4995 +/- 0.588, bounds=[10.0, 50.0]>
<Parameter:'Triolein_3 - sld', value=3.5925 +/- 0.0197, bounds=[0.1, 6.36]>
<Parameter:'Triolein_4 - thick', value=288.646 +/- 1.47 , bounds=[200.0, 500.0]>
<Parameter:'Triolein_4 - sld', value=5.21239 +/- 0.00517, bounds=[0.1, 6.36]>
```

In [35]:
```
with open('C:/Users/humphreys/My Drive/refnx/2021 ILL Figaro TLL
                                        experiment/CSV/pH8     np.savetxt(fh,
                  np.array(pure_objective.model.structure.sld_profile()).T, delimi

with open('C:/Users/humphreys/My Drive/refnx/2021 ILL Figaro TLL experiment/CSV/pHp
save_arr_pure = np.array([pure_data.x, pure_data.y, pure_data.y_err, pure_model
np.savetxt(fh, save_arr_pure.T, delimiter=',', header='Q, measured R, measured
```

In [36]:
```
with open('C:/Users/humphreys/My Drive/refnx/2021 ILL Figaro TLL
                                        experiment/CSV/pH8     np.savetxt(fh,
                  np.array(dig_objective.model.structure.sld_profile()).T, delimit

with open('C:/Users/humphreys/My Drive/refnx/2021 ILL Figaro TLL experiment/CSV/pH8
save_arr_dig = np.array([dig_data.x, dig_data.y, dig_data.y_err, dig_model(dig_
np.savetxt(fh, save_arr_dig.T, delimiter=',', header='Q, measured R, measured R
```

In [ ]:
```
```