A comparison of different Fourier transform procedures for analysis of diffraction data from noble gas fluids

Supplementary information

J.E. Proctor[1], C.G. Pruteanu[2], B. Moss[1], M.A. Kuzovnikov[2], G.J. Ackland[2], C.W. Monk[1] and S. Anzellini[3]

1. Materials and Physics Research Group, School of Science, Engineering and Environment, University of Salford, Manchester M5 4WT, UK

2. SUPA, School of Physics & Astronomy and Centre for Science at Extreme Conditions, the University of Edinburgh, Edinburgh EH9 3FD, UK

3. Diamond Light Source Ltd., Harwell Science and Innovation Campus, Diamond House, Didcot OX11 0DE, UK

1. Dataset A real-space structure determination

We performed real-space structure determination on Dataset A using the Dissolve software package [19], producing a $g(r)$ in agreement with that obtained from standard FT of the dataset for the full range in Q-values. The simulation employed a 5000 atom simulation box and the OPLS-Noble Gases forcefield potential. It was not necessary in this case to refine the potential. The structure was sampled over 5000 accumulations. Figure S1 shows both $g(r)$ functions.

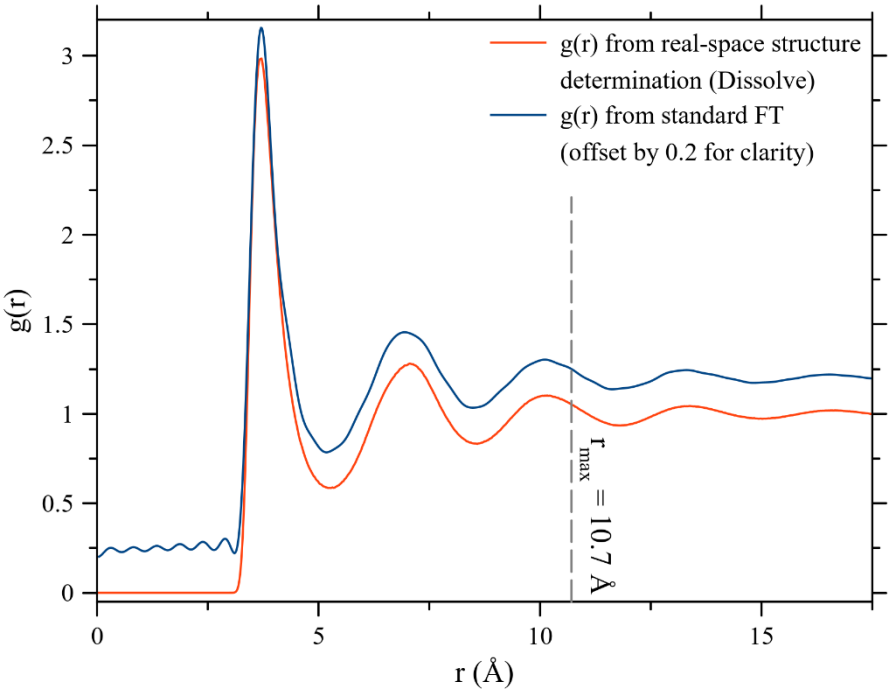

Figure S1. The $g(r)$ functions for Dataset A obtained via standard Fourier transform using the full range of Q-values for $S(Q)$, and via real-space structure determination using the Dissolve package.

2. Dataset B experimental methods

Pressure was applied using a custom-constructed piston-cylinder diamond anvil cell (DAC) equipped with 600 µm diameter culets and a $2\theta = 20°$ opening on the cylinder (downstream) side. This opening corresponds to $Q = 5.15$ Å^{-1}. Data collected at higher Q than this were not utilized in our analysis, therefore it was not necessary to collect / compute a transmission function for the DAC seats as was done in ref. [2]. An indented stainless steel gasket was utilized, in which a hole was prepared using a custom-constructed spark eroder device. Argon (BOC zero grade, 99.999%) was loaded cryogenically by placing the DAC in a small chamber into which Argon was pumped after purging the chamber of air. The chamber was placed in a bath of liquid nitrogen in order to condense the Argon, and the screws were turned to close the DAC whilst it was completely immersed in liquid Argon.

Synchrotron X-ray diffraction data were collected at Diamond Light Source beamline I15 using a 29.3 keV X-ray beam. The beam was focussed to 9 µm x 6 µm (FWHM) and a Pilatus CdTe 2M detector was used. The sample-to-detector distance (approx. 424 mm) and beam energy were calibrated using a CeO_2 standard. Typical data acquisition time was 30 s. Azimuthal integration was performed using Dioptas v0.5.5 software. Detector artefacts, Bragg peaks from diamond anvils and shadows from beamstop and other auxiliary equipment were manually masked, and all detector images were processed with the same mask to enable a subtraction of individual patterns. "The X-ray beam was aligned to the centre of the sample chamber before collecting each pattern".

Pressure was measured using the Ruby photoluminescence method. Due to the need to collect many closely spaced datapoints, pressure was measured before and after each X-ray diffraction pattern was collected and data were rejected if the change in pressure during data collection exceeded 30 MPa.

The experiment on I15 in which these data were collected was setup at short notice following an altnernate experiment being aborte for technical reasons. As a result, the I15 beamline was not setup with the optimum X-ray wavelength, geometry and detector for this experiment.

3. Dataset B normalization

In pressure points comprising dataset B $I_{raw}(Q)$, the raw coherent scattering intensity from the sample, was obtained from the total raw scattering intensity by subtraction of a background originating from either the empty DAC or the DAC containing Ar in the solid state. Figure S2 below shows examples of both these calculations.

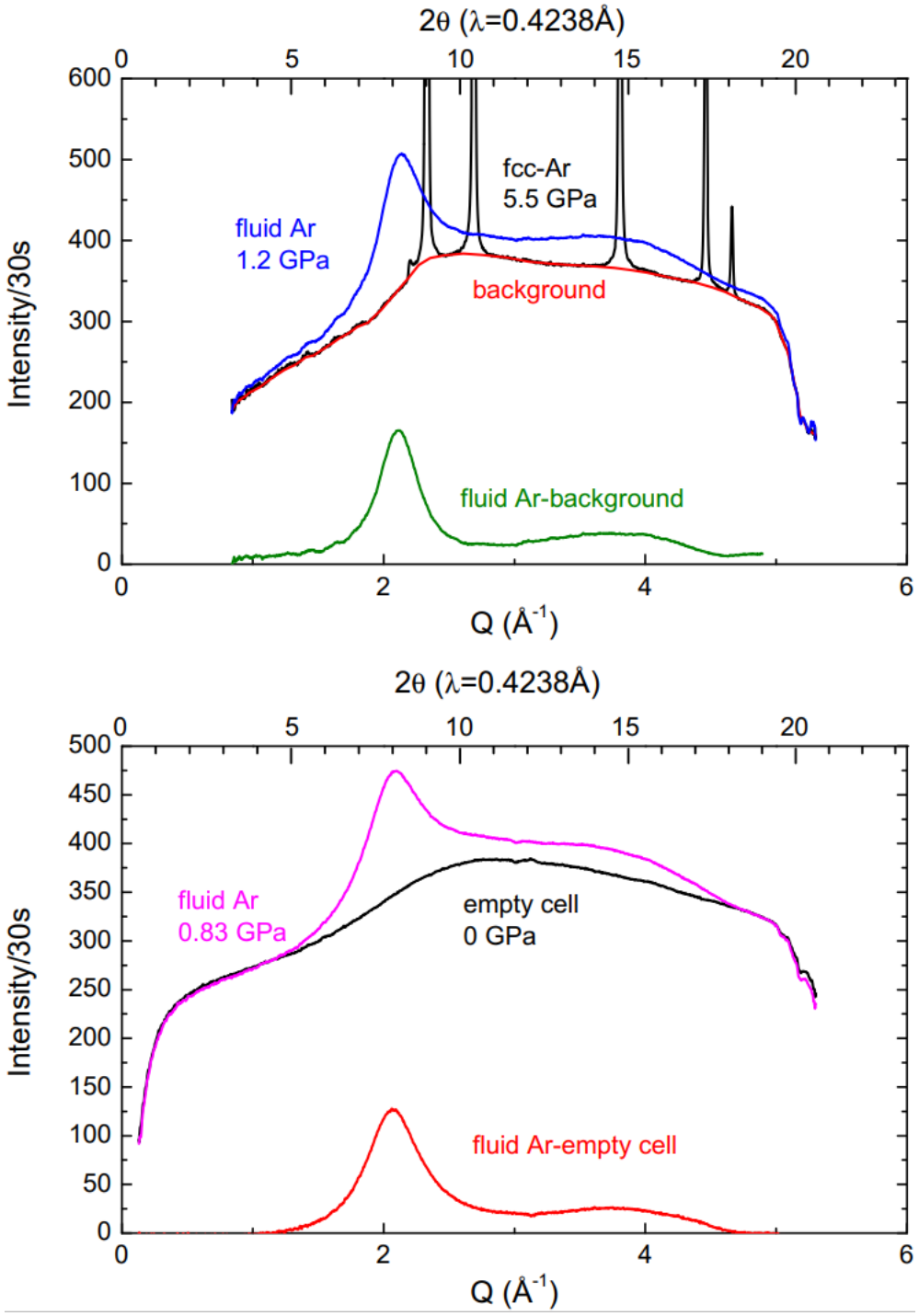

Figure S2. Examples of background subtraction on dataset B to obtain $I_{raw}(Q)$ by subtraction of the background signal from Ar confined in the DAC in the solid state (upper) and from the empty DAC (lower).

Dataset B was normalized according to equation 3, reproduced below:

$$S(Q) = \frac{I_{raw}(Q)}{N' f(Q)^2}$$

The parameter N' is a fitting parameter to ensure that $S(Q) \to 1$ in the high Q limit, whilst $f(Q)$ is the atomic form factor. The most commonly used set of values are those given in the International Tables for Crystallography [1]. We used the linear interpolations between the values given in the

International Tables for Crystallography shown in figure S3. In practice, performing background subtraction that produced reasonable outcomes at all Q without obtaining negative $I_{raw}(Q)$ for some low values of Q was challenging. The data were therefore shifted by a small constant (less than 5% of the peak value) to ensure that the lowest value of $S(Q)$ was at least zero.

Figure S4 shows, for example data at 830 MPa, the stages in the normalization process beginning with $I_{raw}(Q)$. Firstly, $I_{raw}(Q)$ is divided by $f(Q)^2$, after which it is normalized to 1 in the high-Q limit. Since the value of with $I_{raw}(Q)$ is still oscillating at $Q = 5$ Å^{-1}, this was done by normalizing such that the average value of the second peak (at $Q \approx 4$ Å^{-1}), and the trough following it, was 1.

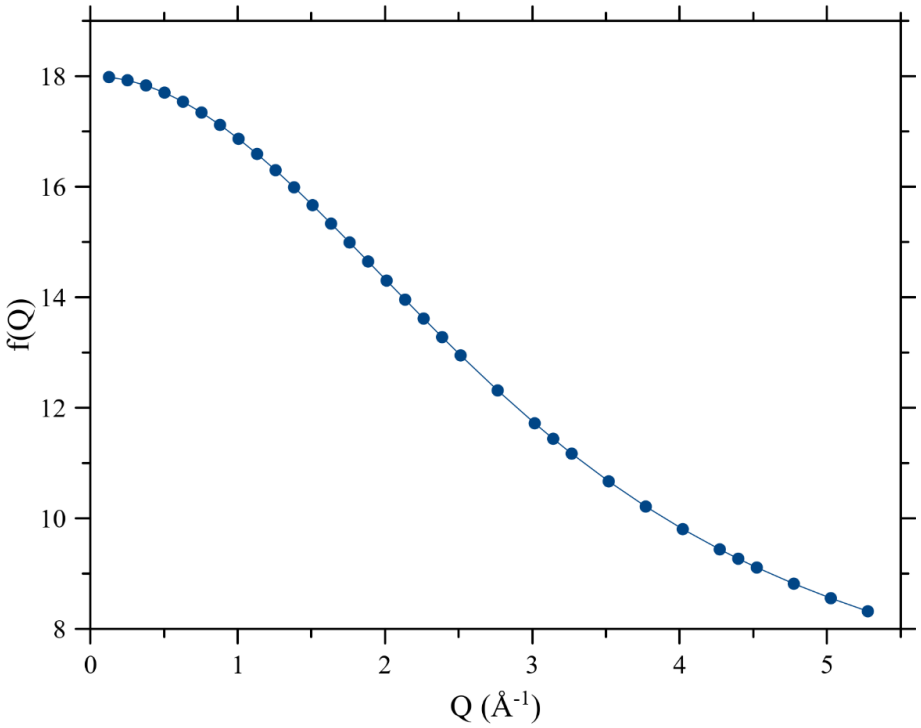

Figure S3. Tabulated f(Q) data from the Intl. Tables for Crystallography [1] (points) and our interpolation (line).

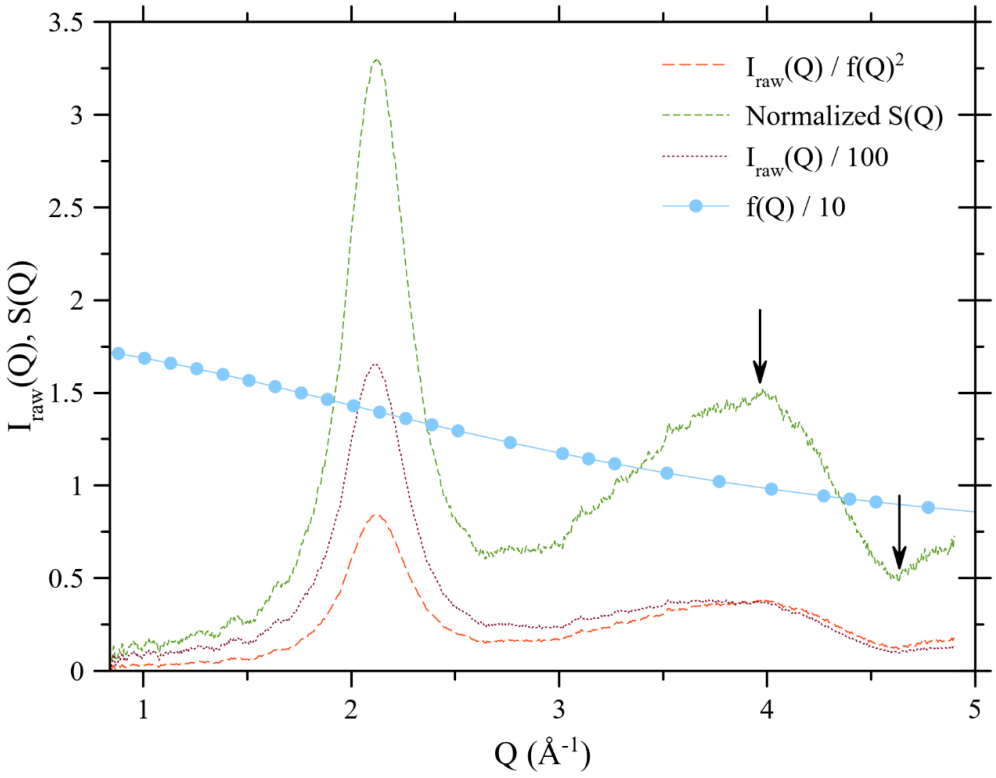

Figure S4. Stages in the normalization process to obtain $S(Q)$ from $I_{raw}(Q)$ at 830 MPa. The division by 100 applied to $I_{raw}(Q)$ and $(I_{raw}(Q)/f(Q)^2)$ was performed solely to enable presentation on this figure alongside $f(Q)$ and $S(Q)$, and was not part of the normalization process. The arrows indicate the points selected in the last stage of the normalization process: Ensuring that the average value of S(Q) between these points is 1.

4. Phase diagrams

Figure S5 (below) shows the phase diagrams of fluid Ar (from ref. [18]) and Kr (compiled for this work using the methodology presented in ref. [18]), with the P,T points marked at which the different datasets in the present work were collected.

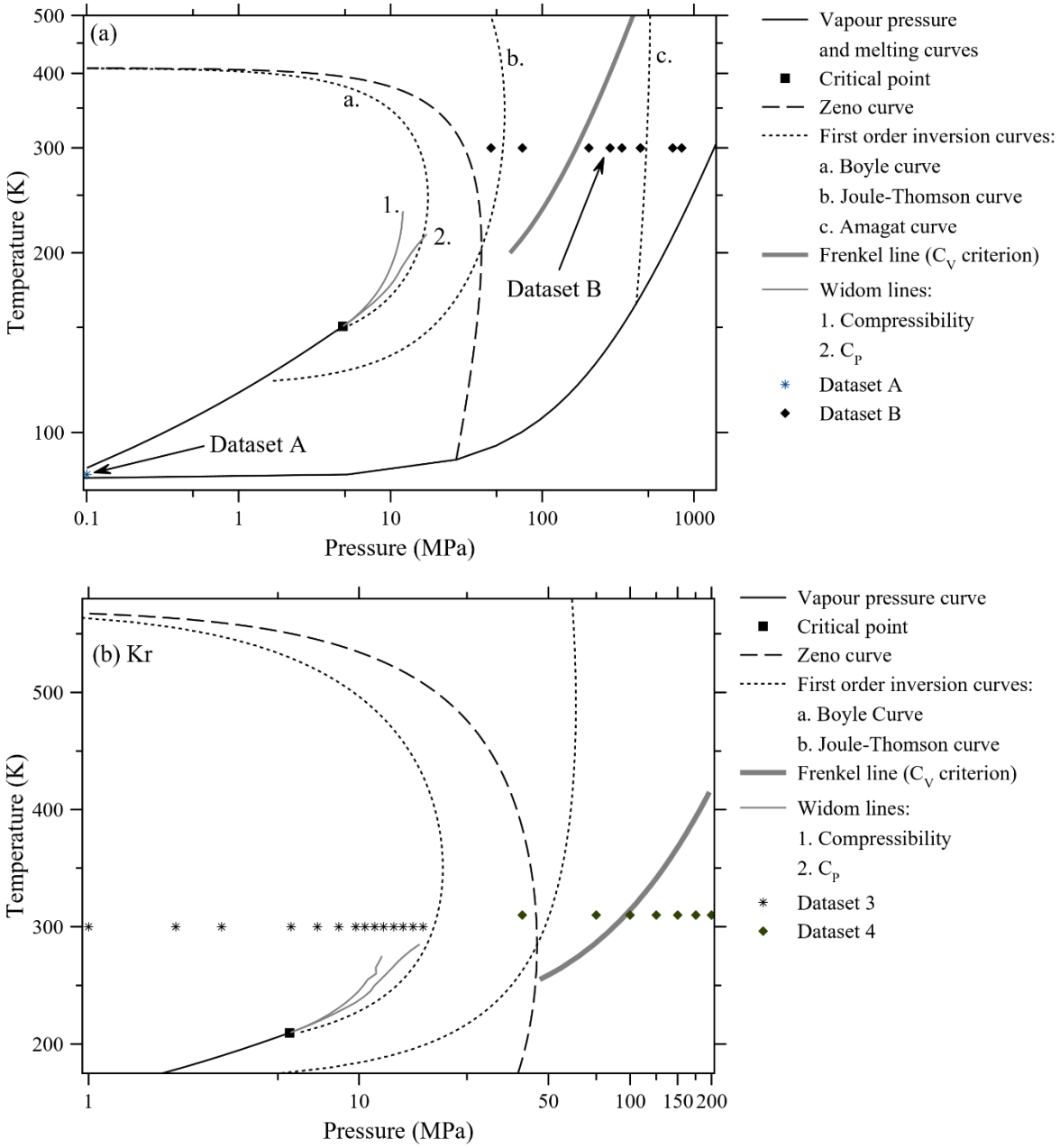

Figure S5. (a) Phase diagram of Ar (from ref. [18] with the P,T points for datasets A and B added) and panel (b) Phase diagram of Kr (compiled using the methodology of ref. [18] with the P,T points for datasets C and D added).

5. Code documentation

The Fourier transform (FT) results described in the text were obtained using our own code as documented below. We used Octave (an open-source version of Matlab) throughout. There are three separate scripts, one for the standard Fourier transform and one for each modification function. The code calculates a FT using the full range of the input data (all Q values) and a FT using Q values only up to a certain lower cutoff. The scripts for the modification functions are based closely on the script for the standard Fourier transform so we will document this first. All three scripts are available to download with this supplementary information.

5.1 Standard Fourier transform

The $S(Q)$ input data should be provided in tab separated variable format where the first column is filled with the Q-values (in Å^{-1}) and the second column with the $S(Q)$ values. All text such as column headings must be deleted from the input data file by the user prior to running the script. The data should be stored in the same directory as the script, in which case to load the data it is simply necessary to write the file name including extension in line 4 and the file name without the extension in line 5.

The following additional data need to be entered by the user:

- Line 6, the number density of particles p in at./m^3.
- Line 7 Qmax, the row number in the $S(Q)$ input data file at which to stop for the FT using the cutoff at lower Q.
- Line 9 n, the number of (equally spaced) values of r at which to calculate $g(r)$.
- Line 10, r_max, the maximum value of r (in metres) to which to calculate $g(r)$.

The remaining variables are as follows:

Name	Purpose
Q_values	Array of Q values (read from file in Å^{-1}, converted immediately to m^{-1} when array is filled).
S_Q_values	Array of normalized $S(Q)$ values read from file.
r_values	The array of n values of r (in metres) for which $g(r)$ is to be calculated, starting at r_spacing (since equation 4 in the main text fails at $r = 0$) and ending at r_max.
r_spacing	the spacing between r values.
Q_values_cutoff	Equivalent of Q_values but containing data only up to Q_values(Qmax).
S_Q_values_cutoff	Equivalent of S_Q_values but containing data only up to S_Q_values(Qmax).
raw_g_r	Array of all (unnormalized) $[g(r) - g_0]$ values for the FT of the full dataset.
raw_g_r_cutoff	Equivalent to raw_g_r but for the FT of the data up to Q_values(Qmax).
norm_g_r	Normalized g(r) for full FT.
norm_g_r_cutoff	Ditto but for cutoff FT (same number of r values in both cases).
Q_value	Value of Q at the cutoff point.
CN	Calculated co-ordination number (CN) with full range $g(r)$.
CN_cutoff	Calculated co-ordination number (CN) with $g(r)$ obtained with $S(Q)$ cutoff at lower Q-value.

Table S1. Names of principal non-user-defined variables in the code for the standard Fourier transform.

The code obtains the normalized $g(r)$ and the co-ordination number (CN) for the full set of $S(Q)$ data and the data cutoff at the lower value of Q in the following steps. Firstly, the equation below (equation 4 from the main text rearranged and with upper and lower limits inserted in the integral) is used to calculate $[g(r) - g_0]$.

$$g(r) - g_0 = \frac{1}{2\pi^2 r^2} \int_{Q_{min}}^{Q_{max}} Qr[S(Q) - 1]sin(Qr)dQ$$

This is performed for each value of r using the trapz numerical integration function. The parameter Q_{min} is simply the minimum value of Q for which $S(Q)$ experimental data is provided.

The same procedure is used to normalize the $g(r)$ functions for both the full and reduced range in Q-values. A constant is added to the value of $g(r)$ at all r to ensure that $g(r_{min}) = 0$ (essentially just removing the arbitrary constant g_0). Then the $g(r)$ data at all r are divided by $g(r_{max})$. Clearly, this procedure will not work if the ripples in $g(r)$ at low r result in $g(r_{max}) < g(r_{min})$ so first the code checks for this and displays an error message if this is the case. Clearly it would be possible to contrive other, more complex, procedures for normalizing $g(r)$ in this case but we suggest that if this is required then the data are not worth the effort.

The final calculation performed is the further integration to obtain the CN for both normalized g(r) functions using the equation below:

$$\mathrm{CN} = 4\pi\rho \int_{r_1}^{r_2} r^2 g(r) dr$$

Mathematically, the CN calculation consists of integrating $r^2 g(r)$ from the minimum before the first peak (r_1) to the minimum after the first peak (r_2). The code locates the highest peak in $g(r)$, then locates the minima on each side and integrating using the trapz numerical integration function. In contrast to the theoretical definition of the CN (an integration starting from $r = 0$, equation 14 in the main text) the finite lower limit on r has to be included when analysing $g(r)$ originating from FT of real experimental data to avoid including the area of unphysical ripples at very low r in the calculation.

If the unphysical ripples in $g(r)$ caused by the FT process cause the first physically meaningful peak to not be the highest peak, then the procedure will fail. *It is the responsibility of the user to check this by viewing the graphs of the relevant functions produced by the code*.

The CNs calculated from both $g(r)$ functions are outputted in variables that the user can read in the Octave workspace, and the following outputs are saved to file:

- The raw $[g(r) - g_0]$ and normalized $g(r)$ functions obtained with the full $S(Q)$ dataset.
- The normalized g(r) function obtained with the $S(Q)$ dataset over a reduced range in Q.

The following figures are created:

- Figure 1. The $S(Q)$ that was provided, over the full range in Q.
- Figure 2. The raw $[g(r) - g_0]$ functions obtained from the FT of the full $S(Q)$ dataset and the dataset over a reduced range in Q.
- Figure 3. The normalized $g(r)$ functions obtained from the FT of the full $S(Q)$ dataset and the dataset over a reduced range in Q.

5.2 Fourier transform with Lorch modification function

In this case, when the Fourier transform is performed to obtain $[g(r_0) - g_0]$ equation 6 from the main text is utilized, but with a finite lower limit Q_{min} similarly to the direct FT described above. Q_{max} is the maximum value of Q for which S(Q) data are provided for the FT process, and Δ is obtained from Q_{max} using $\Delta = \pi/Q_{max}$ as justified in the main text.

5.3 Fourier transform with Soper-Barney modification function

In this case, when the Fourier transform is performed to obtain $[g(r_0) - g_0]$ equation 11 from the main text is utilized, but with a finite lower limit Q_{min} similarly to the direct FT described above. Q_{max} is the maximum value of Q for which $S(Q)$ data are provided for the FT process, and Δ is obtained from Q_{max} using $\Delta = 4.49/Q_{max}$ as justified in the main text.

6. Code

It should be possible to copy and paste this code directly into Octave or Matlab. Each code is a separate self-contained script. The code is also available on request in Octave (.m) format.

6.1 Standard Fourier transform

```
## STEP 1: LOAD S(Q) DATA ##

clear variables
load Ar_335977.txt;
data_file = Ar_335977;
p = 1.75132E+27; #number density of particles
Qmax = 50; #Q cutoff variables

Q_values = data_file(:,1)*10^10; #Q
S_Q_values = data_file(:,2); #S(Q)
n = 1000; #number of points to be calculated
r_max = 20*10^-10;
r_spacing = r_max/n;
r_values = r_spacing:r_spacing:r_max; #array of r values

Q_value = Q_values(Qmax)*10^-10; #Just for readout, the actual value of Q cutoff in A^-1
Q_values_cutoff = [];
S_Q_values_cutoff = [];

for i = 1:Qmax #input values into Q for cutoff integral
  Q_values_cutoff = [Q_values_cutoff;Q_values(i)];
  S_Q_values_cutoff = [S_Q_values_cutoff;S_Q_values(i)];
end

## STEP 2: FINDING G(R) - G0 ##
raw_g_r = []; #g(r) - g0 with full range
raw_g_r_cutoff = []; #g(r) - g0 with cut-off

for i = 1:length(r_values) #we integrate for all values of r
  #the full-range integral
  F = Q_values.*(S_Q_values-1).*sin(Q_values*r_values(i));
  g = trapz(Q_values,F)*2/pi; #integrate
  gr = g/(4*pi*r_values(i)); #the integral gives 4*pi*r*[g(r) - g0], so divide by 4*pi*r
  raw_g_r = [raw_g_r;gr];
  #the cut-off integral
```

```
  F = Q_values_cutoff.*(S_Q_values_cutoff-1).*sin(Q_values_cutoff*r_values(i));
  g = trapz(Q_values_cutoff,F)*2/pi; #integrate
  gr = g/(4*pi*r_values(i)); #the integral gives 4*pi*r*[g(r) - g0], so divide by 4*pi*r
  raw_g_r_cutoff = [raw_g_r_cutoff;gr];
end

## STEP 3: NORMALIZATION ##

norm_g_r = []; #normalized g(r) for full range
norm_g_r_cutoff = []; #normalized g(r) for cut-off

if (raw_g_r(1) > raw_g_r(length(r_values)))
  disp('Full range g(r) data cannot be normalized');
end

if (raw_g_r_cutoff(1) > raw_g_r_cutoff(length(r_values)))
  disp('Qmax cutoff g(r) data cannot be normalized');
end

for i = 1:length(r_values)
  norm_g_r = [norm_g_r;(raw_g_r(i) - raw_g_r(1))];
  norm_g_r_cutoff = [norm_g_r_cutoff;(raw_g_r_cutoff(i) - raw_g_r_cutoff(1))];
end

for i = 1:length(r_values)
  norm_g_r(i) = norm_g_r(i) / norm_g_r(length(r_values));
  norm_g_r_cutoff(i) = norm_g_r_cutoff(i) / norm_g_r_cutoff(length(r_values));
end

## STEP 4: CO-ORDINATION NUMBER ##

#CN: full range
int_table = [];
Max_Peak = 0;
max_peak_pos = 1; #Position of first Cshell max
coord_max = 1; #Upper limit for CN integration
coord_min = 1; #Lower limit for CN integration
func = norm_g_r;

for i = 2:length(r_values) #locate tallest peak
  if (func(i) > Max_Peak)
    Max_Peak = func(i);
    max_peak_pos = i;
  end
end

#see where minimum is after tallest peak
Lowest = func(max_peak_pos);

for i = (max_peak_pos+1):length(r_values)
  if (Lowest > func(i)) #i.e. gradient is negative
```

```
    Lowest = func(i);
    coord_max = i;
  else
    break
  end
end

#See where minimum is before tallest peak
for i = 2:(max_peak_pos-1)
    if (func(i-1) > func(i)) #So it sets coord_min if there is a negative gradient
      coord_min = i;
    endif
end

for i = coord_min:coord_max #Num should be at the point of the lowest peak
  int_table = [int_table, (r_values(i)^2).*norm_g_r(i)]; #get sets of data points for CN integral
end

area = trapz(r_values(coord_min:coord_max),int_table); #integrate
CN = 4*pi*p*area;

#CN: cutoff range
int_table = [];
Max_Peak = 0;
max_peak_pos = 1; #Position of first Cshell max
coord_max = 1; #Upper limit for CN integration
coord_min = 1; #Lower limit for CN integration
func = norm_g_r_cutoff;

for i = 2:length(r_values) #locate tallest peak
  if (func(i) > Max_Peak)
    Max_Peak = func(i);
    max_peak_pos = i;
  end
end

#see where minimum is after tallest peak
Lowest = func(max_peak_pos);

for i = (max_peak_pos+1):length(r_values)
  if (Lowest > func(i)) #i.e. gradient is negative
    Lowest = func(i);
    coord_max = i;
  else
    break
  end
end

#See where minimum is before tallest peak
for i = 2:(max_peak_pos-1)
    if (func(i-1) > func(i)) #So it sets coord_min if there is a negative gradient
```

```
    coord_min = i;
  endif
end

for i = coord_min:coord_max #Num should be at the point of the lowest peak
 int_table = [int_table, (r_values(i)^2).*norm_g_r_cutoff(i)]; #get sets of data points for CN integral
end

area = trapz(r_values(coord_min:coord_max),int_table); #integrate
CN_cutoff = 4*pi*p*area;

## EXPORTING AND GRAPHS ##
#Make a descending list of r values so we can save them in a text document
r_list = [];

for i = 1:length(r_values)
 r_list = [r_list;r_values(i)*10^10];
end

RDFdata = [r_list norm_g_r];
RDFdata2 = [r_list raw_g_r];
CutOffdata = [r_list norm_g_r_cutoff];
save Std_norm_RDF.txt RDFdata;
save Std_raw_RDF.txt RDFdata2;
save Std_norm_cutoff_RDF.txt CutOffdata;

figure(1)
plot(Q_values,S_Q_values)
xlabel('Q')
ylabel('S(Q)')
title('Structure Factor')
grid on
box on

figure(2)
plot(r_values,raw_g_r,'r',r_values,raw_g_r_cutoff,'b')
legend('Max Q','Q Cutoff')
xlabel('r')
ylabel('G(r) - G0')
title('Radial Distribution Function: Fourier transform')
grid on
box on

figure(3)
plot(r_values,norm_g_r,'r',r_values,norm_g_r_cutoff,'b')
legend('Max Q','Q Cutoff')
xlabel('r')
ylabel('g(r)')
title('Normalised g(r)')
grid on
box on
```

6.2 Fourier transform with Lorch modification

```
## STEP 1: LOAD S(Q) DATA ##

clear variables
load Ar_335953.txt;
data_file = Ar_335953;
p = 2.5924E28; #number density of particles
Qmax = 870; #Q cutoff variables

Q_values = data_file(:,1)*10^10; #Q
S_Q_values = data_file(:,2); #S(Q)
n = 1000; #number of points to be calculated
r_max = 20*10^-10;
r_spacing = r_max/n;
r_values = r_spacing:r_spacing:r_max; #array of r values

Q_value = Q_values(Qmax)*10^-10; #Just for readout, the actual value of Q cutoff in A^-1
Q_values_cutoff = [];
S_Q_values_cutoff = [];

delta = pi/Q_values(length(Q_values));
cutoff_delta = pi/Q_values(Qmax);

for i = 1:Qmax #input values into Q for cutoff integral
  Q_values_cutoff = [Q_values_cutoff;Q_values(i)];
  S_Q_values_cutoff = [S_Q_values_cutoff;S_Q_values(i)];
end

## STEP 2: FINDING G(R) - G0 ##
raw_g_r = []; #g(r) - g0 with full range
raw_g_r_cutoff = []; #g(r) - g0 with cut-off

for i = 1:length(r_values) #we integrate for all values of r
  #the full-range integral
  F = Q_values.*(S_Q_values-
1).*sin(Q_values.*r_values(i)).*(sin(Q_values.*delta)./(Q_values.*delta));
  g = trapz(Q_values,F)*2/pi; #integrate
  gr = g/(4*pi*r_values(i)); #the integral gives 4*pi*r*[g(r) - g0], so divide by 4*pi*r
  raw_g_r = [raw_g_r;gr];
  #the cut-off integral
  F = Q_values_cutoff.*(S_Q_values_cutoff-
1).*sin(Q_values_cutoff.*r_values(i)).*(sin(Q_values_cutoff.*cutoff_delta)./(Q_values_cutoff.*cutoff
_delta));
  g = trapz(Q_values_cutoff,F)*2/pi; #integrate
  gr = g/(4*pi*r_values(i)); #the integral gives 4*pi*r*[g(r) - g0], so divide by 4*pi*r
  raw_g_r_cutoff = [raw_g_r_cutoff;gr];
end
```

```
## STEP 3: NORMALIZATION ##

norm_g_r = []; #normalized g(r) for full range
norm_g_r_cutoff = []; #normalized g(r) for cut-off

if (raw_g_r(1) > raw_g_r(length(r_values)))
  disp('Full range g(r) data cannot be normalized');
end

if (raw_g_r_cutoff(1) > raw_g_r_cutoff(length(r_values)))
  disp('Qmax cutoff g(r) data cannot be normalized');
end

for i = 1:length(r_values)
  norm_g_r = [norm_g_r;(raw_g_r(i) - raw_g_r(1))];
  norm_g_r_cutoff = [norm_g_r_cutoff;(raw_g_r_cutoff(i) - raw_g_r_cutoff(1))];
end

for i = 1:length(r_values)
  norm_g_r(i) = norm_g_r(i) / norm_g_r(length(r_values));
  norm_g_r_cutoff(i) = norm_g_r_cutoff(i) / norm_g_r_cutoff(length(r_values));
end

## STEP 4: CO-ORDINATION NUMBER ##
#CN: full range
int_table = [];
Max_Peak = 0;
max_peak_pos = 1; #Position of first Cshell max
coord_max = 1; #Upper limit for CN integration
coord_min = 1; #Lower limit for CN integration
func = norm_g_r;

for i = 2:length(r_values) #locate tallest peak
  if (func(i) > Max_Peak)
    Max_Peak = func(i);
    max_peak_pos = i;
  end
end

#see where minimum is after tallest peak
Lowest = func(max_peak_pos);

for i = (max_peak_pos+1):length(r_values)
  if (Lowest > func(i)) #i.e. gradient is negative
    Lowest = func(i);
    coord_max = i;
  else
    break
  end
end
```

```
#See where minimum is before tallest peak
for i = 2:(max_peak_pos-1)
   if (func(i-1) > func(i)) #So it sets coord_min if there is a negative gradient
     coord_min = i;
   endif
end

for i = coord_min:coord_max #Num should be at the point of the lowest peak
  int_table = [int_table, (r_values(i)^2).*norm_g_r(i)]; #get sets of data points for CN integral
end

area = trapz(r_values(coord_min:coord_max),int_table); #integrate
CN = 4*pi*p*area;

#CN: cutoff range
int_table = [];
Max_Peak = 0;
max_peak_pos = 1; #Position of first Cshell max
coord_max = 1; #Upper limit for CN integration
coord_min = 1; #Lower limit for CN integration
func = norm_g_r_cutoff;

for i = 2:length(r_values) #locate tallest peak
  if (func(i) > Max_Peak)
    Max_Peak = func(i);
    max_peak_pos = i;
  end
end

#see where minimum is after tallest peak
Lowest = func(max_peak_pos);

for i = (max_peak_pos+1):length(r_values)
  if (Lowest > func(i)) #i.e. gradient is negative
    Lowest = func(i);
    coord_max = i;
  else
    break
  end
end

#See where minimum is before tallest peak
for i = 2:(max_peak_pos-1)
   if (func(i-1) > func(i)) #So it sets coord_min if there is a negative gradient
     coord_min = i;
   endif
end

for i = coord_min:coord_max #Num should be at the point of the lowest peak
  int_table = [int_table, (r_values(i)^2).*norm_g_r_cutoff(i)]; #get sets of data points for CN integral
end
```

```
area = trapz(r_values(coord_min:coord_max),int_table); #integrate
CN_cutoff = 4*pi*p*area;

## EXPORTING AND GRAPHS ##
#Make a descending list of r values so we can save them in a text document
r_list = [];

for i = 1:length(r_values)
  r_list = [r_list;r_values(i)*10^10];
end

RDFdata = [r_list norm_g_r];
RDFdata2 = [r_list raw_g_r];
CutOffdata = [r_list norm_g_r_cutoff];
save Lorch_norm_RDF.txt RDFdata;
save Lorch_raw_RDF.txt RDFdata2;
save Lorch_norm_cutoff_RDF.txt CutOffdata;

figure(1)
plot(Q_values,S_Q_values)
xlabel('Q')
ylabel('S(Q)')
title('Structure Factor')
grid on
box on

figure(2)
plot(r_values,raw_g_r,'r',r_values,raw_g_r_cutoff,'b')
legend('Max Q','Q Cutoff')
xlabel('r')
ylabel('G(r) - G0')
title('Radial Distribution Function: Fourier transform')
grid on
box on

figure(3)
plot(r_values,norm_g_r,'r',r_values,norm_g_r_cutoff,'b')
legend('Max Q','Q Cutoff')
xlabel('r')
ylabel('g(r)')
title('Normalised g(r)')
grid on
box on
```

6.3 Fourier transform with Soper-Barney modification

```
## STEP 1: LOAD S(Q) DATA ##

clear variables
```

```
load Ar_335953.txt;
data_file = Ar_335953;
p = 2.5924E28; #number density of particles
Qmax = 870; #Q cutoff variables

Q_values = data_file(:,1)*10^10; #Q
S_Q_values = data_file(:,2); #S(Q)
n = 1000; #number of points to be calculated
r_max = 20*10^-10;
r_spacing = r_max/n;
r_values = r_spacing:r_spacing:r_max; #array of r values

Q_value = Q_values(Qmax)*10^-10; #Just for readout, the actual value of Q cutoff in A^-1
Q_values_cutoff = [];
S_Q_values_cutoff = [];

delta = 4.49./Q_values(length(Q_values));
cutoff_delta = 4.49./Q_values(Qmax);

for i = 1:Qmax #input values into Q for cutoff integral
  Q_values_cutoff = [Q_values_cutoff;Q_values(i)];
  S_Q_values_cutoff = [S_Q_values_cutoff;S_Q_values(i)];
end

## STEP 2: FINDING G(R) - G0 ##
raw_g_r = []; #g(r) - g0 with full range
raw_g_r_cutoff = []; #g(r) - g0 with cut-off

for i = 1:length(r_values) #we integrate for all values of r
  #the full-range integral
  F = Q_values.*(S_Q_values-
1).*sin(Q_values.*r_values(i)).*(3./((Q_values.*delta).^3)).*(sin(Q_values.*delta)-
(Q_values.*delta).*cos(Q_values.*delta));
  g = trapz(Q_values,F)*2/pi; #integrate
  gr = g/(4*pi*r_values(i)); #the integral gives 4*pi*r*[g(r) - g0], so divide by 4*pi*r
  raw_g_r = [raw_g_r;gr];
  #the cut-off integral
  F = Q_values_cutoff.*(S_Q_values_cutoff-
1).*sin(Q_values_cutoff.*r_values(i)).*(3./((Q_values_cutoff.*cutoff_delta).^3)).*(sin(Q_values_cuto
ff.*cutoff_delta)-(Q_values_cutoff.*cutoff_delta).*cos(Q_values_cutoff.*cutoff_delta));
  g = trapz(Q_values_cutoff,F)*2/pi; #integrate
  gr = g/(4*pi*r_values(i)); #the integral gives 4*pi*r*[g(r) - g0], so divide by 4*pi*r
  raw_g_r_cutoff = [raw_g_r_cutoff;gr];
end

## STEP 3: NORMALIZATION ##

norm_g_r = []; #normalized g(r) for full range
norm_g_r_cutoff = []; #normalized g(r) for cut-off

if (raw_g_r(1) > raw_g_r(length(r_values)))
```

```
  disp('Full range g(r) data cannot be normalized');
end

if (raw_g_r_cutoff(1) > raw_g_r_cutoff(length(r_values)))
  disp('Qmax cutoff g(r) data cannot be normalized');
end

for i = 1:length(r_values)
  norm_g_r = [norm_g_r;(raw_g_r(i) - raw_g_r(1))];
  norm_g_r_cutoff = [norm_g_r_cutoff;(raw_g_r_cutoff(i) - raw_g_r_cutoff(1))];
end

for i = 1:length(r_values)
  norm_g_r(i) = norm_g_r(i) / norm_g_r(length(r_values));
  norm_g_r_cutoff(i) = norm_g_r_cutoff(i) / norm_g_r_cutoff(length(r_values));
end

## STEP 4: CO-ORDINATION NUMBER ##

#CN: full range
int_table = [];
Max_Peak = 0;
max_peak_pos = 1; #Position of first Cshell max
coord_max = 1; #Upper limit for CN integration
coord_min = 1; #Lower limit for CN integration
func = norm_g_r;

for i = 2:length(r_values) #locate tallest peak
  if (func(i) > Max_Peak)
    Max_Peak = func(i);
    max_peak_pos = i;
  end
end

#see where minimum is after tallest peak
Lowest = func(max_peak_pos);

for i = (max_peak_pos+1):length(r_values)
  if (Lowest > func(i)) #i.e. gradient is negative
    Lowest = func(i);
    coord_max = i;
  else
    break
  end
end

#See where minimum is before tallest peak
for i = 2:(max_peak_pos-1)
    if (func(i-1) > func(i)) #So it sets coord_min if there is a negative gradient
      coord_min = i;
    endif
```

```
end

for i = coord_min:coord_max #Num should be at the point of the lowest peak
  int_table = [int_table, (r_values(i)^2).*norm_g_r(i)]; #get sets of data points for CN integral
end

area = trapz(r_values(coord_min:coord_max),int_table); #integrate
CN = 4*pi*p*area;

#CN: cutoff range
int_table = [];
Max_Peak = 0;
max_peak_pos = 1; #Position of first Cshell max
coord_max = 1; #Upper limit for CN integration
coord_min = 1; #Lower limit for CN integration
func = norm_g_r_cutoff;

for i = 2:length(r_values) #locate tallest peak
  if (func(i) > Max_Peak)
    Max_Peak = func(i);
    max_peak_pos = i;
  end
end

#see where minimum is after tallest peak
Lowest = func(max_peak_pos);

for i = (max_peak_pos+1):length(r_values)
  if (Lowest > func(i)) #i.e. gradient is negative
    Lowest = func(i);
    coord_max = i;
  else
    break
  end
end

#See where minimum is before tallest peak
for i = 2:(max_peak_pos-1)
    if (func(i-1) > func(i)) #So it sets coord_min if there is a negative gradient
      coord_min = i;
    endif
end

for i = coord_min:coord_max #Num should be at the point of the lowest peak
  int_table = [int_table, (r_values(i)^2).*norm_g_r_cutoff(i)]; #get sets of data points for CN integral
end

area = trapz(r_values(coord_min:coord_max),int_table); #integrate
CN_cutoff = 4*pi*p*area;

## EXPORTING AND GRAPHS ##
```

```
#Make a descending list of r values so we can save them in a text document
r_list = [];

for i = 1:length(r_values)
  r_list = [r_list;r_values(i)*10^10];
end

RDFdata = [r_list norm_g_r];
RDFdata2 = [r_list raw_g_r];
CutOffdata = [r_list norm_g_r_cutoff];
save Soper_norm_RDF.txt RDFdata;
save Soper_raw_RDF.txt RDFdata2;
save Soper_norm_cutoff_RDF.txt CutOffdata;

figure(1)
plot(Q_values,S_Q_values)
xlabel('Q')
ylabel('S(Q)')
title('Structure Factor')
grid on
box on

figure(2)
plot(r_values,raw_g_r,'r',r_values,raw_g_r_cutoff,'b')
legend('Max Q','Q Cutoff')
xlabel('r')
ylabel('G(r) - G0')
title('Radial Distribution Function: Fourier transform')
grid on
box on

figure(3)
plot(r_values,norm_g_r,'r',r_values,norm_g_r_cutoff,'b')
legend('Max Q','Q Cutoff')
xlabel('r')
ylabel('g(r)')
title('Normalised g(r)')
grid on
box on
```