ASSIGNMENT 11: SOLUTION

E. ARNOLD, M-S. LIU, R. PRABHU & C.S. RICHARDS
THE UNIVERSITY OF CAMBRIDGE

Written in XeLaTeX

BAYESIAN FITTING

PREREQUISITES

The chapter *Scattering Functions* from the *Theory Handbook*.

INTRODUCTION

In this tutorial, we will explore how we can use the intermediate scattering functions (ISFs) measured in a spin echo experiment to determine the paramaters of an adsorbate-substrate system.

THEORETICAL BACKGROUND

We will begin the theory section with an example system. Consider a Ru(0001) surface. This is a close-packed plane. On this surface, a substrate undergoes jump diffusion between the two different types of hollow site. The approach that we will use to determine the system parameters is:

1. Find a theoretical model for the ISF (as a function of momenum transfer $\Delta\mathbf{K}$ and spin-echo time t) that accurately describes the system we are measuring.
2. Use an appropriate fitting technique to fit the experimental ISF measurements to the theoretical model, determining which model parameters provide the best fit.

In the following text, we will, in turn, discuss how to perform these two steps.

Analytical Model for Jump Diffusion in Non-Bravais Lattices

How do we progress towards our objective of finding the system parameters? We first require a theoretical model; this model must describe diffusion in the adsorbate-substrate system. When the system displays weak interactions between adsorbates, and high atomic scale friction, analytical models often exist to describe the intermediate scattering function [1]. It should be emphasised that this is very useful. For most systems, the forces present are too complicated to model;

thus it is usually impossible to produce an analytical solution. Therefore we can fit the model used to some data, allowing us to obtain the system parameters.

In this tutorial, we will consider a close-packed plane. On this plane, an adsorbate can undergo jump diffusion between two different types of sites. We will describe the ISF of this system using the model developed in [2]. The key result from this model, which we can apply to any surface, is that for a adsorbate-substrate system where there are m different types of adsorption sites, the ISF can be written as a sum of m exponentials. Here the result of evaluating the model will be summarised for the system considered in this tutorial.

The system that we are interested in is a close-packed plane on iron. For this system, there are two site types, denoted 1 and 2, with total exit jump rates for the two site types being given by $1/\tau_{12}$ and $1/\tau_{21} \equiv 1/\lambda\tau_{12}$ respectively. The adsorbate concentrations on each site type are given by c_1 and $c_2 \equiv \lambda c_1$ respectively. The lattice constant is denoted a. We will only consider the analytic form of the ISF along the high-symmetry $[1\bar{1}0]$ and $[11\bar{2}]$ directions (see Figure 1).

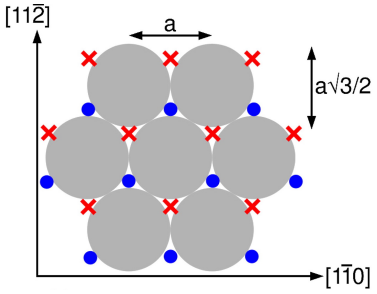

Figure 1 | A schematic of the fcc-(111) surface that we are measuring, with the two different types of hollow site labelled with circles and crosses. Taken from [2]

For the $[1\bar{1}0]$ direction, the ISF is given by:

$$I(\Delta\mathbf{K},t) = \frac{c_1}{n_1}\left|1-\lambda\frac{4\cos\left(\frac{\Delta Ka}{2}\right)+2}{3\lambda-3+z}\right|^2\exp\left(-\frac{1}{6\lambda\tau_{12}}(3\lambda+3+z)t\right);$$
$$+\frac{c_1}{n_2}\left|1-\lambda\frac{4\cos\left(\frac{\Delta Ka}{2}\right)+2}{3\lambda-3-z}\right|^2\exp\left(-\frac{1}{6\lambda\tau_{12}}(3\lambda+3-z)t\right).$$

The constants $n_{1,2}$ and z are given by:

$$n_{1,2} = 1+\lambda\left(\frac{4\cos\left(\frac{\Delta Ka}{2}\right)+2}{3\lambda-3\pm z}\right)^2;$$
$$z = \sqrt{9\lambda^2+16\lambda\cos^2\left(\frac{\Delta Ka}{2}\right)+16\lambda\cos\left(\frac{\Delta Ka}{2}\right)-14\lambda+9}.$$

For the $[11\bar{2}]$ direction, the ISF is given by:

$$I(\Delta\mathbf{K},t) = \frac{c_1}{m_1}\left|1-2\lambda\frac{\exp\left(i\frac{\Delta Ka}{\sqrt{3}}\right)+2\exp\left(-i\frac{\Delta Ka}{2\sqrt{3}}\right)}{3(\lambda-1+y)}\right|^2\exp\left(-\frac{1}{2\lambda\tau_{12}}(\lambda+1+y)t\right);$$
$$+\frac{c_1}{m_2}\left|1-2\lambda\frac{\exp\left(i\frac{\Delta Ka}{\sqrt{3}}\right)+2\exp\left(-i\frac{\Delta Ka}{2\sqrt{3}}\right)}{3(\lambda-1-y)}\right|^2\exp\left(-\frac{1}{2\lambda\tau_{12}}(\lambda+1+y)t\right).$$

The constants $m_{1,2}$ and y are given by:

$$m_{1,2} = 1+4\lambda\left|\frac{\exp\left(i\frac{\Delta Ka}{\sqrt{3}}\right)+2\exp\left(-i\frac{\Delta Ka}{2\sqrt{3}}\right)}{3(\lambda-1\pm y)}\right|^2;$$
$$y = \sqrt{\lambda^2+\frac{2\lambda}{9}\left(8\cos\left(\frac{\sqrt{3}\Delta Ka}{2}\right)+1\right)+1}.$$

We can see that this provides an analytical model that is paramaterised by two variables: the constants τ_{12} and λ. By fitting experimental ISF measurements to this model ISF, we can estimate the values of these parameters for the system that we are measuring. The constant τ_{12} will be denoted τ for the remainder of the document.

Fitting ISF measurements to the analytical model

In order to find the model that best describes a given set of data, we want to maximise the quantity denoted $P(M|D)$. This represents the probability that the model M is accurate given that the dataset D was obtained. By Bayes' Theorem, this probability can be written as:

$$P(M|D) = \frac{P(D|M)P(M)}{P(D)}$$

where $P(D|M)$ is the probability of obtaining the dataset given that the model is accurate, $P(M)$ is the probability that the model is accurate prior to obtaining the dataset, and $P(D)$ is the probability of obtaining the dataset. We know that choosing different values for the parameters in our model does not change $P(D)$, so this is a constant in our fitting problem. We also make the assumption that there is no *a priori* preference for any particular values for the model paramaters, meaning that $P(M)$ is also a constant. Therefore, the task of maximising $P(M|D)$ is equivalent to the task of maximising $P(D|M)$.

Assuming that each datapoint d_i within the dataset D is measured with a random error chosen from a Gaussian distribution, of mean 0 and standard deviation σ_i, then we can find the probability that the datapoint d_i is obtained from a model point of m_i; this is

$$P(d_i|m_i) = \frac{1}{\sigma_i\sqrt{2\pi}} \exp\left(\frac{-(d_i - m_i)^2{}^2}{2\sigma_i}\right).$$

This means that we can equivalently express the conditional probability $P(D|M)$ as

$$P(D|M) = \prod_i P(d_i|m_i).$$

The model described above gives an analytical form for the ISF; this is due to the motion of dynamic scatterers. In order to fully model a spin echo measurement, we also need to add a term, denoted C, that accounts for the signal due to static scatterers [3]. This means that the model to which we are trying to fit our experimental data can be written in the form:

$$M(t) = Af(t) + C,$$

where $f(t)$ is the model given above, paramaterised by τ and λ, and A and C are further parameters (although we are not interested in their values for this exercise).

This means that $P(D|M)$ can be considered to be a function of A, C, and $f(t)$. If we wish to find the $P(D|M)$ as a function of just one of these variables, we need to 'marginalise' (i.e. integrate) over the other variables to obtain $P(D|M)$ as a function of the desired variable only. Therefore

$$P(D|M) = P(f(t)) = \iint P(A, C, f(t))\,\mathrm{d}A\mathrm{d}C,$$

where the integration must be carried out over all possible values of A and C. It would, in principle, be possible to use this equation to find the most likely values of the parameters τ and λ in $f(t)$. However, given that this would be very difficult to achieve computationally, there is an analytic way of marginalising over A and C such that we can obtain an expression for $P(f(t))$. We can then evaluate this expression at many points in model parameter space and find the maximum of the probability distribution, thus finding the most likely values of λ and τ.

This method discussed above will not be reproduced here - it is beyond the scope of this document. However, the code that carries it out is provided for you. If the reader is interested, they may read about the method in [3], or the supplementary material of [1].

TASK

1. Generate data to simulate a spin-echo measurement on an adsorbate-substrate system. In this system, the substrate is a Ru(0001) surface, and the adsorbate may attach to either of the two types of hollow site. The data should contains ISFs along the $[1\bar{1}0]$ and $[11\bar{2}]$ azimuths, for a reasonable range of values of the parallel momentum transfer ΔK.
2. Use the Bayesian fitting method, and the analytic model outlined above, to find the best estimate for the values of the constants λ and τ that correspond to your experimental data
3. Plot a graph in τ - λ space to illustrate your results

The code that uses the marginalised Bayesian fitting method to give the conditional probability $P(D|M)$ is provided below.

Tip 1

Generate the data by using the analytic model provided to model the system for a particular value of each constant τ and λ. Then add white noise (a Gaussian distribution with a fixed standard deviation, and a mean of 0).

Tip 2

Think carefully about the inputs and outputs of the Bayesian fitting function: as it is written, the function takes a set of data points and a set of model points; it returns $P(D|M)$. Given that we want to evaluate the conditional probability $P(D|M)$ for each possible value of λ and τ, our code should:

1. Evaluate the model prediction for the ISF for each value of λ and τ
2. Input this model prediction, along with the experimental data, into the Bayesian fitting function. This will calculate $P(D|M)$ for these particular values of λ and τ
3. Repeat for different values of λ and τ, making sure you cover a reasonable range of λ - τ space
4. Maximise $P(D|M)$ with respect to both λ and τ to find the most likely values of these parameters.

BAYESIAN FITTING FUNCTION

This function evaluates $P(D|M)$ for a particular dataset D, and a particular model M. It is not necessarily important that you follow every step of the code, but make sure you understand what the inputs and outputs are so that you can use the function properly:

```
1  function [prob_analytical,logprob,prob_exact,A0,C0,flag] =
       BProb2_data(n1,n2,D,error,fitval,flag1,flag2)
2  % The function returns the relative probability that a data set
       D can be fitted by a model function plus a constant
       background offset. The analytical method for this can be
       found in Pepijn's thesis
3  %
4  % Input arguments:
5  %
6  %  D:       The 1xn array (where n is at least n2) containing
       the n datapoints
7  %
8  %  error:   The 1xn array containing the standard deviation of
       each data point in D
9  %
10 %  fitval:  The 1xn array containing the model function
       evaluated at the
11 %           same values of DeltaK and t that each datapoint in D
        was evaluated at
12 %
13 %  flag1:   If this is set to 1 then the function will calculate
        the
14 %           numerically integrated marginalised function (
       prob_exact) as well
15 %           as the analytical one (prob_analytical). Otherwise,
       the function simply sets
16 %           prob_exact=prob_analytical
17 %
18 %  flag2:   When set to 1, the best fitting value of A is simply
        assumed to
```

```
19  %           be 1. Use this if you know your experimental data is
        of the
20  %           form D = model + C + noise (with no prefactor on
        model)
21  %
22  %  n1:      The first element in the arrays D, error and fit to
        be used
23  %  n2:      The last element in these arrays to be used in this
        fit.
24  %
25  %
26  % Output arguments:
27
28  % prob_analytical: The (relative) probability that function
        given in the array 'fitval' plus an offset constant will fit
         the data, formed by integrating the probability over (i.e
        marginalising) all possible values of A and C where attempts
         are made to fit the data with:  D(i)=A*fitval(i)+C.
29  %
30  %'prob_exact': The same as prob_analytical, except worked out by
         a numerical integration routine (that requires the function
         prob2.m) only over non negative values of A and C
31  %
32
33  global K0 K1 K2 K11 K12 K22 evecK sdum A0 C0
34
35  % nstand is the number of standard deviations away from peak we
        will numercially integrate.
36  % A good value is nstand=4
37
38  % try setting it as 4
39  nstand=4;
40
41
42  if (isrow(fitval) && iscolumn(D)) || (isrow(D) && iscolumn(
        fitval))
43      fitval = fitval';
44  end
45
46  [D,fitval,devider] = prep_isf_for_comparison(D,fitval);
47
48
49  % ftol is the fractional tolerance that we wish to integrate
        numerically to
50  % A reasonable value is ftol=1e-4
51  ftol=1e-4;
52
53  % create the sums needed for the least squares fit
54  K0=0.0;
55  K1=0.0;
56  K2=0.0;
57  K11=0.0;
58  K12=0.0;
59  K22=0.0;
60  sdum=0.0;
61
62  variance=error(n1:n2).*error(n1:n2);
63  K0=sum(D(n1:n2).*D(n1:n2)./variance);
64  K1=sum(fitval(n1:n2).*D(n1:n2)./variance);
65  K2=sum(D(n1:n2)./variance);
66  K11=sum(fitval(n1:n2).*fitval(n1:n2)./variance);
```

```
K12=sum(fitval(n1:n2)./variance);
K22=sum(1./variance);
sdum=sum(log(variance));

sdum=sdum+(1+n2-n1)*log(2*pi);

% sum from n1 to n2, so the number of terms is 1+n2-n1
%  the least squares best fit values of A0 and C0
A0=(K22*K1-K12*K2)/(K11*K22-K12^2);
C0=(K11*K2-K12*K1)/(K11*K22-K12^2);
if flag2==1
    A0=1;
    C0=0;
end


% perform the marginalisation - integrate probability of a fit
    over all (reasonable!) values of A and C
flag=1;

% this piece of code was a rough attempt at eliminating values
    where the fits went out of the acceptable range and uses '
    flag=0' to show  when this is occurring - i.e. the most
    likely value of A and C, A0 and C0 have one of the negative
    if(C0 < 0.0)
if(C0/(A0+C0) < -0.1)
    flag=0;
end
if(A0 < 0.0)
    flag=0;
end

S0=K0-2*A0*K1-2*C0*K2+A0*A0*K11+2*A0*C0*K12+C0*C0*K22+sdum;
tmp = K11*K22-K12^2; if tmp<=0, tmp=eps; end
logprob=-S0/2+log(2*pi)-0.5*log(tmp);

%prob(ialpha)=exp(max(-200,logprob(ialpha)));
prob_analytical=exp(logprob);
if(flag < 1)
   prob_analytical=0.0;
end
if(flag1==1)

% Calculate the numerical integrated value. Start by working out
     eigenvectors (give the symmetry directions of the pdf in A
    and C space) and values of the K matrix
KMat(1,1)=K11;
KMat(2,1)=K12;
KMat(1,2)=K12;
KMat(2,2)=K22;
evalK=eig(KMat);
[evecK,Dum] = eig(KMat);

% prob=exp(-S/2)  where
% S=S0+evalK(1).(evec_coord1)^2+evalK(2).(evec_coord2)^2
% eigenvalues are 1/(stdev)^2 in each direction
evc1range=1/sqrt(evalK(1))*nstand;
evc2range=1/sqrt(evalK(2))*nstand;

% Integrate exp(-0.5*(S0+(A-A0,C-C0)Kmat(A-A0,C-C0)T)) but in
    eigenvector coordinates over nstand standard deviations in
```

```
        each eigen vector %direction - prob2 works out internally A
        and C, and then sets the integrand to zero if A<0 or if C<0
120 tol=ftol*exp(logprob);
121
122 % use the previous analytical estimate of the probability so as
        to work out an absolute tolerance (tol) as needed by the
        integration routine as opposed to the relative (or
        fractional) tolerance (ftol) that it is more useful to think
         in terms of.
123 prob_exact = dblquad(@prob2,-evc1range,evc1range,-evc2range,
        evc2range,tol);
124 else
125
126 % if flag1 is not set to 1 we do not numerically integrate, but
        just set the exact probability to be equal to the analytical
         one.
127     prob_exact=prob_analytical;
128 end
129 end
130
131 function z = prob2( ec1,ec2 )
132         % ec1 and ec2 are coordinates in eigen vector space
133         global K0  K1 K2 K11 K12 K22 evecK sdum A0 C0
134
135         A=evecK(1,1)*ec1+evecK(1,2)*ec2+A0;
136         C=evecK(2,1)*ec1+evecK(2,2)*ec2+C0;
137         z=(abs(A)+A)./A.*(abs(C)+C)./C/4.*exp(-(K0-2*A*K1-2*C*K2
                +A.*A*K11+2*A.*C*K12+C.*C*K22+sdum)/2);
138 end
139
140 function [isfA,isfB,devider] = prep_isf_for_comparison(isfA,isfB
        )
141         max_A = max(isfA);
142         max_B = max(isfB);
143         if max_A > 1 || max_B > 1
144             max_of_all = max(max_A,max_B);
145             isfA = isfA/max_of_all;
146             isfB = isfB/max_of_all;
147         else
148             max_of_all = 1;
149         end
150
151         devider = max_of_all;
152 end
```

REFERENCES

1. B. A. J. Lechner, P. R. Kole, H. Hedgeland, A. P. Jardine, W. Allison, B. J. Hinch and J. Ellis, *Phys. Rev. B* 89(12),121405
2. F. E. Tuddenham, H. Hedgeland, A. P. Jardine, B. A. J. Lechner, B. J. Hinch, and W. Allison, *Surf. Sci.* 604, 1459 (2010).
3. Pepjin Kole, *PhD Thesis*.

SOLUTIONS

GLOBAL BAYESIAN FITTING: MAIN TASK

The first step is to write a function that using the analytical model from [2] to output a model ISF for particular values of λ and τ. This is done below:

```
function [ISF, preexp1, preexp2, exp1, exp2] =
    BProb_hollow_model(an,tau,lambda,dk,setime,azimuth)

% Function to output Fay's model for hollow site adsorption for
    given
% values of dK, lambda, tau and spin-echo time
%
% Input parameters:
% an:       lattice parameter for respective surface
% tau:      residence time in site 1; >=1
% lambda:   ratio of residence times in two sites; 1 means
    degenerate sites; >=1
% dk:       magnitude momentum transfer along given azimuth
% setime:   spin-echo time [ps]
% azimuth:  direction of momentum transfer; can be 110 or 112
%
% output parameters:
% The model outputs an ISF of the form:
% ISF = preexp1 * e^(exp1 t) +  preexp2 * e^(exp2 t)



% Coding in the equation from Tuddenham et al, 2010:

%For the $[1\bar{1}0]$ direction:
if azimuth == 110

    %The value of z (will be included as a variable in the ISF)
    z = sqrt(9*lambda.^2 + 16*lambda*((cos(dk*an/2)).^2) + 16*
        lambda*cos(dk*an/2) - 14*lambda +9);

    %The value of n1 (will be included as a variable in the ISF)
    n1 = 1 + lambda.*((4*cos(dk*an/2)+2)./(3*lambda-3+z)).^2;

    %The value of n2 (will be included as a variable in the ISF)
    n2 = 1 + lambda.*((4*cos(dk*an/2)+2)./(3*lambda-3-z)).^2;

    %The value of the ISF is encoded below in the form ISF =
        preexp1 * e^(exp1 *t) + preexp2 * e^(exp2 * t)

```

```
36      preexp1 = 1/n1.*(abs(1-lambda.*(4*cos(dk*an/2)+2)./(3*lambda
            -3+z))).^2;
37      preexp2 = 1/n2.*(abs(1-lambda.*(4*cos(dk*an/2)+2)./(3*lambda
            -3-z))).^2;
38
39      exp1 = -1/(6*lambda*tau)*(3*lambda+3+z);
40      exp2 = -1/(6*lambda*tau)*(3*lambda+3-z);
41
42  %For the $[1\bar{1}0]$ direction:
43  elseif azimuth == 112
44
45      %The value of y (will be included as a variable in the ISF)
46      y = sqrt(lambda.^2 + 2*lambda/9*(8*cos(sqrt(3)*dk*an/2)+1) +
             1);
47
48      %The value of m1 (will be included as a variable in the ISF)
49      m1 = 1 + 4*lambda*(abs((exp(1i*dk*an/sqrt(3))+2*exp(-1i*dk*
            an/(2*sqrt(3))))./(3*(lambda-1+y)))).^2;
50
51      %The value of m2 (will be included as a variable in the ISF)
52      m2 = 1 + 4*lambda*(abs((exp(1i*dk*an/sqrt(3))+2*exp(-1i*dk*
            an/(2*sqrt(3))))./(3*(lambda-1-y)))).^2;
53
54      %The value of the ISF is encoded below in the form ISF =
            preexp1 * e^(exp1 *t) + preexp2 * e^(exp2 * t)
55
56      preexp1 = 1/m1.*(abs(1-2*lambda*(exp(1i*dk*an/sqrt(3))+2*exp
            (-1i*dk*an/(2*sqrt(3))))./(3*(lambda-1+y)))).^2;
57      preexp2 = 1/m2.*(abs(1-2*lambda*(exp(1i*dk*an/sqrt(3))+2*exp
            (-1i*dk*an/(2*sqrt(3))))./(3*(lambda-1-y)))).^2;
58
59      exp1 = -1/(2*lambda*tau)*(lambda+1+y);
60      exp2 = -1/(2*lambda*tau)*(lambda+1-y);
61
62  end;
63
64  %The value of the ISF
65  ISF = preexp1.*exp(exp1.*setime) + preexp2.*exp(exp2.*setime);
66
67  end
```

Moving to our main script, we first need to generate some experimental data that we can fit to the model. As suggested in Tip 1, we will do this by first generating some model data using the `BProb_hollow_model` function above and then adding some noise.

We also need to decide in what form to store this experimental data. There are many ways to do this, but the approach used here is to store the data in an object called a 'structure array'. For each element in a structure array, there are a collection of data containers known as 'fields', which can contain data of any type or size. In our case, we are going to generate m ISF measurements, which will each be their own element in the structure array (such that the structure array has dimensions of 1xm), and each of these experimental measurements will have associated with it the fields:

- ISF - the value of the experimental ISF 'measurement' (i.e. the generated model value plus some Gaussian noise

- dK - the value of dK at which the ISF measurement was taken
- azimuth - the direction along which the ISF measurement was taken
- setime - the spin echo time used to take the ISF measurement
- std - the standard deviation of the random error in the ISF measurement
- temperature - the temperature at which the ISF measurement was taken

The structure array, which we have called `res`, is generated in the code below. It should take the form shown in Figure 1.

```
% Ru(0001) lattice constant for jump model
an = 2.71;

%-------------------------------------------------

%Creating experimental data (with m ISF values)

%Value of lambda in our (made up) experimental system
lambda = 2;

%Value of tau in our experimental system
tau = 3;

%Magnitudes of dK at which 'measurements' are taken
dK = 0:0.5:4;

%Temperature at which 'measurements' are taken
temperature = 200;

%Spin echo time at which 'measurements' are taken
setime = 3;

%Standard deviation of random error in 'measurements'
std = 0.1;

%1xm matrix with Gaussian random error for each ISF value
noise = normrnd (0, std, [1, 2*size(dK,2)]);

%Adding ISF measurements for 110 azimuth
for i = 1:size(dK, 2)

    %Generating model ISFs using Fay model
    ISF (1, i) = BProb_hollow_model(an, tau, lambda, dK (i),
        setime, 110);

    %Adding noise to model ISF to simulate experimental data
    ISF_wth_noise (1, i) = ISF (1, i) + noise (i);

    %Populating the fields of the 'res' structure

    %ISF measurements
    res(i).ISF = ISF_wth_noise (1, i);

    %Values of dK at which the measurements were taken
    res(i).dK = dK (i);

    %Direction along which the measurements were taken
```

```
47      res(i).azimuth = 110;
48
49      %The spin echo time used to take the measurements
50      res(i).setime = setime;
51
52      %Standard deviation of random error in measurements
53      res(i).std = std;
54
55      %Temperature at which measurements are taken
56      res(i).temperature = temperature;
57  end
58
59
60  %Adding ISF measurements for 112 azimuth
61  for i = 1:size(dK, 2)
62
63      %Generating model ISF using Fay model
64      ISF (2, i) = BProb_hollow_model(an, tau, lambda, dK (i),
            setime, 112);
65
66      %Defining an index that starts from where 110 measurements
            left off
67      j = i + size (dK, 2);
68
69      %Adding noise to model ISF to simulate experimental data
70      ISF_wth_noise (2,i) = ISF (2, i) + noise (j);
71
72      %Populating the fields of the 'res' structure
73
74      %ISF measurements
75      res(j).ISF = ISF_wth_noise (2, i);
76
77      %Values of dK at which the measurements were taken
78      res(j).dK = dK (i);
79
80      %Direction along which the measurements were taken
81      res(j).azimuth = 112;
82
83      %The spin echo time used to take the measurements
84      res(j).setime = setime;
85
86      %Standard deviation of random error in measurements
87      res(j).std = std;
88
89      %Temperature at which measurements are taken
90      res(j).temperature = temperature;
91  end
```

We then extract some relevant information from the `res` structure, which will become useful later:

```
92  %Prepare matrices that will be useful later
93
94  %The number of ISF measurements
95  numISFs = length(res);
96
97  %The magnitudes of momentum transfer for each measurement
98  dK = [res.dK];
99
100 %The temperatures for each measurement
101 T = [res.temperature];
102
```

1x18 struct with 6 fields

Fields	ISF	dK	azimuth	setime	std	temperature
1	3.0538	0	110	3	0.1000	200
2	2.9040	0.5000	110	3	0.1000	200
3	1.9023	1	110	3	0.1000	200
4	1.7023	1.5000	110	3	0.1000	200
5	1.3787	2	110	3	0.1000	200
6	1.1849	2.5000	110	3	0.1000	200
7	1.4737	3	110	3	0.1000	200
8	2.0023	3.5000	110	3	0.1000	200
9	2.9284	4	110	3	0.1000	200
10	3.2769	0	112	3	0.1000	200
11	2.5858	0.5000	112	3	0.1000	200
12	2.4378	1	112	3	0.1000	200
13	1.7299	1.5000	112	3	0.1000	200
14	1.4518	2	112	3	0.1000	200
15	1.5149	2.5000	112	3	0.1000	200
16	1.4101	3	112	3	0.1000	200
17	1.3403	3.5000	112	3	0.1000	200
18	1.4499	4	112	3	0.1000	200

Figure 2 | An example of the form that the res structure will take if created using the code in this solution

```
%The azimuth for each measurement
azimuth = [res.azimuth];
```

We then need to find $P(D|M)$ for each value of λ and τ. The way we do this here is, for each pair of λ and τ values, we find $P(d_i|M)$ for each datapoint, i, and then add together the log of the probabilities for each datapoint to find the total $P(D|M)$ for that particular pair λ and τ. Note that, each time we call the `BProb2_data` function, we are only inputting a single datapoint (with a single modelpoint), so `n1` and `n2` (the start and end points for the fit) should simply both be set to 1:

```
% Setting up range of tau and lambda over which to try fitting

%This means that we will try fitting to lambda values between 2
    and 500, with intervals of 30
lambda_vector = linspace(2,30,500);

%This means that we will try fitting to lambda values between
    0.01 and 500, with intervals of 2.5
tau_vector = linspace(0.01,2.5,500);

% Pre-allocate the dimensions of some matrices for speed (the
    values in these matrices will be allocated later):
prob = zeros(length(lambda_vector),length(tau_vector),numISFs);
prob_exact = zeros(length(lambda_vector),length(tau_vector),
    numISFs;
A0 = zeros(length(lambda_vector),length(tau_vector),numISFs);
C0 = zeros(length(lambda_vector),length(tau_vector),numISFs);
flag = zeros(length(lambda_vector),length(tau_vector),numISFs);
S = zeros(length(lambda_vector),length(tau_vector));
S2 = zeros(length(lambda_vector),length(tau_vector));

%Starting point for fit
n1 = 1;
%Ending point for fit
n2 = 1;

% The loop that finds P(D|M) for each pair of lambda and tau:


```

```
%Adding a waitbar to show progress in fitting
h = waitbar(0,'Please wait...'); steps=length(lambda_vector);


%Loop over lambda values
for ind1 = 1:length(lambda_vector)
    %Loop over tau values
    for ind2 = 1:length(tau_vector)
        %Loop over ISF measurements
        for dg = 1:numISFs


            % Finding model function for current value of lambda
                 and tau using Brop_hollow_model function
            [fitval{ind1,ind2,dg} preexp1{ind1,ind2,dg} preexp2{
                ind1,ind2,dg}...
                exp1{ind1,ind2,dg} exp2{ind1,ind2,dg}] =
                    BProb_hollow_model(an,...
                tau_vector(ind2),lambda_vector(ind1),dK(dg),res(
                    dg).setime,azimuth(dg));

            % Find P(D|M) for current ISF measurement and
                current lambda and tau using BProb2_data:
            [prob(ind1,ind2,dg),prob_exact(ind1,ind2,dg),A0(ind1
                ,ind2,dg),...
                C0(ind1,ind2,dg),flag(ind1,ind2,dg)] =
                    BProb2_data(n1,...
                n2,res(dg).ISF,res(dg).std,fitval{ind1,ind2,dg
                    },0,0);

            % Sum over log of probabilities for all ISF
                measurements to get total probability for this
                lambda/tau pair (using max function to avoid 0
                probability values):
            S(ind1,ind2) = S(ind1,ind2) + log(max(prob_exact(
                ind1,ind2,dg),1e-323));
            S2(ind1,ind2) = S2(ind1,ind2) + log(max(prob(ind1,
                ind2,dg),1e-323));
        end
    end
    waitbar(ind1/steps)
end
close(h)
```

The `S` and `S2` outputs from that process contain the probability that the data fits the model, for each possible pair of λ and τ. The two variables contain the same information, except that the `BProb2_data` function calculates `S` using an analytic technique and `S2` using a numerical technique.

The next step is to plot this probability matrix in λ-τ space:

```
% Plot the probability matrix in lambda/tau space:
if length(lambda_vector)>1
    figure(1)
    hold on; box on;
    set(gcf,'windowstyle','docked')

    %Creating a grid at which to sample P(D|M)
    [X Y] = meshgrid(tau_vector,lambda_vector);

```

```
169     %Creating a surf plot of log(P(D|M)) in lambda-tau space
170     surf(X,Y,S);
171     shading interp
172     view(2)
173     xlabel('$\tau$')
174     ylabel('$\lambda$')
175     title('Log of probability as a function of $\lambda$ and $\
            tau$');
176
177     % Calculating the relative probabilities:
178
179     %Scaling S
180     S_scale = S - max(max(S));
181
182     %Exponentiating to get P(D|M)
183     rel_prob = exp(S_scale);
184
185     figure(2)
186     hold on; box on;
187     set(gcf,'windowstyle','docked')
188
189     %Creating a grid at which to sample P(D|M)
190     [X Y] = meshgrid(tau_vector,lambda_vector);
191
192     %Creating a surf plot of P(D|M) in lambda-tau space
193     surf(X,Y,rel_prob);
194     shading interp
195     view(3)        % 3D view
196     %view(2)       % 2D view
197     xlabel('$\tau$')
198     ylabel('$\lambda$')
199     c=colorbar;
200     ylabel(c,'Relative probability');
```

Finally, we can marginalise (integrate) over τ to obtain $P(D|M)$ as a function of λ, and thus obtain the most likely value of λ (although of course you could do this the other way around by marginalising over λ instead). The code below plots $P(D|M)$ as a function of λ and outputs the most likely value of λ:

```
202 % Marginalising over tau:
203     for ind1 = 1:length(lambda_vector)
204         prob_lambda(ind1) = sum(rel_prob(ind1,:));
205     end;
206
207 %Plotting P(D|M) as function of lambda
208     figure(3)
209     hold on; box on;
210     set(gcf,'windowstyle','docked')
211     plot(lambda_vector,prob_lambda,'-*');
212     xlabel('\lambda')
213     ylabel('Relative probability')
214
215 %Displaying most likely value of lambda
216     [lambda_max, lambda_max_ind] = max(prob_lambda);
217     lambda_fit = lambda_vector(lambda_max_ind);
218
219     disp(['max probability found for lambda = ' num2str(
            lambda_fit)]);
```

If it is of interest, you may also wish to look at what values you may have assigned to λ and τ if you had simply looked at each ISF measurement seperately. The code below allows you to do that:

```
%Get optimum tau and lambda for each ISF measurement and plot
    results:

%Looping over the ISF measurements
for dg = 1:numISFs
    %Find best fit in tau-lambda space:

    %Max of lambda for each tau
    [maxprob_lambda,maxind_lambdavec] = max(prob(:,:,dg));

    %Max in lambda-tau space
    [maxprob,maxind_tau] = max(maxprob_lambda);
    maxind_lambda = maxind_lambdavec(maxind_tau);

    %ind_tau (j) stores most likely value of tau for j^th
        measurement
    ind_tau(dg) = maxind_tau;

    %ind_lambda (j) stores most likely value of lambda for j^th
        measurement
    ind_lambda(dg) = maxind_lambda;

    % Plotting alpha(dK):
    figure(4)
    set(gcf,'windowstyle','docked')
    hold on; box on;

    %Plotting the first decay constant associated with each of
        the most likely values of lambda and tau
    plot(-dK(dg),-exp1{ind_lambda(dg),ind_tau(dg),dg},'r*');

    %Plotting the second decay constant associated with each of
        the most likely values of lambda and tau
    plot(-dK(dg),-exp2{ind_lambda(dg),ind_tau(dg),dg},'b*');
    legend ('exp1', 'exp2', 'Location', 'southoutside')
    xlabel ('$\Delta K$')
    ylabel ('Decay constant')

    %Plotting preexp (dK):
    figure(5)
    set(gcf,'windowstyle','docked')
    hold on; box on;

    %Plotting the first pre-exponential factor associated with
        each of the most likely values of lambda and tau
    plot(-dK(dg),preexp1{ind_lambda(dg),ind_tau(dg),dg},'r*');

    %Plotting the second pre-exponential factor associated with
        each of the most likely values of lambda and tau
    plot(-dK(dg),preexp2{ind_lambda(dg),ind_tau(dg),dg},'b*');
    legend ('preexp1', 'preexp2', 'Location', 'southoutside')
    xlabel ('$\Delta K$')
    ylabel ('Pre-exponential constant')
end;
```

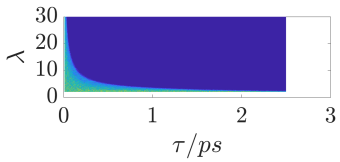

Figure 3 | A colour plot of $\log(P(D|M))$ as a function of the parameters λ and τ. The simulation was run with experimental data for a system where λ is 2, τ is 3ps, the random error in the data had a standard deviation of 0.1 and the other system parameters are given in the code provided in this solution.

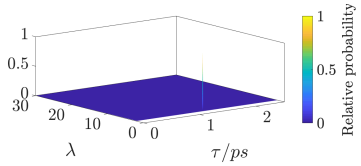

Figure 4 | A plot of $P(D|M)$ as a function of the parameters λ and τ. The simulation was run with experimental data for a system where λ is 2, τ is 3ps, the random error in the data had a standard deviation of 0.1 and the other system parameters are given in the code provided in this solution.

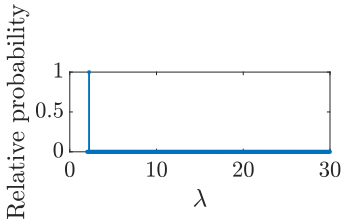

Figure 5 | A plot of $P(D|M)$ as a function of λ only. The simulation was run with experimental data for a system where λ is 2, τ is 3ps, the random error in the data had a standard deviation of 0.1 and the other system parameters are given in the code provided in this solution.