Supporting Information for: Trust-Region Algorithm for the Inversion of Molecular Diffusion NMR Data

Kaipin Xu and Shangmin Zhang*

Physics Department and Shanghai Key Laboratory of Magnetic Resonance, East China Normal University, Shanghai, 200062, China

E-mail: shanminz@hotmail.com

Description of the algorithm

A scheme of the Trust-Region Algorithm for the Inversion (TRAIn) of molecular diffusion NMR data is given below.

Step 1 *Initialize the algorithm*

Input the experimental data set vector **s**, the parameter set vector **p** and the set of discrete diffusion coefficient values, **d**.

Generate the coefficient matrix **C** according to

$$\begin{aligned} C_{ij} &= \exp(-d_j p_i) \\ \text{where} \quad p_i &= \gamma^2 \delta^2 g_i^2 \Delta' \end{aligned} \tag{S.1}$$

Estimate the noise level of the data set via the NNLS algorithm (by using the MATLAB function *lsqnonneg*), i.e.

$$\varepsilon = \left\| \mathbf{C}\mathbf{h}_{\text{NNLS}} - \mathbf{s} \right\|_2 \tag{S.2}$$

Set iteration index $k = 0$, input initial guess $\mathbf{h}^{(0)}$ or $\boldsymbol{\eta}^{(0)}$, initial trust-region radius $r^{(1)}$, and controlling parameters $0 < \mu_0 \le \mu_1 < \mu_2 < 1$, $0 < \nu_1 < 1 < \nu_2$. The parameters μ_0, μ_1 and μ_2 are the thresholds controlling whether to update the solution and the trust-region radius while ν_1 and ν_2 are the multipliers of the trust-region radius, with ν_1 for shrinking the region of trust and ν_2 for expanding it. Typical values for these controlling parameters are $\mu_0 = 0.01$, $\mu_1 = 0.25$, $\mu_2 = 0.75$, $\nu_1 = 0.5$ and $\nu_2 = 2$.

Compute and store matrix $\mathbf{C}^{\mathrm{T}}\mathbf{C}$ and vector $\mathbf{C}^{\mathrm{T}}\mathbf{s}$ since they will be frequently used during the iteration procedure. Calculate the objective function $\Phi(\boldsymbol{\eta}^{(0)})$.

Step 2 *Start main loop iteration*

Set $k = k + 1$ and compute the *gradient* vector and the *Hessian* matrix of $\Phi(\boldsymbol{\eta}^{(k)})$. Since the *Jacobian* matrix of $\Phi(\boldsymbol{\eta}^{(k)})$ at the k-th iteration is

$$\mathbf{J}^{(k)} = 2\mathbf{C} \cdot diag(\boldsymbol{\eta}^{(k)}) \tag{S.3}$$

the *gradient* vector and the *Hessian* matrix at the k-th iteration become

$$\begin{aligned} \mathbf{g}^{(k)} &= 2\mathbf{J}^{(k)^{\mathrm{T}}}(\mathbf{C}\mathbf{h}^{(k)} - \mathbf{s}) = 4 \cdot diag(\boldsymbol{\eta}^{(k)}) \cdot (\mathbf{C}^{\mathrm{T}}\mathbf{C}\mathbf{h}^{(k)} - \mathbf{C}^{\mathrm{T}}\mathbf{s}) \\ \mathbf{H}^{(k)} &= 2\mathbf{J}^{(k)^{\mathrm{T}}}\mathbf{J}^{(k)} = 8 \cdot diag(\boldsymbol{\eta}^{(k)}) \cdot \mathbf{C}^{\mathrm{T}}\mathbf{C} \cdot diag(\boldsymbol{\eta}^{(k)}) \end{aligned} \tag{S.4}$$

In consideration of computational efficiency, the stored matrix $\mathbf{C}^{\mathrm{T}}\mathbf{C}$ and vector $\mathbf{C}^{\mathrm{T}}\mathbf{s}$ can be directly applied to Eq. (S.4).

Initialize the sub loop for solving the trust-region subproblem.

Step 3 *Start sub loop iteration*

Solve the trust-region subproblem:

$$\min_{\|\mathbf{z}^{(k)}\|_2 \le r^{(k)}} \mathbf{g}^{(k)\mathrm{T}}\mathbf{z}^{(k)} + \frac{1}{2}\mathbf{z}^{(k)\mathrm{T}}\mathbf{H}^{(k)}\mathbf{z}^{(k)} \tag{S.5}$$

where $\mathbf{z}^{(k)}$ is the trial step at the k-th iteration. For details of the truncated conjugate gradient algorithm proposed by Steihaug, one may consult ref 33.

Check whether the break condition of the sub loop is satisfied. If yes, go to *Step 4*. Otherwise, go back to *Step 3*.

Step 4 *Update the solution, the trust-region radius and prepare for next iteration*

Define the actual reduction of the objective function as $\Delta\Phi_{\mathrm{Act}}$, and the predicted reduction as $\Delta\Phi_{\mathrm{Pre}}$, where

$$\begin{aligned} \Delta\Phi_{\mathrm{Act}} &= \Phi(\boldsymbol{\eta}^{(k-1)}) - \Phi(\boldsymbol{\eta}^{(k-1)} + \mathbf{z}^{(k)}) \\ \Delta\Phi_{\mathrm{Pre}} &= -\mathbf{g}^{(k)\mathrm{T}}\mathbf{z}^{(k)} - \frac{1}{2}\mathbf{z}^{(k)\mathrm{T}}\mathbf{H}^{(k)}\mathbf{z}^{(k)} \end{aligned} \tag{S.6}$$

and set the ratio $\rho = \Delta\Phi_{\mathrm{Act}} / \Delta\Phi_{\mathrm{Pre}}$.

Update the solution according to

$$\boldsymbol{\eta}^{(k)} = \begin{cases} \boldsymbol{\eta}^{(k-1)} & (\text{if } \rho \le \mu_0) \\ \boldsymbol{\eta}^{(k-1)} + \mathbf{z}^{(k)} & (\text{if } \rho > \mu_0) \end{cases} \tag{S.7}$$

Update the trust-region radius according to

$$r^{(k+1)} = \begin{cases} \max(\nu_1 r^{(k)}, r_{\min}) & (\text{if } \rho \le \mu_1) \\ \min(\nu_2 r^{(k)}, r_{\max}) & (\text{if } \rho \ge \mu_2) \\ r^{(k)} & (\text{otherwise}) \end{cases} \tag{S.8}$$

Calculate the objective function $\Phi(\boldsymbol{\eta}^{(k)})$. Check whether the termination condition (Eq. (9)) is satisfied. If yes, go to *Step 5*. Otherwise, go back to *Step 2*. A typical value for τ is around 1.02 (in all simulations and experiments of this work, we adjusted τ from 1.01 to 1.03 with respect to the residual curve plot).

Step 5 *Terminate the computation procedure and output the results*

Stop iteration and compute $\mathbf{h}^{(k^*)} = diag(\boldsymbol{\eta}^{(k^*)})\cdot\boldsymbol{\eta}^{(k^*)}$ and $\boldsymbol{\chi} = \mathbf{s} - \mathbf{C}\mathbf{h}$. Return $\mathbf{h}^{(k^*)}$ as the distribution function and $\boldsymbol{\chi}$ as the fitting residual.

MATLAB code of the TRAIn method

Example:

dSet = linspace(dMin, dMax, dGrids); (linearly equally spaced discrete *D* values)

or dSet = logspace(log10(dMin), log10(dMax), dGrids); (logarithmically equally spaced discrete *D* values)

kMax = 1000; tau = 1.02; [h, res, eps] = demoTRAIn(dSet, pSet, sSet, kMax, tau);

```
function [distFun,fittRes,noisLev] ...
    = demoTRAIn(difCoef,expPara,expData,iterMax,termFac)

%                Trust-Region Algorithm for the Inversion (TRAIn)
%                          of Molecular Diffusion NMR Data
%
%                             Kaipin Xu and Shanmin Zhang*
%
%    Physics Department and Shanghai Key Laboratory of Magnetic Resonance,
%    East China Normal University, Shanghai 200062, China
%
%                             E-mail: shanminz@hotmail.com
%
%          This function enables the inversion of molecular diffusion NMR
%     data via the trust-region algorithm.  The trust-region subproblem
%     is approximately solved via the truncated conjugate gradient method.
%     The final solution of distribution function is adopted according to
%     Morozov's discrepancy principle.
%
%                                                          Kaipin Xu, June 2013
%
% Inputs:
%
%    <difCoef> Set of discrete diffusion coefficient values
%    <expPara> Set of experimental parameters
%    <expData> Set of experimental data
%    <iterMax> Maximum number of iterations
%    <termFac> Termination factor a little bit larger than 1
%
% Outputs:
%
%    <distFun> Final solution of the distribution function
%    <fittRes> Vector of the fitting residual: s-C*h
%    <noisLev> Estimated noise level of the data set

% Check inputs
if ~isreal(difCoef) || any(difCoef) <= 0
    error('Diffusion coefficient values must be real and positive.')
elseif ~isreal(expPara) || any(expPara) < 0
    error('Experimental parameters must be real and non-negative.')
elseif ~isreal(expData)
    error('Experimental data must be real.')
elseif ~isreal(iterMax) || ~isscalar(iterMax) ...
        || min(iterMax) < 1 || min(iterMax) ~= round(min(iterMax))
    error('Maximum number of iterations must be a positive interger.')
elseif ~isreal(termFac) || ~isscalar(termFac) || min(termFac) <= 1
    error('Termination factor must be a real scalar larger than 1.')
end
if size(difCoef,1) == 1 && size(difCoef,2) > 1
    d = difCoef';
elseif size(difCoef,1) > 1 && size(difCoef,2) == 1
    d = difCoef;
else
    error('Diffusion coefficient set input error.')
end
if size(expPara,1) == 1 && size(expPara,2) > 1
    p = expPara';
elseif size(expPara,1) > 1 && size(expPara,2) == 1
    p = expPara;
else
    error('Experimental parameter set input error.')
end
if size(expData,1) == 1 && size(expData,2) == length(p)
    s = expData';
elseif size(expData,1) == length(p) && size(expData,2) == 1
    s = expData;
else
    error('Sizes of experimental parameter and data sets must match.')
end
% Initialize the m*n coefficient matrix C
m = length(p); n = length(d); C = zeros(m,n);
```

```
for i = 1:m
    for j = 1:n
        C(i,j) = exp(-d(j)*p(i));
    end
end
% Estimate the noise level of the data set via the NNLS algorithm
hNNLS = lsqnonneg(C,s); noisLev = sqrt((C*hNNLS-s)'*(C*hNNLS-s));
% Initialize the main loop
h = 1e-6*ones(n,1)*min(abs(s))/n; eta = sqrt(h); subMax = 100;
radius0 = .01*sqrt(sqrt(s'*s)); radius = radius0; sSub0 = zeros(n,1);
thr0 = .01; thr1 = .25; thr2 = .75; mul1 = .5; mul2 = 2;
% Calculate CTC, CTs, the diagonal matrix and the objective function
CTC = C'*C; CTs = C'*s; Q = diag(eta); f = (C*h-s)'*(C*h-s);
% Start main loop iteration
iter = 0; subSum = 0; exitMain = false; tic
while iter < iterMax && ~exitMain
    % Compute the gradient vector and the Hessian matrix
    grad = 4*Q*(CTC*h-CTs); Hess = 8*Q*CTC*Q;
    % Initialize the sub loop
    iter = iter+1; iterSub = 0; exitSub = false;
    sSub = sSub0; gSub = grad; dSub = -gSub;
    % Start sub loop iteration
    while iterSub < subMax && ~exitSub
        subSum = subSum+1; iterSub = iterSub+1;
        kappa = dSub'*Hess*dSub;
        if kappa <= 0
            exitSub = true;
        else
            alpha = -gSub'*dSub/kappa; sSub1 = sSub+alpha*dSub;
            if sSub1'*sSub1 >= radius^2
                exitSub = true;
            else
                gSub1 = gSub+alpha*Hess*dSub;
                beta = (gSub1'*gSub1)/(gSub'*gSub);
                dSub = -gSub1+beta*dSub; sSub = sSub1; gSub = gSub1;
            end
        end
        if exitSub
            lambda = (sqrt((sSub'*dSub)^2-(dSub'*dSub) ...
                *(sSub'*sSub-radius^2))-sSub'*dSub)/(dSub'*dSub);
            sSub = sSub+lambda*dSub;
        end
    end
    eta1 = abs(eta+sSub); h1 = eta1.^2; f1 = (C*h1-s)'*(C*h1-s);
    % Update the trust-region radius and prepare for next iteration
    rho = (f1-f)/(grad'*sSub+sSub'*Hess*sSub/2);
    if rho > thr0
        eta = eta1; f = f1; h = h1; Q = diag(eta);
    end
    if rho <= thr1
        radius = max(mul1*radius,1e-12*radius0);
    elseif rho >= thr2
        radius = min(mul2*radius,radius0);
    end
    % Check for termination
    if sqrt(f) <= termFac*noisLev
        exitMain = true;
    end
    time = toc; clc, disp(' ')
    fprintf('Number of iterations runned (k): %i\n',iter)
    fprintf('Residual sum of squares (RSS): %1.4e\n',f)
    fprintf('Residual norm / noise level: %1.4e\n',sqrt(f)/noisLev)
    fprintf('Trust-region radius: %1.4e\n',radius)
    fprintf('Computational time (sec): %4.2f\n',time), disp(' ')
end
if exitMain
    disp('Inversion accomplished.')
elseif iter == iterMax
    disp('Maximum number of iterations exceeded.')
end
if iterMax < 100
    disp('Warning: maximum number of iterations may be insufficient.')
end
if termFac < 1.001
    disp('Warning: termination factor may be too small.')
elseif termFac > 2
    disp('Warning: termination factor may be too large.')
end
h(h < 1e-6*max(h)) = 0; fittRes = s-C*h; distFun = h; disp(' ')
end
```

Example parameters and data

Parameter set (in units of s/m^2)

pSet = [28010658.3708; 112042633.483; 252095925.337; 448170533.933;
700266459.27; 1008383701.35; 1372522260.17; 1792682135.73; 2268863328.04;
2801065837.08; 3389289662.87; 4033534805.4; 4733801264.67; 5490089040.68;
6302398133.43; 7170728542.93; 8095080269.16; 9075453312.14; 10111847671.9;
11204263348.3; 12352700341.5; 13557158651.5; 14817638278.2; 16134139221.6;
17506661481.8; 18935205058.7; 20419769952.3; 21960356162.7; 23556963689.8;
25209592533.7; 26918242694.3; 28682914171.7; 30503606965.8; 32380321076.6;
34313056504.2; 36301813248.6; 38346591309.6; 40447390687.4; 42604211382.0;
44817053393.3; 47085916721.3; 49410801366.1; 51791707327.6; 54228634605.9;
56721583200.9; 59270553112.6; 61875544341.1; 64536556886.3; 67253590748.3;
70026645927.0; 72855722422.5; 75740820234.7; 78681939363.6; 81679079809.3;
84732241571.7; 87841424650.8; 91006629046.7; 94227854759.4; 97505101788.8;
100838370135.0; 104227659798.0; 107672970777.0; 111174303074.0; 114731656687.0]

Data set (arbitrary units)

sSet = [1.0; 0.99432; 0.98441; 0.97055; 0.95335; 0.93251; 0.91015; 0.88289;
0.85481; 0.82427; 0.79231; 0.75863; 0.72437; 0.68822; 0.65322; 0.61758;
0.58152; 0.54632; 0.51193; 0.47705; 0.4452; 0.41285; 0.38173; 0.35166;
0.32432; 0.29734; 0.27207; 0.24885; 0.22554; 0.20536; 0.18676; 0.16819;
0.15134; 0.13628; 0.12156; 0.10932; 0.09735; 0.08731; 0.07675; 0.06786;
0.06072; 0.05205; 0.04653; 0.04135; 0.03624; 0.0307; 0.02805; 0.02435;
0.02039; 0.01832; 0.01595; 0.01331; 0.01101; 0.00985; 0.00808; 0.00686;
0.00614; 0.00543; 0.00471; 0.00368; 0.00236; 0.00263; 0.00149; 0.00162]

Diffusion coefficient set (in units of m^2/s)

dSet = logspace(−12, −8, 256)