Self-weighted alternating normalized residue fitting algorithm with application to quantitative analysis of excitation-emission matrix fluorescence data

Wu, Hai-Long; Nie, Jinfang; Zhang, Shurong; Yu, Yongjie; Yu, Ruqin

c0ay00300j

APPENDIX: PROGRAM WRITTEN IN MATLAB FOR SWANRF ALGORITHM

```
% XK=[X..1, X..2, ..., X..K], size: I*(J*K)
% RJ=[X.1.', X.2.', ..., X..K'], size: I*(K*J)
% epsilon is tolerance.
% I is the number of rows
% J is the number of columns
% K is the number of channels
% N is the number of components
% LFT is the loss function
% M is the iterative number
% krao is the Khatri-Rao product
% BC is the Khatri-Rao product of B and C
% AC is the Khatri-Rao product of A and C
% A(I,N),B(J,N),C(K,N)
% function P=krao(X,Y)
% [M1,N1]=size(X);
% [M2,N2]=size(Y);
% if N1==N2
%       for n=1:N1
%             P(:,n)=kron(X(:,n),Y(:,n));
%       end
% end

% ----------STEP 0----------
% decompose the cube X along I,J,K direction respectively
```

```
function [A,B,C,LFT,M]=SWANRF (XK,K,N,epsilon)


if nargin < 4
    epsilon=10*eps*norm(XK,1)*max(size(XK));
end
[I,JK]=size(XK);
J=JK/K;
XIJK=reshape(XK,I,J,K);
%size(XIJK);
XJKI=shiftdim(XIJK,1);
%size(XJKI);
XKIJ=shiftdim(XJKI,1);
%size(XKIJ);
XJ=zeros(size(K,I*J));
for j=1:J
     XJ(1:K,(j-1)*I+1:(j-1)*I+I)=XKIJ(:,:,j);
end
XI=zeros(size(J,K*I));
for i=1:I
     XI(1:J,(i-1)*K+1:(i-1)*K+K)=XJKI(:,:,i);
end


% -----------STEP 1----------
% initialize A & B and compute C
A=randn(I,N);
B=randn(J,N);
BA=krao(B,A);Wba=diag(ones(N,1)./diag(BA'*BA));
PBA= pinv(BA, epsilon)；PWba=inv(Wba);
C=XJ* PBA '*Wba* PBA *BA*PWba*BA'* PBA ';
```

```
TOL=10;
M=0;
LFT=[];
LF=0.01;
while TOL > epsilon && M < 500
    % compute A according B and C
    CB=krao(C,B);Wcb=diag(ones(N,1)./diag(CB'*CB));
    PCB= pinv(CB, epsilon)；PWcb=inv(Wcb);
    A=XK* PCB '*Wcb* PCB *CB*PWcb*CB'* PCB ';

    % normalization of A columnwisely
    for n=1:N
        A(:,n)=A(:,n)/sqrt(A(:,n)'*A(:,n));
    end

    % compute B according A and C
    AC=krao(A,C);Wac=diag(ones(N,1)./diag(AC'*AC));
    PAC= pinv(AC, epsilon)；PWac=inv(Wac);
    B=XI* PAC '*Wac* PAC *AC*PWac*AC'* PAC ';

    % normalization of B columnwisely
    for n=1:N
        B(:,n)=B(:,n)/sqrt(B(:,n)'*B(:,n));
    end

    % compute C according A and B
    BA=krao(B,A);Wba=diag(ones(N,1)./diag(BA'*BA));
    PBA= pinv(BA, epsilon)；PWba=inv(Wba);
    C=XJ* PBA '*Wba* PBA *BA*PWba*BA'* PBA ';
```

```
 % caculate loss function LFT
      LFTT=0;
      for k=1:K
          XXX(:,:,k)=A*diag(C(k,:))*B';
          LFTT=LFTT+trace((XIJK(:,:,k)-XXX(:,:,k))'*(XIJK(:,:,k)-XXX(:,:,k)));
      end
      TOL=abs((LFTT-LF)/LF);
      LFT=[LFT,LFTT];
      LF=LFTT;
      M=M+1;
end
% ------------STEP 3----------------
% post-processing to keep sign convention
[maxa,inda]=max(abs(A));
[maxb,indb]=max(abs(B));
asign=ones(N,1);
bsign=ones(N,1);
for n=1:N
     asign(n)=sign(A(inda(n),n));
     bsign(n)=sign(B(indb(n),n));
end
A=A*diag(asign);
B=B*diag(bsign);
C=C*diag(asign)*diag(bsign);
% plot A,B,C,LFT
subplot(2,2,1)
plot(1:I,A)
title('A')
subplot(2,2,2)
```

```
plot(1:J,B)
title('B')
subplot(2,2,3)
plot(1:K,C)
title('C')
subplot(2,2,4)
plot(1:M,log10(LFT))
title('M--LFT')



function a=krao(x,y)
n=size(x,2);
n1=size(y,2);
if n~=n1
    error
end
for n=1:n
    a(:,n)=kron(x(:,n),y(:,n));
end
```