Determination of optimal in-plane angle for cryo-EM

Peter Schwander UWM, Dec. 20, 2013
Revised Dec. 29, 2013
Revised Jan. 12, 2014

Preample

This write-up is based on an approach to minimize quaternial distances by Russell Fung from Dec. 18, 2013. The main difference is that I use the raw quaternions instead of their rotation axes (no angles need to be extracted first). Further I assert that each snapshot will be solely corrected by a rotation about the beam axis (no other rotation can be applied to a measured snapshot!)

Additionally projection directions are expressed by quaternion notation. A link to the SPIDER package is established by conversion the SPIDER Euler angles to quaternions together with required workflow. I also compare the method to the ones used previously. A benchmark is provided for comparison, including an alternative approach based on the minimization of atomic displacements, which was used as a reference.

Motivation

One way to detect conformational changes is to observe cryoEM images along nearby projection directions. For that the individual images need to be optimally aligned in the images plane (in-plane).

Orientational alignment in CryoEM

In cryoEM projections of a molecule parallel to the electron beam are taken. Each of these projections is a snapshot of the molecule in a particular orientation. A cryoEM alignment algorithm assigns an orientation to each of the N snapshots represented as a rotation $Q_i \quad (i = 1...N)$. The rotation is such that when applied to the molecule (active) its projection yields exactly the observed snapshot.
Note that this assignment of rotations is not unique, any set of rotations $Q_i' = Q_i Q_0 \ (i = 1...N)$ with arbitrary Q_0 is equivalent.

Finding the optimal in-plane angle

Two snapshots i, j are in the same projection direction if their rotations differ only by a rotation about the beam direction (assumed to be z-axis of the laboratory frame).

I.e. $Q_i = Q_z(\alpha) Q_j, \quad \alpha \in [0...2\pi[$

Then the snapshots can be matched by rotating snapshot j by $Q_z(\alpha) = Q_i Q_j^{-1}$.
In practice, no two snapshots will be exactly in the same projection direction. But we must try in the best way as possible. The ways to do this depends on the metric (cost function) we choose.

Optimization based on quaternion distance

We use quaternion notation, i.e. represent each rotation by a unit quaternion $Q_i = [q_{i,0}, q_{i,1}, q_{i,2}, q_{i,3}]$.

Claim: The optimal in-plane angle based on quaternion distance is given by the following simple formula.

$$2\arctan\left(\frac{q_3^i q_0^j - q_0^i q_3^j + q_2^i q_1^j - q_1^i q_2^j}{q_0^i q_0^j + q_1^i q_1^j + q_2^i q_2^j + q_3^i q_3^j}\right) \quad (1)$$

Proof of (1): The angle is obtained by setting the norm of the quaternion distance extreme.

$$F(\alpha) = \left\| Q_i - Q_z(\alpha) Q_j \right\|^2$$
$$\frac{dF(\alpha)}{d\alpha} = 0$$

In components
$Q_z(\alpha) = [\cos(\alpha/2), 0, 0, \sin(\alpha/2)]$, with $c = \cos(\alpha/2)$, $s = \sin(\alpha/2)$
$Q_z(\alpha) Q_j = [cq_0^j - sq_3^j, cq_1^j - sq_2^j, cq_2^j + sq_1^j, cq_3^j + sq_0^j]$
$Q_i - Q_z(\alpha) Q_j = [q_0^i - cq_0^j + sq_3^j, q_1^i - cq_1^j + sq_2^j, q_2^i - cq_2^j - sq_1^j, q_3^i - cq_3^j - sq_0^j]$

$$F(\alpha) = (q_0^i - cq_0^j + sq_3^j)^2 + (q_1^i - cq_1^j + sq_2^j)^2 + (q_2^i - cq_2^j - sq_1^j)^2 + (q_3^i - cq_3^j - sq_0^j)^2$$

$$\frac{dF(\alpha)}{d\alpha} = c(q_0^i q_3^j - q_3^i q_0^j + q_1^i q_2^j - q_2^i q_1^j) + s(q_0^i q_0^j + q_1^i q_1^j + q_2^i q_2^j + q_3^i q_3^j) = 0$$
i.e. $c(q_0^i q_3^j - q_3^i q_0^j + q_1^i q_2^j - q_2^i q_1^j) = -s(q_0^i q_0^j + q_1^i q_1^j + q_2^i q_2^j + q_3^i q_3^j)$

$$\frac{s}{c} = \tan(\alpha/2) = \frac{q_3^i q_0^j - q_0^i q_3^j + q_2^i q_1^j - q_1^i q_2^j}{q_0^i q_0^j + q_1^i q_1^j + q_2^i q_2^j + q_3^i q_3^j}$$

$$\alpha = 2\arctan\left(\frac{q_3^i q_0^j - q_0^i q_3^j + q_2^i q_1^j - q_1^i q_2^j}{q_0^i q_0^j + q_1^i q_1^j + q_2^i q_2^j + q_3^i q_3^j}\right)$$

First note that this result is invariant for all possible sign combinations of the quaternions ($\pm Q_i, \pm Q_j$). The double coverage of the quaternions thus vanishes completely! This surprising result is due to the fact that $F(\alpha)$ goes through a maximum, should Q_i, Q_j not be on the same hemisphere of S3.

Second, we need to discuss carefully possible singularities. The denominator will be zero when the two quaternions are orthogonal. Two cases need to be considered, depending on the value of the nominator. If it is non-zero, α can admit both $+\pi$ and $-\pi$ i.e. a difference of 2π, the same in-plane rotation!

The situation is subtle when the denominator vanishes at the same time.
To start with, the investigation can be simplified by assuming that the first quaternion is unity.

$$Q_i = [1,0,0,0]$$
$$Q_j = \left[\cos\left(\delta^j/2\right), \sin\left(\delta^j/2\right)\hat{u}_1^j, \sin\left(\delta^j/2\right)\hat{u}_2^j, \sin\left(\delta^j/2\right)\hat{u}_3^j\right]$$

The argument of the arctan function becomes

$$\frac{-q_3^j}{q_0^j} = \frac{-\sin\left(\delta^j/2\right)\hat{u}_3^j}{\cos\left(\delta^j/2\right)}$$

However the nominator will only be zero if $\hat{u}_3^j$ vanishes. In that case the angle α becomes zero. This corresponds to a rotation about an axis in the xy-plane by π. For in-plane alignment though this is a pathological case. The corresponding image pairs are then exact mirrors of each other! Of course this is not limited to the xy-plane, any rotation about an axis perpendicular to the projection direction by π has this properties.

So there are mathematically no ambiguities or pitfalls when using the quaternion coordinates in plain form.

Classification of projection directions

In order to find images along nearby projection directions, the projection directions must be extracted from the quaternions.

We characterize a projection direction by a three-dimensional unit vector $\hat{p}$ in coordinate system of the object. We know that a rotation Q transforms $\hat{p}$ to a $\hat{p}_z$, parallel to the z-axis.

In quaternion notation we can think of $\hat{p}$ as a pure vector quaternion, i.e. a quaternion with scalar part zero. The vector $\hat{p}$ is then transformed as follows

$$\hat{p}_z = Q\hat{p}Q^* .$$

The projection direction for any image is therefore given by

$$\hat{p}_j = Q_j^* \hat{p}_z Q_j$$

Or expressed in components

$$\begin{aligned}
\hat{p}_1^j &= 2\left(q_1^j q_3^j - q_0^j q_2^j\right) \\
\hat{p}_2^j &= 2\left(q_0^j q_1^j + q_2^j q_3^j\right) \qquad (3) \\
\hat{p}_3^j &= 2\left((q_0^j)^2 + (q_3^j)^2 - 1/2\right)
\end{aligned}$$

Note that the transformation $Q_j \to -Q_j$ leaves the projection direction unchanged. There is no issue with double coverage of the quaternions.

For measuring differences of projection directions we use naturally the quaternion metric.

Alignment of an ensemble

For the in-plane alignment of multiple images we need to find the optimal reference quaternion.

First we discuss the uniqueness of such a reference. A rotation about the beam axis (z-axis) does not change the cost function. Algebraically this can be seen as follows.

$$F(\alpha_j) = \left|\left|Q_z(\alpha_0)Q_r - Q_z(\alpha_j)Q_z(\alpha_0)Q_j\right|\right|^2 = \left|\left|Q_z(\alpha_0)Q_r - Q_z(\alpha_0)Q_z(\alpha_j)Q_j\right|\right|^2 = \left|\left|Q_z(\alpha_0)\left(Q_r - Q_z(\alpha_j)Q_j\right)\right|\right|^2$$
$$= \left|\left|Q_r - Q_z(\alpha_j)Q_j\right|\right|^2$$

This means that all references differing by a rotation about the z-axis are equally optimal.

So we can choose a reference that does not include a rotation about the z-axis. This can always be achieved by setting $q_3^r = 0$.
Formula (1) then simplifies to

$$\alpha_j = 2\arctan\left(\frac{-q_0^r q_3^j + q_2^r q_1^j - q_1^r q_2^j}{q_0^r q_0^j + q_1^r q_1^j + q_2^r q_2^j}\right) \qquad (2)$$

For any reference this formula provides the optimal rotation to minimize the quaternion distance.

Now the question remains what is the optimal reference?

Using the quaternion metric for the projection directions we get the cost function

$$G(p_r) = \sum_{j=1}^{n} \left|\left| p_r - \hat{p}_j \right|\right|^2$$

This will be optimal if we take the average projection direction.

$$\overline{p}^r = \frac{1}{n}\sum_{j=1}^{n} \hat{p}_j \, .$$

In general, this will not result in a unit vector, but we can always normalize by a multiplication of a constant provided the projection directions are confined to the one hemisphere, i.e.

$$\hat{p}^r = \frac{\overline{p}^r}{\left|\left|\overline{p}^r\right|\right|}$$

The reference quaternion must then rotate $\hat{p}_r$ to parallel to the z-axis. This is achieved by a rotation $\delta = \arccos(\hat{p}_r \hat{p}_z) = \arccos(p_3^r)$ about an axis $\hat{v} = [p_2^r, -p_1^r, 0] \Big/ \sqrt{\left(p_1^r\right)^2 + \left(p_2^r\right)^2}$, and therefore

$$Q^r = \left[\cos(\delta/2), \sin(\delta/2) p_2^r, -\sin(\delta/2) p_1^r, 0\right]$$

Using the identities $\sin(\delta/2) = \sqrt{\frac{1-\cos(\delta)}{2}}$, $\cos(\delta/2) = \sqrt{\frac{1+\cos(\delta)}{2}}$ for $\delta \in [0, \pi]$, and $\cos(\delta) = p_3^r$ we derive

$$Q^r = \left[\sqrt{\frac{1+p_3^r}{2}}, \sqrt{\frac{1-p_3^r}{2\left(1-\left(p_3^r\right)^2\right)}} p_2^r, -\sqrt{\frac{1-p_3^r}{2\left(1-\left(p_3^r\right)^2\right)}} p_1^r, 0\right] = \left[\sqrt{\frac{1+p_3^r}{2}}, \frac{1}{\sqrt{2\left(1+p_3^r\right)}} p_2^r, \frac{-1}{\sqrt{2\left(1+p_3^r\right)}} p_1^r, 0\right]$$

and finally

$$Q^r = \frac{1}{\sqrt{2\left(1+p_3^r\right)}}\left[1+p_3^r, p_2^r, -p_1^r, 0\right]$$

So the optimal in-plane angle becomes

$$\alpha_j = 2\arctan\left(\frac{-(1+p_3^r)q_3^j - p_1^r q_1^j - p_2^r q_2^j}{(1+p_3^r)q_0^j + p_2^r q_1^j - p_1^r q_2^j}\right)$$

Note that the normalization pre-factor canceled in the expression.

SPIDER Euler Angles

The Euler angles in SPIDER & Web are defined as three successive rotations in a right hand coordinate system.

1. First, the object is rotated CLOCKWISE around the Z-axis (angle φ)
2. Then it is rotated CLOCKWISE around the original Y-axis (angle θ)
3. Finally, it is rotated CLOCKWISE around the original Z-axis (angle ψ).

All rotations are done around axes of the original SPACE coordinate system and direction of rotation is determined by looking to the origin.

(Excerpt from http://spider.wadsworth.org/spider_doc/spider/docs/euler.html).

Expressed in the language of quaternions

$$Q = Q_z(-\psi)Q_z(-\theta)Q_z(-\varphi)$$
$$= [\cos(\psi/2),0,0,-\sin(\psi/2)][\cos(\theta/2),0,0,-\sin(\theta/2)][\cos(\varphi/2),0,0,-\sin(\varphi/2)]$$

The negative signs are due to the clockwise convention.

When this conversion of quaternions is used for in-plane angle determination, it does not give the correct result. The quaternions require inversion. I interpret this as follows. We can either think of the rotation as the operation that acts on the molecule to generate the observed image, or as the action that undoes this operation for reconstruction, i.e. its inverse The SPIDER orientation algorithm seems to assign the later.

A final issue pointed out by Robert Langlois is that the ψ angle should be negated due to a convention used for 2D images.

Altogether the conversion formula then reads

$$Q = Q_z(\varphi)Q_z(\theta)Q_z(-\psi)$$
$$= [\cos(\varphi/2),0,0,\sin(\varphi/2)][\cos(\theta/2),0,\sin(\theta/2),0][\cos(\psi/2),0,0,-\sin(\psi/2)]$$

Once the conversion is done all can be done with quaternion algebra consistently.

Workflow

The workflow to calculate the CTF corrected distances for different projection direction consists of the following three steps

(I) Generate quaternions from SPIDER angles

(II) Classify projection directions

(III) Calculate CTF-corrected distances for each projection direction

See appendix for a full MATLAB script of the workflow.

Comparison with the Projection Method used previously

Previously I used a projection method for the global embedding. I estimated the in-plane angle by projection to a rotation about the z-axis described as follows.

Calculate $R_{\tilde{z}} = R_i R_j^T$, and then projecting $R_{\tilde{z}}$ to a z-axis rotation.
In quaternion notation this reads

$$\begin{aligned} Q_{\tilde{z}} = Q_i Q_j^{-1} = [&-q_0^i q_0^j - q_1^i q_1^j - q_2^i q_2^j - q_3^i q_3^j, \\ &q_0^i q_1^j - q_1^i q_0^j + q_2^i q_3^j - q_3^i q_2^j, \\ &q_0^i q_2^j - q_1^i q_3^j - q_2^i q_0^j + q_3^i q_1^j, \\ &q_0^i q_3^j + q_1^i q_2^j - q_2^i q_1^j - q_3^i q_0^j] \end{aligned}$$

Projecting to z-axis rotation yields.

$$Q_z = A[-q_0^i q_0^j - q_1^i q_1^j - q_2^i q_2^j - q_3^i q_3^j, 0, 0, q_0^i q_3^j + q_1^i q_2^j - q_2^i q_1^j - q_3^i q_0^j],$$

where A is a non-zero normalization constant for preserving quaternion unit length.

Identification of the rotation angle from the quaternion reveals

$$\cos(\alpha'/2) = A(-q_0^i q_0^j - q_1^i q_1^j - q_2^i q_2^j - q_3^i q_3^j)$$
$$\sin(\alpha'/2) = A(q_0^i q_3^j + q_1^i q_2^j - q_2^i q_1^j - q_3^i q_0^j)$$

$$\alpha' = 2\arctan\left(\frac{q_3^i q_0^j - q_0^i q_3^j + q_2^i q_1^j - q_1^i q_2^j}{q_0^i q_0^j + q_1^i q_1^j + q_2^i q_2^j + q_3^i q_3^j}\right)$$

This result is identical to expression (1). Therefore the quaternion distance optimization method and the projection methods are identical. This equivalence has not been evident before.

Benchmarks

Pairwise in-plane angles were estimated by different methods for 800,000 random rotations (Lovisolo), classified into 100 projection directions on the two-sphere (Lovisolo). This corresponds to a patch diameter of about ten Shannon angles for the Ribosome (0.04 radians).

As a reference I used an estimate obtained from a random spherical assembly of 50,000 atoms. The in-plane angle was determined by minimizing the projected atomic coordinates.

The following figures show the error histograms for the 100 patches of projection directions for the different methods. The histograms are sorted from the north pole (top left) to the south pole (bottom right). Note the different ranges in the histograms.

Fig. 1 Direct Euler angles. The method performs reasonable only near the equator.

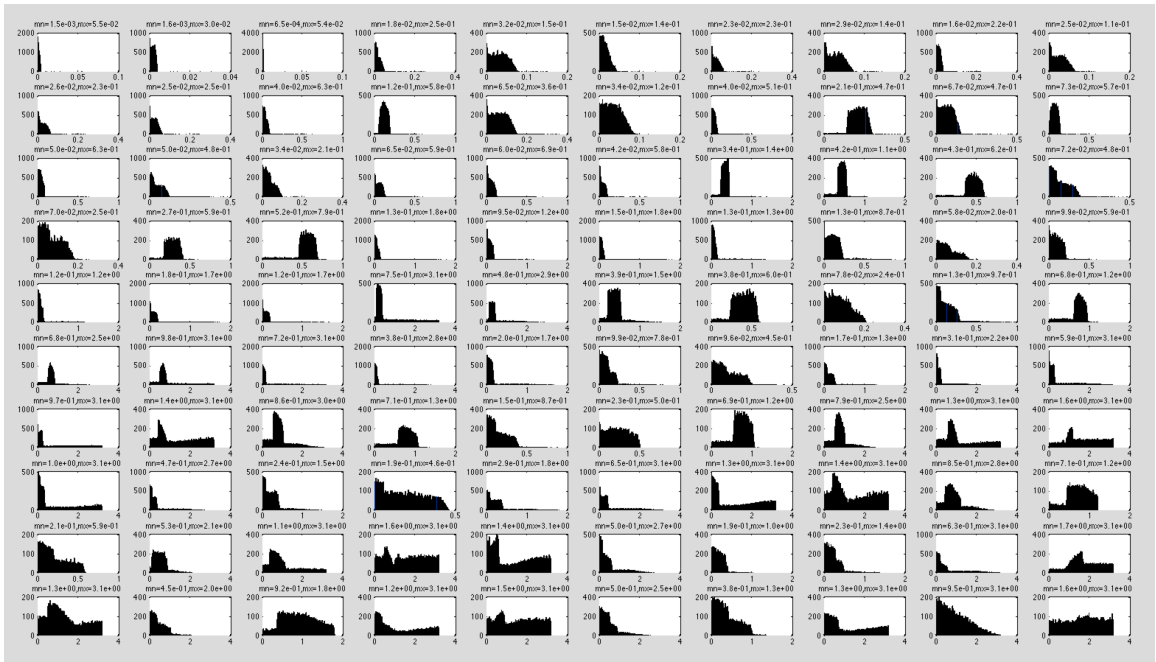

Fig.2 Projection method without a reference. The method performs well only near the north pole.

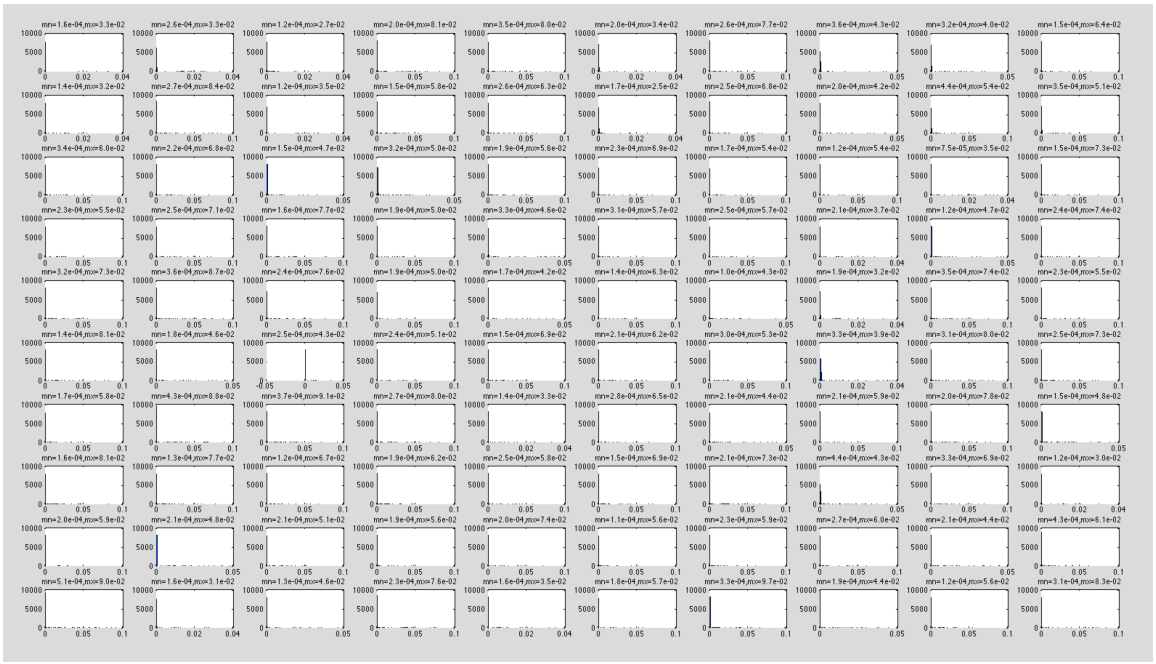

Fig. 3 Projection method with one of the snapshots as reference. This works well with average errors about two orders of magnitude below Shannon, but shows some spurious effects.

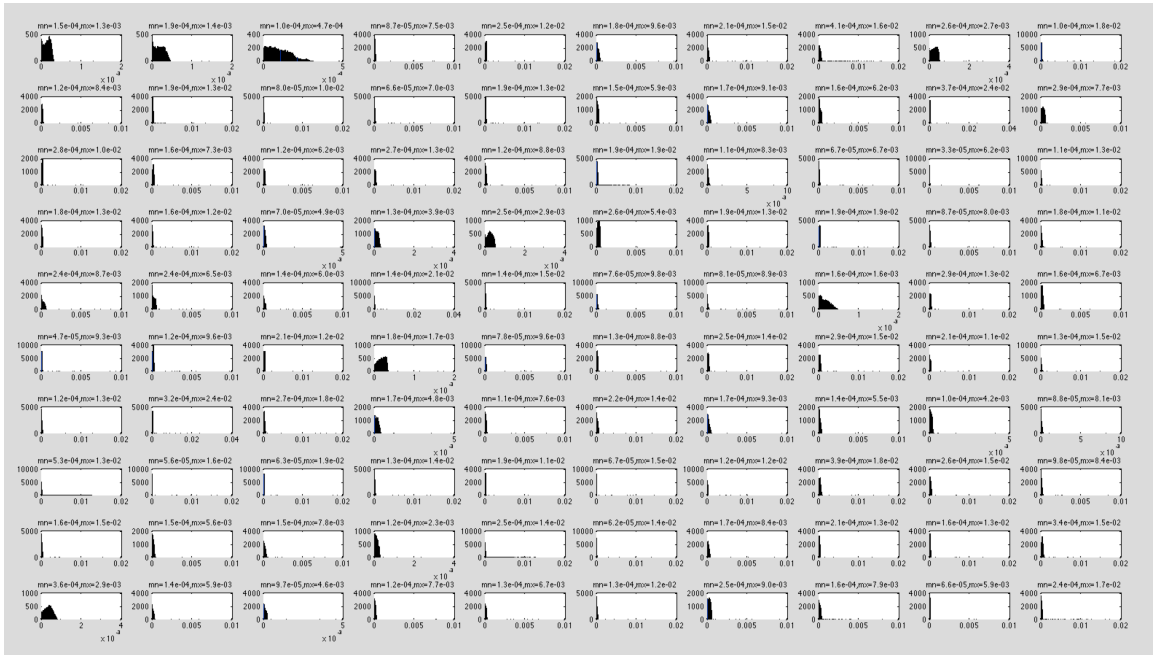

Fig. 4: Projection method with the average as reference. This works well with average errors about two orders of magnitude below Shannon. Shows some spurious outliers.

Fig. 5: Quaternion distance optimization. This works well with average errors about two orders of magnitude below Shannon. Even the maximum deviation is two orders of magnitude away from Shannon.

Appendix Source Code

```
% Workflow
% (I) Generate quaternions from SPIDER angles

spider = importdata('refine_005.tls');

% SPIDER Euler angles
phi = spider.data(:,5)*pi/180;
theta = spider.data(:,4)*pi/180;
psi = spider.data(:,8)*pi/180;

df = spider.data(:,20);

zros = zeros(size(phi));

% important q's must be 4xn
qz =  [cos(phi/2),zros,zros,sin(phi/2)]';
qy =  [cos(theta/2),zros,sin(theta/2),zros]';
qzs = [cos(psi/2),zros,zros,-sin(psi/2)]';  % negative sign do to 2D
image convention in SPIDER


q = qMult_bsx(qzs,qMult_bsx(qy,qz));

clear spider phi theta psi zros qz qy  qzs

% Double number of data points by augmentation.
% optional
%

% (II) Classify projection directions
Ng = 1500;

[S20 it] = distribute3Sphere(Ng);
it

nS = size(q,2);

S2 = 2*[q(2,:).*q(4,:)-
q(1,:).*q(3,:);q(1,:).*q(2,:)+q(3,:).*q(4,:);q(1,:).^2+q(4,:).^2-0.5]';

scatter3(S2(1:100:end,1),S2(1:100:end,2),S2(1:100:end,3)), axis vis3d

[IND NC] = classS2(S2, S20);

[NCS N] = sort(NC,'descend');

C = cell(Ng,4);

for i = 1:Ng
   tmp = find(IND == N(i));

   C{i,1} = i; % class number sorted
```

```
   C{i,2} = N(i); % class ID (Lovisolo)
   C{i,3} = numel(tmp);
   C{i,4} = tmp;

end

save('data','q','df','C');

% (III) Calculate CTF-corr. distances for each proj. direction

emPar.Cs = 2.26;         % Spherical aberration [mm]
emPar.EkV = 300;         % Acceleration voltage [kV]
emPar.gaussEnv = Inf;    % Gaussian damping envelope [A^-1]
emPar.nPix = 125;        % lateral pixel size
emPar.dPix = 3;          % Pixel size [A]
emPar

options.verbose = 0;
options.avgOnly = 0;
options.visual = 0;
options

sigmaH = 1.5;

imgFileName = '/state/partition1/tmp/ribosome800k.dat';


% setup cluster jobs
jm = findResource('scheduler', 'type',
'jobmanager','name','default_jobmanager','LookupURL','riemann.phys.uwm.
edu');
job = createJob(jm,'MaximumNumberOfWorkers', 38);

% set dependencies
set(job,'FileDependencies',{'/home/pschwan/peter/Frank_3D/Utilities/get
DistanceCTF_local6.m','/home/pschwan/peter/Frank_3D/Utilities/annularMa
sk.m','/home/pschwan/peter/Frank_3D/Utilities/ctemh_cryoFrank.m'});
set(job, 'RestartWorker', true);

for j = 1:Ng

   j
   ind = C{j,4};
   numel(ind)
   outFile =
sprintf('/state/partition1/tmp/ribosome800k_sigmaH_15_PD_%d',j)

   obj = createTask(job, @getDistanceCTF_local6, 1, {ind, q, df, emPar,
sigmaH, imgFileName, outFile, options});

end

submit(job);
```

```
function [D, imgAvg, imgAvgFlip] = getDistanceCTF_local6(ind, q, df,
emPar, sigmaH, imgFileName, outFile, options)
% [D, imgAvg, imgAvgFlip] = getDistanceCTF_local6(ind, q, df, emPar,
sigmaH, imgFileName, outFile, options)
% Calculates squared Euclidian distances for snapshots in similar
% projection directions. Includes CTF correction of microspope.
%
% % Input parameters
% ind           Global image indexes
%
% q             Rotation quaternions of all images, 4xN
%
% df            Defocus values of all images
%
% emPar         Common microscope parameters
%                 emPar.Cs          Spherical aberration [mm]
%                 emPar.EkV         Acceleration voltage [kV]
%                 emPar.gaussEnv    Gaussian damping envelope [A^-1]
%                 emPar.nPix        lateral pixel size
%                 emPar.dPix        Pixel size [A]
%
% sigmaH        Image filter Gaussian width [pixel]
%
% imgFileName   Image file with all raw images
%
% outFile       Output filename
%
% options       Options field
%                 options.verbose   Gives verbose information
%                 options.avgOnly   Skips calculation of distances
%                 options.visual    Displays averaged images
%
% Copyright (c) UWM, Peter Schwander 2014
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

version = 'getDistanceCTF_local6, V 0.9';

% define default options, if not set
if ~isfield(options,'verbose')
   options.verbose = false;
end
if ~isfield(options,'avgOnly')
   options.avgOnly = false;
end
if ~isfield(options,'visual')
   options.visual = false;
end

nS = size(ind,1);

% initialize arrays
q = q(:,ind);
df = df(ind);
Psis = nan(nS,1);
Nom = nan(nS,1);
Dnom = nan(nS,1);
```

```
imgAvg = zeros(emPar.nPix,emPar.nPix);
imgAvgFlip = zeros(emPar.nPix,emPar.nPix);
y = zeros(emPar.nPix^2, nS);
fy = complex(zeros(emPar.nPix,emPar.nPix,nS));
CTF = zeros(emPar.nPix,emPar.nPix, nS);
D = zeros(nS,nS);

msk = annularMask(0,emPar.nPix/2,emPar.nPix,emPar.nPix);

% maps images to file
m = memmapfile(imgFileName,'Format',{'single',emPar.nPix^2,'y'});

% read images
for iS=1:nS
   y(:,iS) = m.Data(ind(iS)).y;
end

% filter images, if sigmaH~= 0
if sigmaH ~= 0
   if options.verbose
       display(sprintf('Filtering images sigmaH = %g', sigmaH));
   end
   % filter images
   nPK = 10*sigmaH; % width of filter kernel
   h = [ 0 : nPK ] - nPK / 2;
   h = exp( - .5 * h .^2 / sigmaH ^ 2 ) / sigmaH / sqrt( 2 * pi );
   for iS=1:nS
       img = reshape(y(:,iS),emPar.nPix,emPar.nPix);
       % filter patterns by a Gaussian filter (sigmaH)
       img = conv2( h, h, img, 'same' );
       y(:,iS) = img(:);
   end
end

% Calculate average projection directions
PDs = 2*[q(2,:).*q(4,:) - q(1,:).*q(3,:); ...
   q(1,:).*q(2,:) + q(3,:).*q(4,:); ...
   q(1,:).^2 + q(4,:).^2 - ones(1,nS)/2 ];

% reference PR is the average
PD = sum(PDs,2);

% make it a unit vector
PD = PD/sqrt(sum(PD.^2));

% setup mesh for CTF
if mod(emPar.nPix,2)
   [X, Y] = meshgrid(-(emPar.nPix-1)/2:(emPar.nPix-1)/2,-(emPar.nPix-
1)/2:(emPar.nPix-1)/2); % emPar.nPix odd
else
   [X, Y] = meshgrid(-emPar.nPix/2:emPar.nPix/2-1,-
emPar.nPix/2:emPar.nPix/2-1); % emPar.nPix even
end
Q = (1/(2*emPar.dPix*(emPar.nPix/2)))*sqrt(X.^2+Y.^2);
```

```
% rotate images and perform FFT
for iS=1:nS

   % Quaternion approach
   s = -(1+PD(3))*q(4,iS) - PD(1)*q(2,iS) - PD(2)*q(3,iS);
   c =  (1+PD(3))*q(1,iS) + PD(2)*q(2,iS) - PD(1)*q(3,iS);

   Psi = 2*atan(s/c);  % note that the Psi are in the interval [-pi,pi]

   if isnan(Psi) % this happens only for a rotation of pi about an axis
perpendicular to the projection direction
       Psi = 0;
   end

   Psis(iS) = Psi;  % save image rotations

   Dnom(iS) = c;  % save denominator
   Nom(iS) = s;   % save nominator


   if options.verbose
       display(sprintf('Image rotation Psi = %g', Psi));
   end


   img = reshape(y(:,iS),emPar.nPix,emPar.nPix).*msk;
   img = imrotate(img,(180/pi)*Psi,'bilinear','crop');

   imgAvg = imgAvg + img;

   CTF(:,:,iS) =
ifftshift(ctemh_cryoFrank(Q,[emPar.Cs,df(iS),emPar.EkV,emPar.gaussEnv])
); % ifftshift is correct!
   fy(:,:,iS) = fft2(img);

   imgAvgFlip = imgAvgFlip + ifft2(sign(CTF(:,:,iS)).*fy(:,:,iS));

end
clear y;


imgAvg = imgAvg/nS;
imgAvgFlip = imgAvgFlip/nS;


if options.visual
   subplot(1,2,1);
   imagesc(imgAvg), axis image
   title('Averaged images');
   subplot(1,2,2);
   imagesc(imgAvgFlip), axis image
   title('Averaged phase-flipped images');
   drawnow;
end
```

```
% reshape
fy = reshape(fy,emPar.nPix^2,nS);
CTF = reshape(CTF,emPar.nPix^2,nS);

if ~options.avgOnly
   for iS = 1:nS % sum over all samples

       if options.verbose
           display(sprintf('Image %d/%d', iS, nS));
       end

       fy0 = fy(:,iS);

       CTF0 = CTF(:,iS);

       for iN = iS+1:nS; % calculate only upper part of distance matrix
           D(iN,iS) = sum(abs(CTF0.*fy(:,iN) - CTF(:,iN).*fy0).^2); %
squared distances!
       end

   end
   D = D+D'; % restore full distance matrix
end

% save results to file
save(outFile,'D','ind','q','df','emPar','sigmaH','PD','PDs','Psis','Dno
m','Nom','imgAvg','imgAvgFlip','version', 'options','-v7.3');

end
```

```
function y = ctemh_cryoFrank(k,params)
% from Kirkland, adapted for cryoEM (SPIDER) by P. Schwander
% Version V 1.1
% Copyright (c) UWM, Peter Schwander 2010
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
% Here, the damping envelope is characterized by a single parameter B
% see J. Frank
% params(1)   Cs in mm
% params(2)   df in Angstrom, a positive value is underfocus
% params(3)   Electron energy in keV
% params(4)   B in A-2

% Note: we assume |k| = s

Cs = params(1)*1.0e7;
df = params(2);
kev = params(3);
B = params(4);
mo = 511.0;
hc = 12.3986;
wav = (2*mo)+kev;
wav = hc/sqrt(wav*kev);
w1 = pi*Cs*wav*wav*wav;
w2 = pi*wav*df;
k2 = k.*k;
sigm = B/sqrt(2*log(2)); % B is Gaussian Env. Halfwidth
wi = exp(-k2/(2*sigm^2));
wr = (0.5*w1.*k2-w2).*k2; % gam = (pi/2)Cs lam^3 k^4 - pi lam df k^2

y = (sin(wr)-0.1*cos(wr)).*wi;

end
```

```
function p = qMult_bsx(q,s)
% for any number of quaternions N
% q is 4xN or 4x1
% s is 4xN or 4x1

q0 = q(1,:);
qv = q(2:4,:);
s0 = s(1,:);
sv = s(2:4,:);

c = [bsxfun(@times,qv(2,:),sv(3,:))-bsxfun(@times,qv(3,:),sv(2,:));
    bsxfun(@times,qv(3,:),sv(1,:))-bsxfun(@times,qv(1,:),sv(3,:));
    bsxfun(@times,qv(1,:),sv(2,:))-bsxfun(@times,qv(2,:),sv(1,:))];
p = [bsxfun(@times,q0,s0)-
sum(bsxfun(@times,qv,sv));bsxfun(@times,q0,sv)+bsxfun(@times,s0,qv)+c];

end
```

```
function [IND NC] = classS2(S2, S20)
% function [ IND NC ] = untitled(S2, S20)
% Delaunay Triangulation method
tic

dt = DelaunayTri(S20);
IND = nearestNeighbor(dt, S2);

% this is the number of snapshots contained in each class
NC = histc(IND,1:size(S20,1));

toc

end
```