1 General Idea

In Cryo-EM we have many "snpashots" of a particle in different orientations. We are proposing that once the orientation is determined, that one can use a "double filtering" technique so that each image is filtered against every other image so that only the shared parts of the image are kept. Distance calculations can then be performed on these filtered images

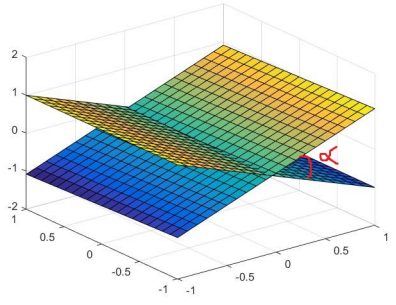
The basis for the application of this idea lies in the central slice theorem which states that any projection of an object is the equivalent of a "slice" through Fourier space of the object which goes through the origin. For diffraction images, these are planes in the 3-dimensional Fourier space.

Quaternion Approach Suppose we have two images with their associated quaternions. To find the proper filter to apply to an image, we need the angle between the image planes in Fourier space, along with the in-plane rotation of an image. The angle between the projection directions should be the same angle between the normal of the planes in Fourier space.

$$q^i = q_0^i + \boldsymbol{q^i} = [q_0^i \; q_1^i \; q_2^i \; q_3^i \; q_4^i]$$

$$q^j = q_0^j + \boldsymbol{q^j} = [q_0^j \; q_1^j \; q_2^j \; q_3^j \; q_4^j]$$

Angle between planes The angle between these two planes in Fourier space should be the angle between the projection directions (which are the normals to the planes in fourier space). This should simply be the cos^{-1} of the vector components.



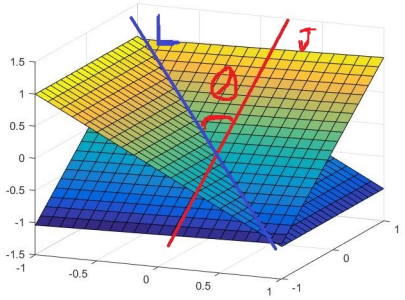
$$\alpha = \measuredangle(PD_1, PD_2)$$
$$\text{where } PD = [2(q_1q_3 - q_0q_2) \quad 2(q_0q_1 + q_2q_3) \quad 2((q_0)^2 + (q_3)^3 - 1/2)]$$
$$\alpha = cos^{-1}(PD_1 \cdot PD_2)$$

Angle of line of intersection Second, we need to find the angle the intersection of the planes forms relative to the vertical of one plane. The three-dimensional direction of the line is given by the cross-product of the normals to each plane. Again, I think this might simply be:

$$L = \boldsymbol{PD_1} \times \boldsymbol{PD_2}$$

New Approach Since this line L necessarily lies in the plane, the angle θ can be found by simply calculating the angle between L and a transformed unit direction vector. This process follows the approach taken to find PD.

$$\text{because } \hat{p_j} = q^* \hat{p_z} q \text{ it follows that}$$
$$J = q^* \hat{p_y} q$$
$$J = q^*[0\ 0\ 1\ 0]q$$
$$J = [2(q_1q_2 + q_0q_3) \quad 2(q_0^2 + q_2^2 - 1/2) \quad 2(q_2q_3 - q_0q_1)]$$
$$\theta = cos^{-1}(L \cdot J)$$



Then rotate an Fourier image by this angle.

It should be noted that if two images share the same projection direction, their cross product will be 0. This case is dealt with by simply replacing zero vectors in the "L" matrix with a corresponding "J" vector. This simply means that if the planes lie ontop of eachother, then $\theta = cos^{-1}(J_1 \cdot J_2)$

Finding the Gaussian Finally, we now have a parameter α and a rotated Fourier image. We can now simply weight the values off of the centerline of this image. The normalized Gaussian should be near 0 at the ends of the picture, and 1 at the center. A form for the Gaussian could be:

$$f = e^{\,k \cdot \frac{(x - .5 n_{pixels})^2}{2\beta^2}}$$

$$\text{where } \beta = \frac{\pi}{2\alpha}$$

If we choose k=3, and use a fraction of α, we see plots like

Code This is a snip of what code might look like to find the angles.

Create 20 Normed 4 Vectors

```
clear
num_proj=20;
q = randn(num_proj,4);
q = q./repmat(sqrt(sum(q.^2,2)),1,4);              %norm 4 vector
```

Suppose we have our q's

```
proj_dir = [2*(q(:,2).*q(:,4)-q(:,1).*q(:,3)) ...   %This form comes from
       2*(q(:,1).*q(:,2)+q(:,3).*q(:,4)) ...        %the documentation
       2*(q(:,1).^2 + q(:,4).^2 - .5)];

alphas = acos(proj_dir*proj_dir');                  %this outer product
                                                    %gives a [20 x 20]
                                                    %matrix

for i=1:num_proj                                    %Take the cross product
    for j=1:num_proj                                %of every proj pair
        L(i,j)={cross(proj_dir(i,:),proj_dir(j,:))};
    end
end

J = [2*(q(:,2).*q(:,3)+q(:,1).*q(:,4)) ...          %Find the "vertical"
     2*(q(:,1).^2 + q(:,3).^2 - .5) ...             %direction for each image
     2*(q(:,3).*q(:,4) + q(:,1).*q(:,2))];

 for i=1:num_proj                                   %calculate theta between
     for j=1:num_proj                               %every image
         theta(i,j)=acos(J(i,:)*L{i,j}');
     end
 end

% This script calculates the two angles between every image pair. It should
% be noted that the angles here are complex, which is something I didn't
% forsee happening and will consider
```