Study of Single Particle 3D Reconstruction in cryoEM

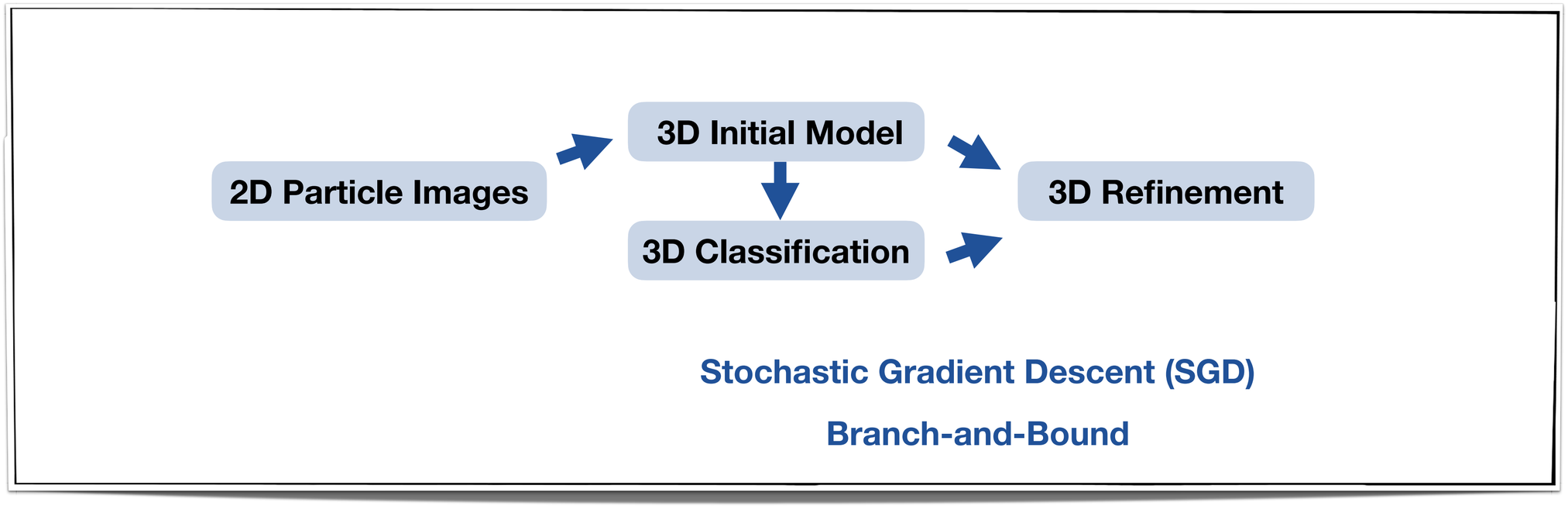
Workshop

Yu-Hsiang Lien 連昱翔

Outline

- Review of single particle 3D reconstruction — cryoSPARC
- Review of Quaternion
- Study of 3D rotation in different representation

Single Particle 3D Reconstruction



- Two algorithms included in cryoSPARC:
  - Stochastic Gradient Descent (SGD)
  - Branch-and-Bound maximum likelihood optimization

Optimization of Objective Function

- The aim of the optimization is to find the 3D structures $V_1 \cdots V_K$ that best explain the observed images $X_1 \cdots X_N$ by marginalizing over class assignment j and the unknown pose variable ϕ_i (3D rotation + 2D translation) for each single particle image.
- The optimization is formulated as the maximization of an **objective function** $f(\mathbf{V})$:

$$\begin{aligned}\arg\max_{V_{1\cdots K}} \log p(V_1, \cdots, V_K | X_1, \cdots, X_N) &= \arg\max_{V_{1\cdots K}} \log p(X_1, \cdots, X_N | V_1, \cdots, V_K) + \log p(V_1, \cdots, V_K) \\ &= \arg\max_{V_{1\cdots K}} \sum_{i=1}^{N} \log p(X_i | \mathbf{V}) + \sum_{j=1}^{K} \log p(V_j) \equiv f(\mathbf{V}) \\ &= \arg\max_{\mathbf{V}} f(\mathbf{V})\end{aligned}$$

$$p(X_i | \mathbf{V}) = \sum_{j=1}^{K} \pi_j p(X_i | V_j) \equiv U_i \qquad p(X_i | V_j) = \int p(X_i | \phi, V_j) p(\phi) \mathrm{d}\phi$$

Objective function

$$f(\mathbf{V}) = \sum_{i=1}^{N} \log \left(\sum_{j=1}^{K} \pi_j \int p(X_i | \phi, V_j) p(\phi) \mathrm{d}\phi \right) + \sum_{j=1}^{K} \log p(V_j)$$

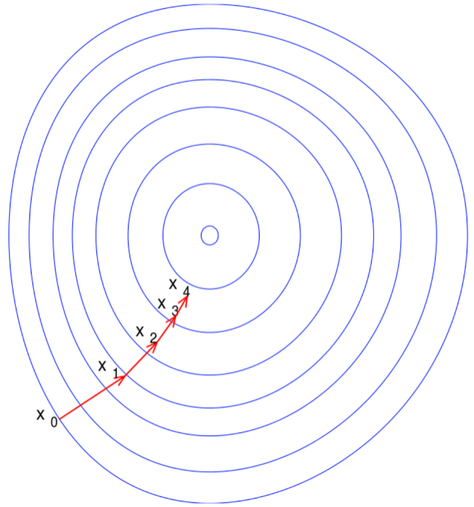
Gradient Descent

- **Gradient descent** is a first-order iterative optimization algorithm for finding a local minimum/maximum of a differentiable function.

$$\mathbf{V}^{(n+1)} = \mathbf{V}^{(n)} \pm \eta \nabla f\left(\mathbf{V}^{(n)}\right)$$

Step size

  - If the step size is too small, the movement in the search space will be small and the search will take a long time. If the step size is too large, the search may bounce around the search space and skip over the optima.



Ref: https://en.wikipedia.org/wiki/Gradient_descent

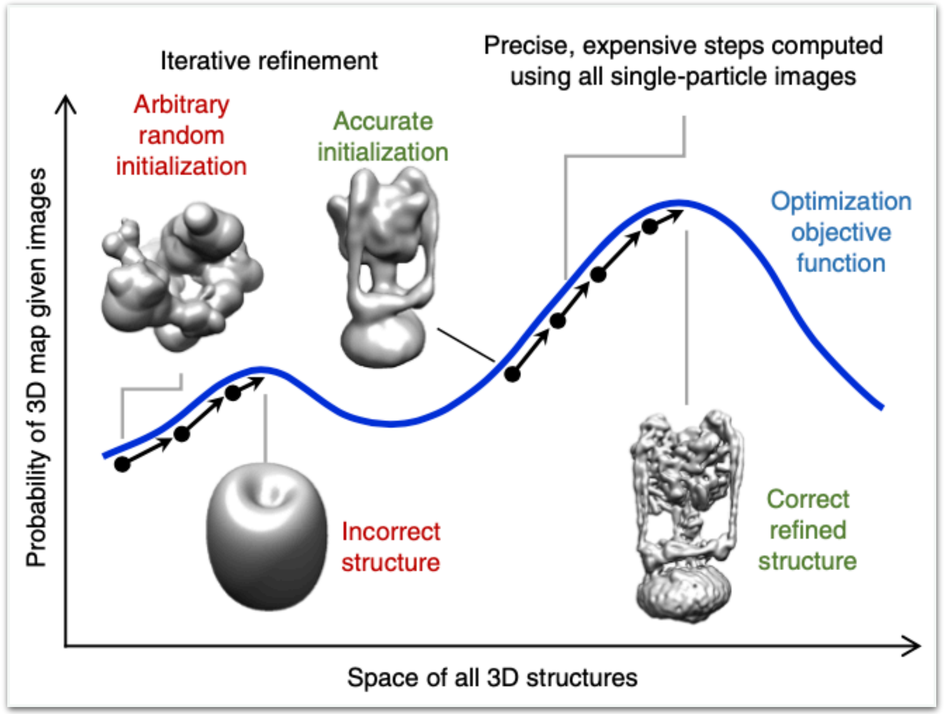
- **Gradient Descent with momentum**: an extension to the gradient descent optimization algorithm, which is designed to accelerate the optimization process.

$$\mathbf{V}^{(n+1)} = \mathbf{V}^{(n)} \pm \left\{ \mu \, \eta \nabla f\left(\mathbf{V}^{(n-1)}\right) + (1-\mu)\, \eta \nabla f\left(\mathbf{V}^{(n)}\right) \right\}$$

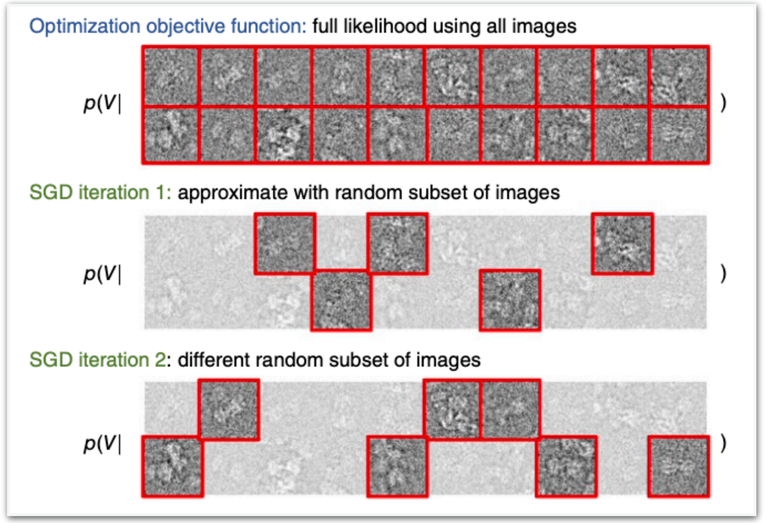
Momentum

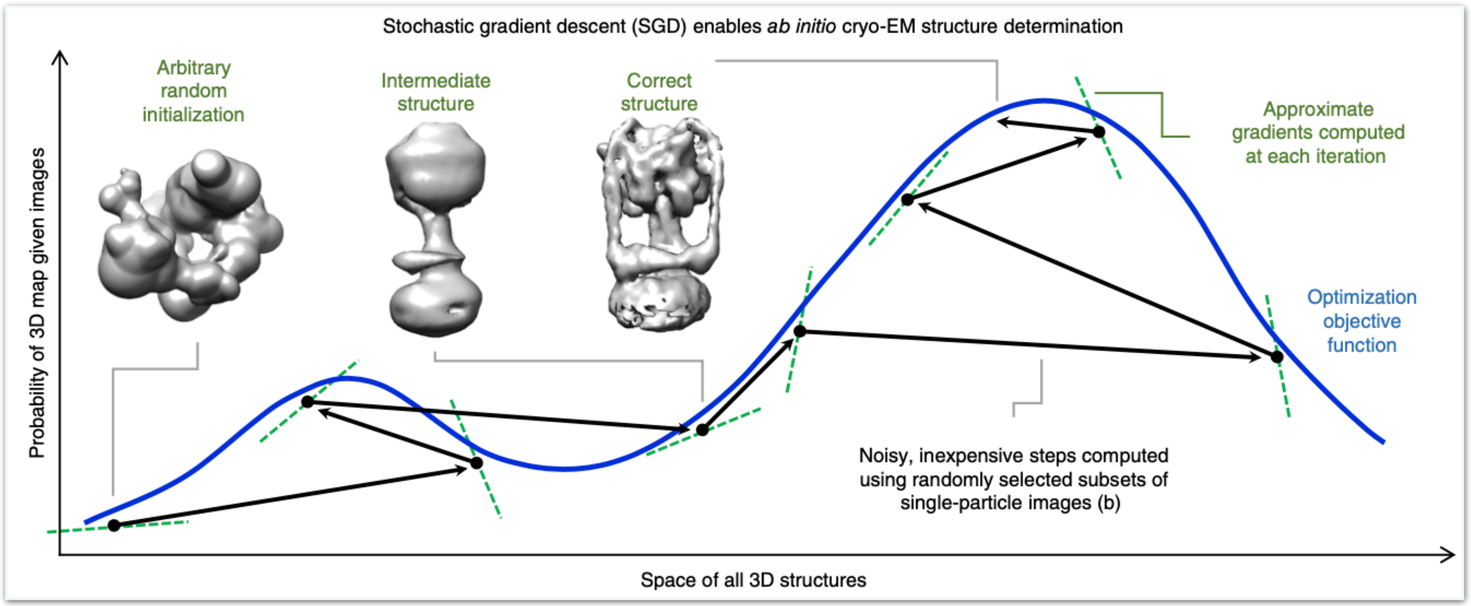
  - The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.

Stochastic Gradient Descent (SGD)



SGD





- The two key parts of SGD:
  - Computation of the approximate gradients.
  - Determination of an appropriate parameter update step based on the gradient.

Gradient and Approximate Gradient

- SGD optimize the objective function $f(\mathbf{V})$ by iteratively updating the parameters $V_1 \cdots V_K$.
- The **gradient of** $f(\mathbf{V})$ w.r.t each structure V_k is computed in order to **take steps**:

Gradient

$$\frac{\partial f(\mathbf{V})}{\partial V_k} = \sum_{i=1}^{N} \frac{1}{U_i} \pi_k \int \frac{\partial}{\partial V_k} p(X_i \mid \phi, V_k) p(\phi) \mathrm{d}\phi + \frac{\partial}{\partial V_k} \log p(V_k)$$

- In SGD, the sum is approximated using **subsampling** (at each iteration, SGD selects a random subset of images from the dataset, called **minibatch**):

Approximate Gradient

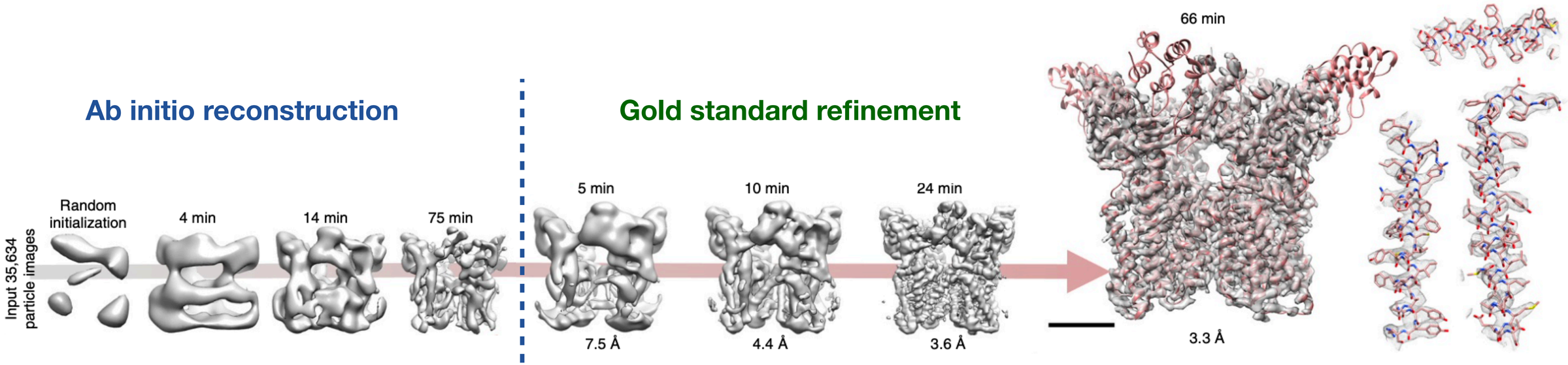
$$\frac{\partial f(\mathbf{V})}{\partial V_k} \approx \frac{N}{M} \sum_{i \in \mathbf{M}} \frac{1}{U_i} \pi_k \int \frac{\partial}{\partial V_k} p(X_i \mid \phi, V_k) p(\phi) \mathrm{d}\phi + \frac{\partial}{\partial V_k} \log p(V_k) \equiv G_k$$

- SGD update rule with **momentum**:

$$\Delta V_k^{(n)} = (\mu) \Delta V_k^{(n-1)} + (1 - \mu)(\eta_k) G_k^{(n)}$$

Momentum — Step size

SGD for 3D Reconstruction



- SGD is able to compute ab initio structures to medium resolution.
- Once SGD has converged, the **Expectation-Maximization algorithm** (a.k.a. iterative refinement) is used to refine the resulting structures to high resolution.
- The computationally expensive part of iterative refinement is the expectation step, in which each 2D image is aligned over 3D orientations and 2D translations to the current estimate of each 3D structure ➟ Solved efficiently using a **branch-and-bound search** technique in cryoSPARC.

Image Alignment Problem Setup

- The probability of observing an image from a particular pose is given in the Fourier domain as following:

$$p(X \mid \phi, V) = p(X \mid r, t, V) = \frac{1}{Z} \exp\left(\sum_l \frac{-1}{2\sigma_l^2} \mid C_l Y_l(r) - S_l(t) X_l \mid^2 \right)$$

- r: 3D orientation
- t: 2D translation
- l: Two-component index of Fourier coefficient (wavevector)
- $S_l(t)$: 2D translation of t pixels
- $Y_l(r)$: Projection of V according to pose r, $Y_l(r) = \Theta_l(r) V$
- C_l: CTF
- σ_l: Gaussian noise (allowing white/colored noise)

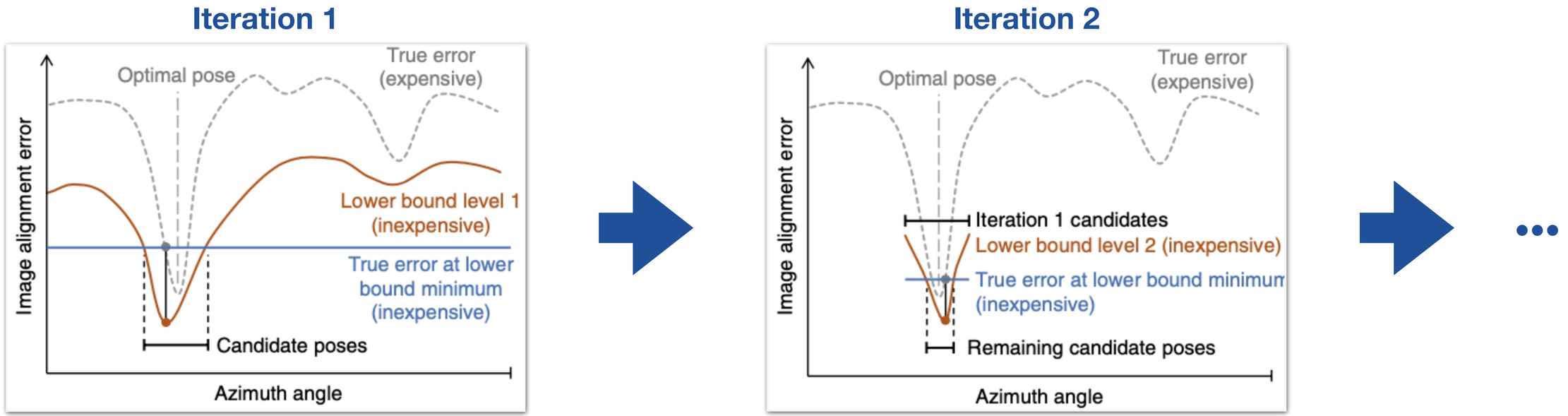
- Taking the **negative log of probability** gives the **image alignment error** (negative log likelihood):

$$E(r, t) = \sum_l \frac{1}{2} \mid C_l Y_l(r) - S_l(t) X_l \mid^2$$

Assume a white noise model $\sigma_l = \sigma = 1$ just for notational clarity.

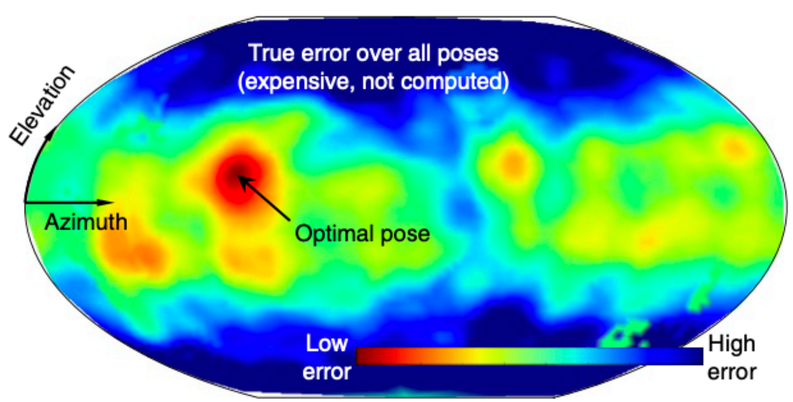
- The aim of image alignment is to **find r and t that minimize this function for the given image X and model V.**

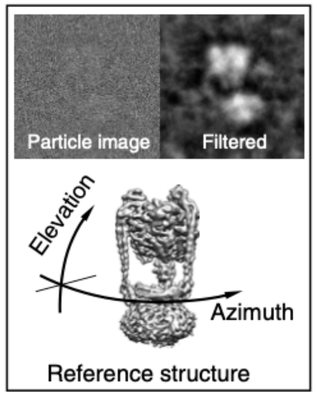
Branch and Bound: Rapid Refinement

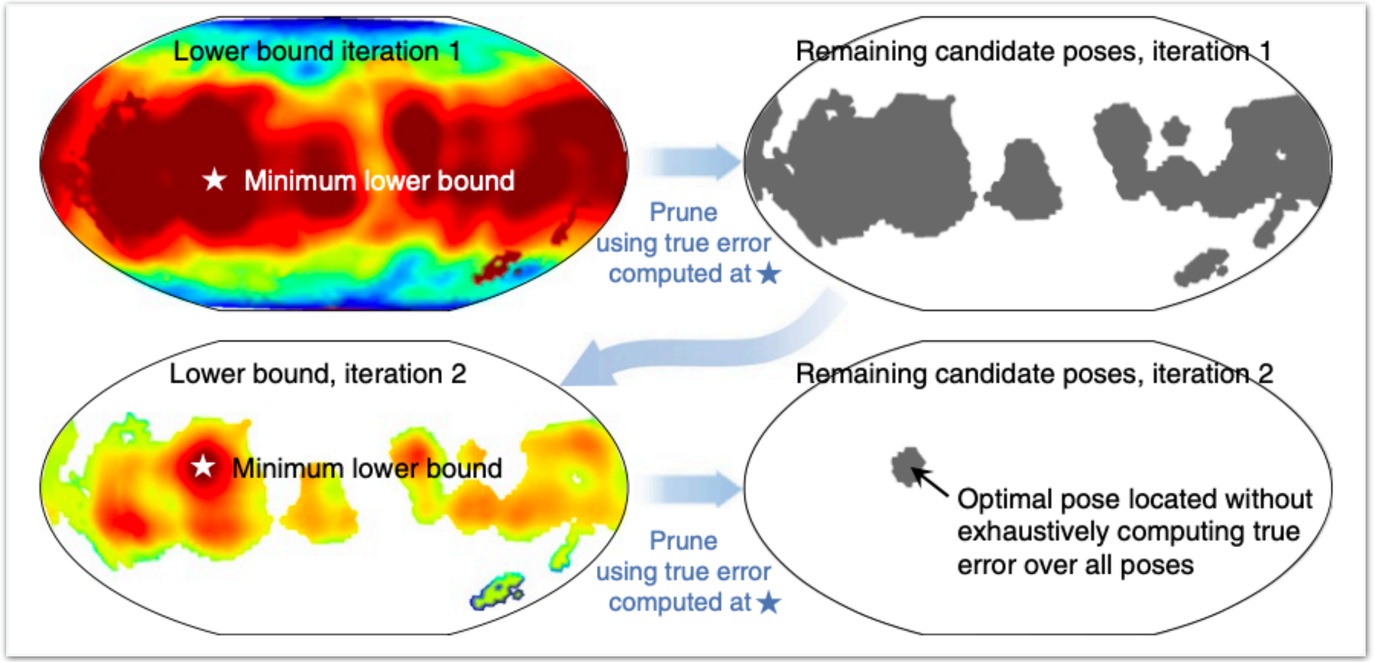


- In cryo-EM map refinement, the optimal pose for a particle image minimizes the error between the observed image and a projection of the 3D map.
- To find this optimal pose using the branch-and-bound approach, an inexpensive lower bound on the error is first computed across the entire space of poses.
- At the pose that minimizes this lower bound, the computationally expensive true error function is evaluated.

Branch-and-Bound Approach to 3D Refinement







$$E(r,t) = \sum_l \frac{1}{2} |C_l Y_l(r) - S_l(t) X_l|^2$$

Image alignment error

$$E(r,t) \geq \sum_{||l||\leq L} \frac{1}{2} |C_l Y_l(r) - S_l(t) X_l|^2 + \sum_{||l||>L} \frac{1}{2} |X_l|^2 - \sum_{||l||>L} \frac{1}{2} C_l^2 |\hat{Y}_l|^2 - 4 \sqrt{\sum_{||l||>L} \frac{1}{2} C_l^2 |\hat{Y}_l|^2}$$

Complete lower bound on $E(r,t)$

- The branch-and-bound approach is a global pose search that requires no prior estimate of an optimal pose.
- For cryo-EM images, the true error function over all poses for an individual particle is never evaluated.

Review of the Quaternion

- A quaternion is an extension of a complex number.
- A quaternion $\mathbf{q}$ is defined as a hypercomplex number composed of **a real part** and **three imaginary parts**:

$$\mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} q_0 \\ \vec{\mathbf{n}}_\mathbf{q} \end{pmatrix}, \quad \text{where } \{\mathbf{i}, \mathbf{j}, \mathbf{k}\} \text{ are three imaginary units,} \quad \vec{\mathbf{n}}_\mathbf{q} = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix}$$

- The conjugate of a quaternion:

$$\mathbf{q}^* = q_0 - q_1\mathbf{i} - q_2\mathbf{j} - q_3\mathbf{k} = \begin{pmatrix} q_0 \\ -q_1 \\ -q_2 \\ -q_3 \end{pmatrix} = \begin{pmatrix} q_0 \\ -\vec{\mathbf{n}}_\mathbf{q} \end{pmatrix}$$

- The norm of a quaternion $\mathbf{q}$ is defined by:

$$|\mathbf{q}| := \sqrt{\mathbf{q} \cdot \mathbf{q}^*} = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = \sqrt{q_0^2 + \vec{\mathbf{n}}_\mathbf{q} \cdot \vec{\mathbf{n}}_\mathbf{q}}$$

Basic Properties of the Quaternion

- The multiplication of two quaternions ($\otimes$: the quaternion product):
  - $1 \otimes 1 = 1$
  - $1 \otimes \mathbf{i} = \mathbf{i} \otimes 1 = \mathbf{i}$
  - $1 \otimes \mathbf{j} = \mathbf{j} \otimes 1 = \mathbf{j}$
  - $1 \otimes \mathbf{k} = \mathbf{k} \otimes 1 = \mathbf{k}$
  - $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$
  - $\mathbf{i} \otimes \mathbf{j} = \mathbf{k}\ , \quad \mathbf{j} \otimes \mathbf{k} = \mathbf{i}, \quad \mathbf{k} \otimes \mathbf{i} = \mathbf{j}$
  - $\mathbf{j} \otimes \mathbf{i} = -\mathbf{k}\ , \mathbf{k} \otimes \mathbf{j} = -\mathbf{i}, \mathbf{i} \otimes \mathbf{k} = -\mathbf{j}$

$\times$	1	i	j	k
1	1	i	j	k
i	i	-1	k	$-j$
j	j	$-k$	-1	i
k	k	j	$-i$	-1

From https://zh.wikipedia.org/wiki/四元數

- The result of quaternion multiplication can also be written in vector form as:

$$\mathbf{q} \otimes \mathbf{q}' = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} \otimes \begin{pmatrix} q'_0 \\ q'_1 \\ q'_2 \\ q'_3 \end{pmatrix} = \begin{pmatrix} q_0 q'_0 - q_1 q'_1 - q_2 q'_2 - q_3 q'_3 \\ q_0 q'_1 + q_1 q'_0 + q_2 q'_3 - q_3 q'_2 \\ q_0 q'_2 - q_1 q'_3 + q_2 q'_0 + q_3 q'_1 \\ q_0 q'_3 + q_1 q'_2 - q_2 q'_1 + q_3 q'_0 \end{pmatrix} = \begin{pmatrix} q_0 \\ \vec{\mathbf{n}}_{\mathbf{q}} \end{pmatrix} \otimes \begin{pmatrix} q'_0 \\ \vec{\mathbf{n}}_{\mathbf{q}'} \end{pmatrix} = \begin{pmatrix} q_0 q'_0 - \vec{\mathbf{n}}_{\mathbf{q}} \cdot \vec{\mathbf{n}}_{\mathbf{q}'} \\ q_0 \vec{\mathbf{n}}_{\mathbf{q}'} + q'_0 \vec{\mathbf{n}}_{\mathbf{q}} + \vec{\mathbf{n}}_{\mathbf{q}} \times \vec{\mathbf{n}}_{\mathbf{q}'} \end{pmatrix}$$

Complex form **Vector form**

- The complex form and vector form of quaternion multiplication are equivalent.

Unit Quaternion Description of 3D Spatial Rotation

- If norm equals to 1 : $|\mathbf{q}| = 1$ —> **Unit quaternion**
- **A unit quaternion can be used to perform 3D rotation**.
- The rotation of a 3D vector $\vec{\mathbf{v}}$ can be achieved by performing unit quaternion multiplication:

$$\mathbf{v}' = \begin{pmatrix} 0 \\ \vec{\mathbf{v}}' \end{pmatrix} = \mathbf{q} \otimes \begin{pmatrix} 0 \\ \vec{\mathbf{v}} \end{pmatrix} \otimes \mathbf{q}^* \equiv R_{\mathbf{q}}(\vec{\mathbf{v}})$$

- Because a pair of unit quaternion is involved in the multiplication, $\mathbf{q}$ and $-\mathbf{q}$ will yield same rotations: $R_{\mathbf{q}} = R_{-\mathbf{q}}$

Example

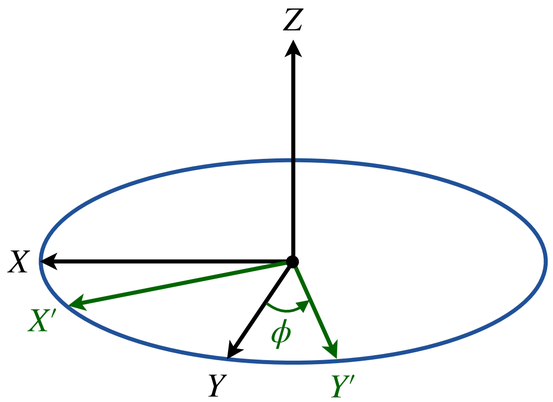
Consider a vector $\vec{\mathbf{v}} = (1,0,0)$, try to rotate it with an unit quaternion $\mathbf{q} = (0,0,0,1)$

$$R_{\mathbf{q}}(\vec{\mathbf{v}}) = \begin{pmatrix} 0 \\ \vec{\mathbf{v}}' \end{pmatrix} = \mathbf{q} \otimes \begin{pmatrix} 0 \\ \vec{\mathbf{v}} \end{pmatrix} \otimes \mathbf{q}^* = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \\ 0 \\ 0 \end{pmatrix}$$

⇒ New rotated vector $\vec{\mathbf{v}}' = (-1,0,0)$

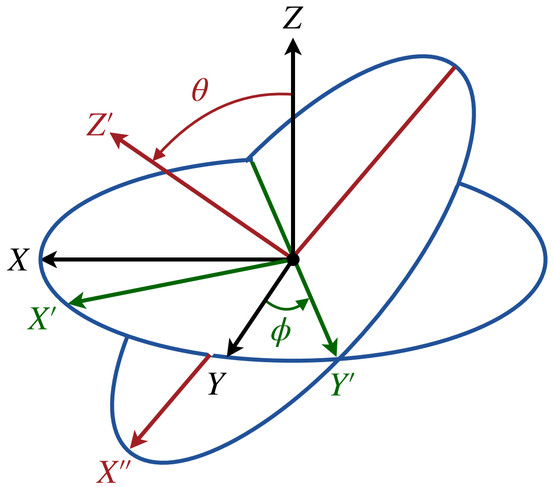
3D Rotation with Euler Angle (ZYZ)

Rotate around Z axis by angle ϕ



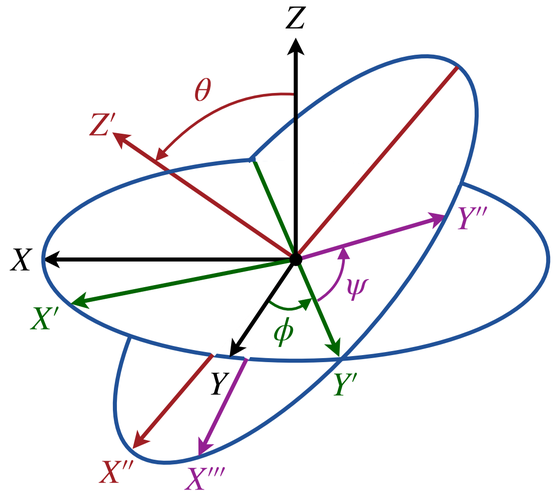
$$R_Z(\phi) = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Rotate around Y' axis by angle θ



$$R_Y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

Rotate around Z' axis by angle ψ

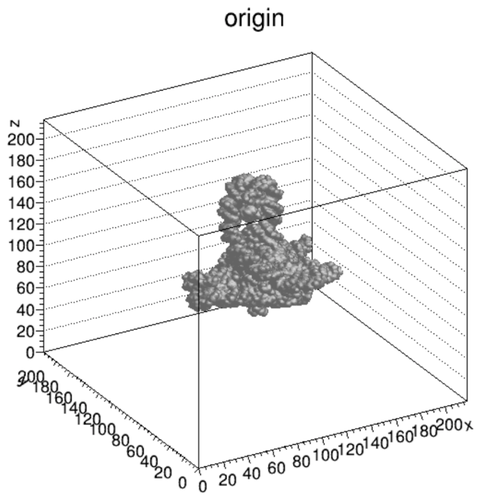


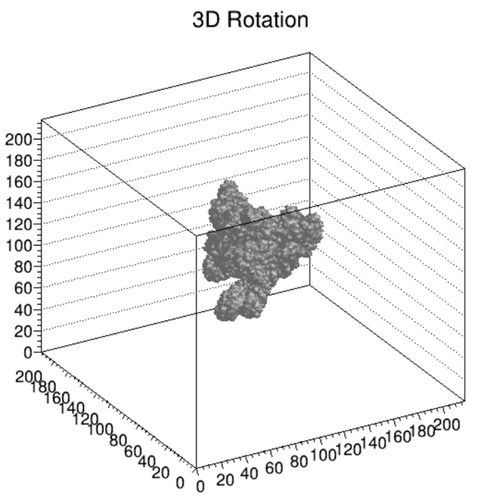
$$R_Z(\psi) = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Example of 3D Rotation

3D model → 3D Rotation → Project 3D model to XY plane → 2D Fourier transform

```
def rotation3D( v=None, phi=0.0, theta=0.0, psi=0.0 ):
    v_rotate = np.zeros_like(v)
    vec_center = np.array( [ v.shape[0]/2, v.shape[1]/2, v.shape[2]/2 ], dtype=floa
    R = Rzyz( phi, theta, psi )
    for x, y, z in it.product( range(v.shape[0]), range(v.shape[1]), range(v.shape
        vec = np.array( [ x, y, z ], dtype=float ) - vec_center
        vec = np.around( np.matmul( R, vec ) + vec_center ).astype('int')
        x_new = vec[0] if vec[0]<v.shape[0] else vec[0]-v.shape[0]
        y_new = vec[1] if vec[1]<v.shape[1] else vec[1]-v.shape[1]
        z_new = vec[2] if vec[2]<v.shape[2] else vec[2]-v.shape[2]
        v_rotate[x_new][y_new][z_new] = v[x][y][z]
    return v_rotate
```



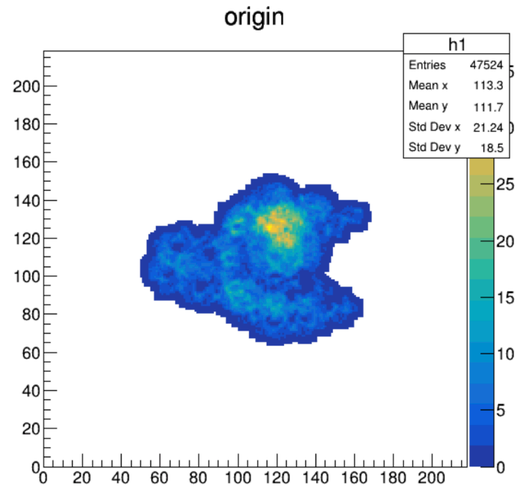


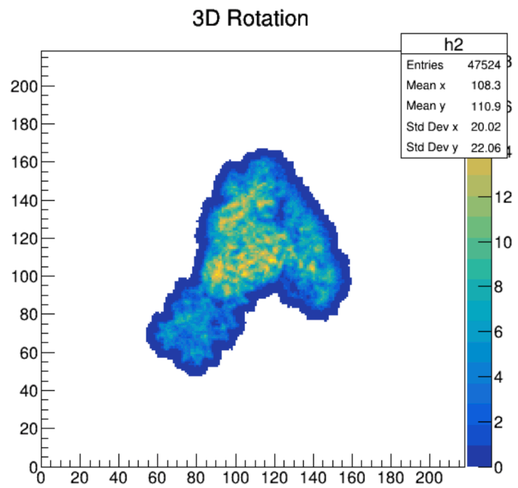
3D rotation test (ϕ=1.4π, θ=0.7π, ψ=0.9π):

```
v_offset_rotate = rotation3D( v_offset, 1.4*np.pi, 0.7*np.pi, 0.9*np.pi )
```

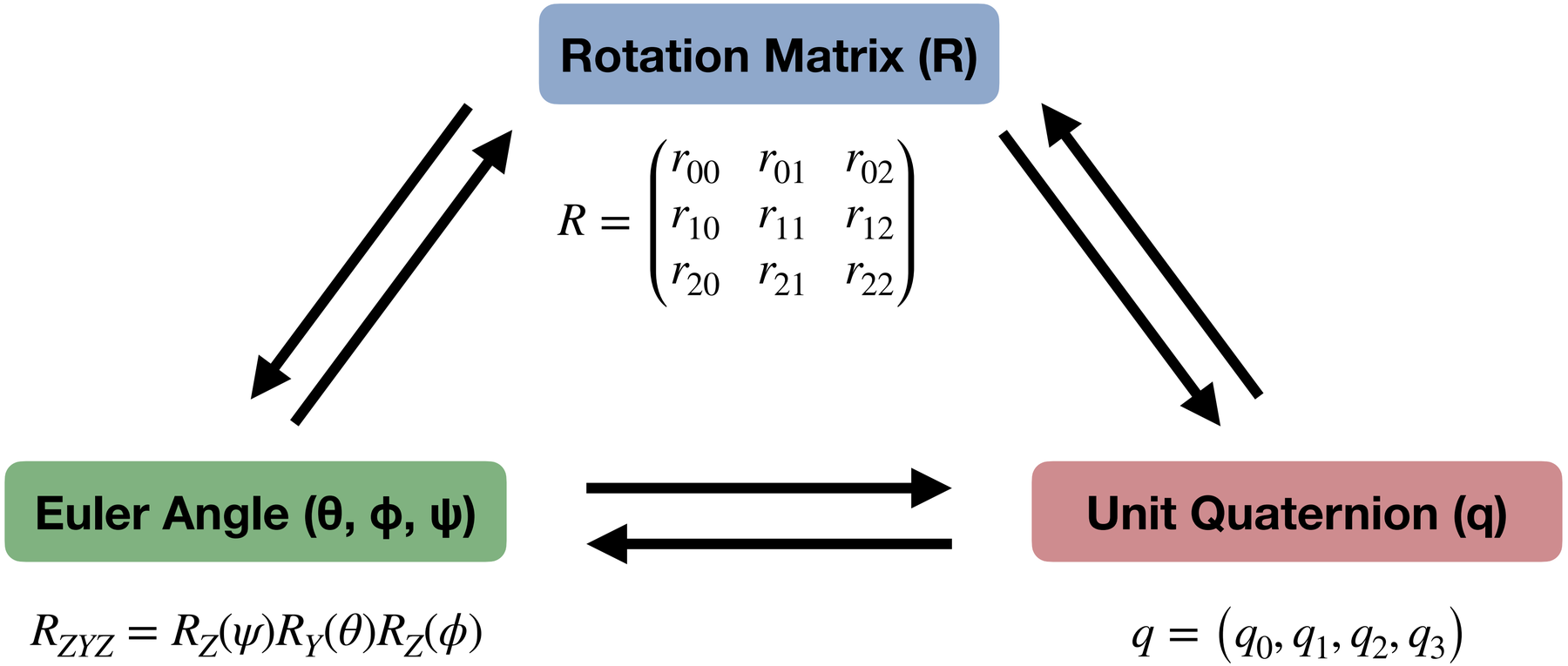
Project to XY plane:

```
v_offset_rotate_proj = np.sum( v_offset_rotate, axis=2 )
v_offset_proj = np.sum( v_offset, axis=2 )
```





Representation of 3D Rotation



Conversion of Rotation Representation

$$R_{ZYZ} = \begin{pmatrix} \cos\theta\cos\phi\cos\psi - \sin\phi\sin\psi & -\cos\theta\sin\phi\cos\psi - \cos\phi\sin\psi & \sin\theta\cos\psi \\ \cos\theta\cos\phi\sin\psi + \sin\phi\cos\psi & -\cos\theta\sin\phi\sin\psi + \cos\phi\cos\psi & \sin\theta\sin\psi \\ -\sin\theta\cos\phi & \sin\theta\sin\phi & \cos\theta \end{pmatrix}$$

$$\phi = 2 \cdot \arctan\left(\frac{-q_0q_2 - q_1q_3 \pm \sqrt{(q_0^2+q_3^2)(q_1^2+q_2^2) + 2q_1q_3}}{q_2q_3 + q_0q_1}\right)$$

$$\theta = 2 \cdot \arcsin\left(\sqrt{q_1^2+q_2^2}\right) = 2 \cdot \arccos\left(\sqrt{q_0^2+q_3^2}\right)$$

$$\psi = 2 \cdot \arctan\left(\frac{-q_0q_2 - q_1q_3 \pm \sqrt{(q_0^2+q_3^2)(q_1^2+q_2^2)}}{q_2q_3 - q_0q_1}\right)$$

$$\phi = \arctan\left(-\frac{r^{21}}{r_{20}}\right)$$

$$\theta = \arccos\left(r_{22}\right)$$

$$\psi = \arctan\left(\frac{r_{12}}{r_{02}}\right)$$

$$q_0 = \cos\frac{\theta}{2}\cos\frac{\phi+\psi}{2}$$

$$q_1 = \sin\frac{\theta}{2}\sin\frac{\phi-\psi}{2}$$

$$q_2 = \sin\frac{\theta}{2}\cos\frac{\phi-\psi}{2}$$

$$q_3 = \cos\frac{\theta}{2}\sin\frac{\phi+\psi}{2}$$

$$q_0 = \frac{1}{2}\sqrt{1 + r_{00} + r_{11} + r_{22}}$$

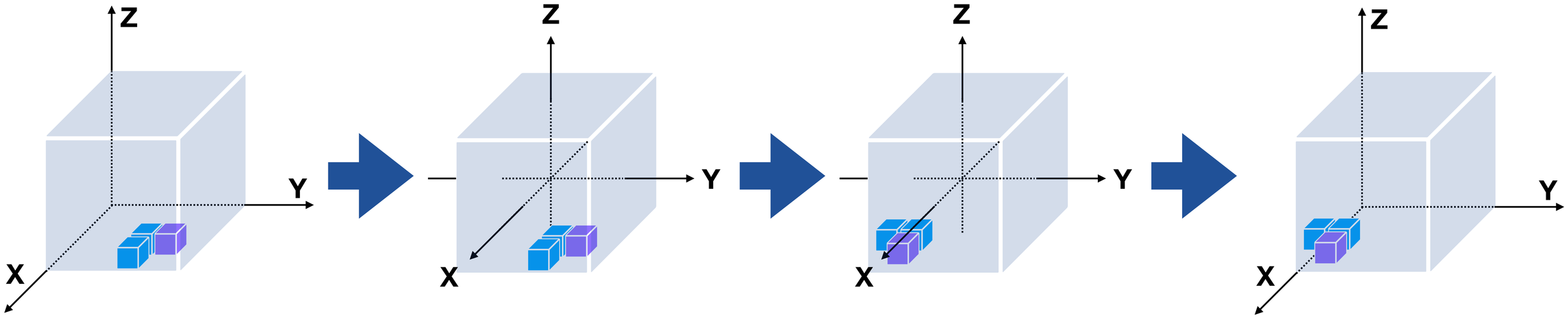
$$q_1 = \frac{1}{2}\frac{r_{21} - r_{12}}{|r_{21} - r_{12}|}\sqrt{1 + r_{00} - r_{11} - r_{22}}$$

$$q_2 = \frac{1}{2}\frac{r_{02} - r_{20}}{|r_{02} - r_{20}|}\sqrt{1 - r_{00} + r_{11} - r_{22}}$$

$$q_3 = \frac{1}{2}\frac{r_{10} - r_{01}}{|r_{10} - r_{01}|}\sqrt{1 - r_{00} - r_{11} + r_{22}}$$

$$R_q = \mathbf{I} + 2q_0\mathbf{A} + 2\mathbf{A}\mathbf{A} = \begin{pmatrix} 1 - 2q_2^2 - 2q_3^2 & -2q_0q_3 - 2q_1q_2 & 2q_0q_2 + 2q_1q_3 \\ 2q_0q_3 + 2q_1q_2 & 1 - 2q_1^2 - 2q_3^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_0q_1 + 2q_2q_3 & 1 - 2q_1^2 - 2q_2^2 \end{pmatrix}$$

Implementation of 3D Rotation



- Fill 3D density info. into 3D array.
- The indexes of 3D array represent each voxels' coordinate.

- Shift the original point from (0,0,0) to the central of volume.

- Rotating each voxel's coordinate along z axis with 270 degree voxel by voxel.

- Shift the volume's original point back to (0,0,0)

```
def rotation3D_Euler( v=None, phi=0.0, theta=0.0, psi=0.0 ):
    v_rotate = np.zeros_like(v)
    vec_center = np.array( [ (v.shape[0]-1)/2, (v.shape[1]-1)/2, (v.shape[2]-1)/2 ] )
    R = np.linalg.inv( Euler2Matrix( phi, theta, psi ) ) # inverse the rotation matrix
    for x, y, z in it.product( range(v.shape[0]), range(v.shape[1]), range(v.shape[2]) ):
        vec = np.array( [ x, y, z ] ) - vec_center
        vec = np.matmul( R, vec ) + vec_center
        vec = np.around( vec ).astype('int')
        x_new = vec[0] if vec[0]<v.shape[0] else vec[0]-v.shape[0]
        y_new = vec[1] if vec[1]<v.shape[1] else vec[1]-v.shape[1]
        z_new = vec[2] if vec[2]<v.shape[2] else vec[2]-v.shape[2]
        v_rotate[x][y][z] = v[x_new][y_new][z_new]
    return v_rotate
```

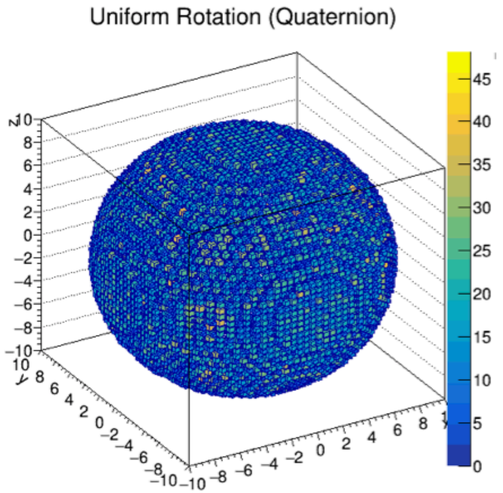
$$\phi = 270^\circ \ , \ \theta = 0^\circ \ , \ \psi = 0^\circ$$

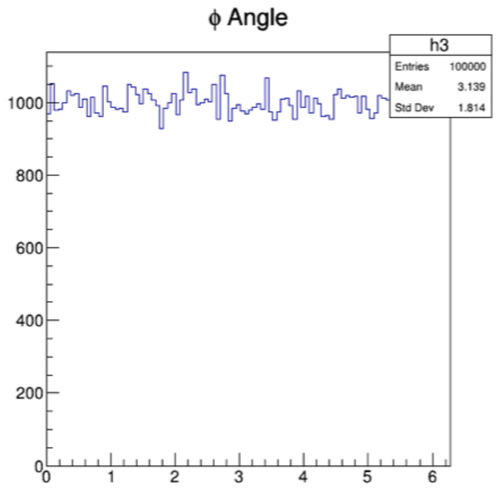
$$R_{ZYZ} = \begin{pmatrix} \cos\theta\cos\phi\cos\psi - \sin\phi\sin\psi & -\cos\theta\sin\phi\cos\psi - \cos\phi\sin\psi & \sin\theta\cos\psi \\ \cos\theta\cos\phi\sin\psi + \sin\phi\cos\psi & -\cos\theta\sin\phi\sin\psi + \cos\phi\cos\psi & \sin\theta\sin\psi \\ -\sin\theta\cos\phi & \sin\theta\sin\phi & \cos\theta \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

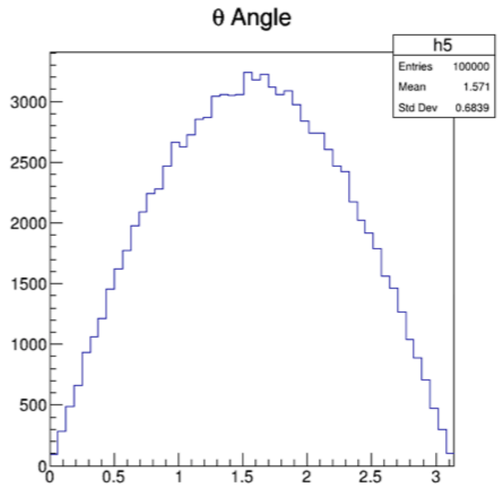
Uniform Random Sampling Test

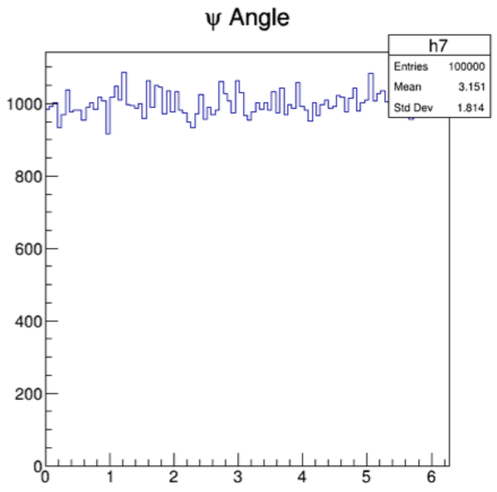
$$s \sim U[0,\ 1]$$
$$\theta_1 \sim U[0,\ 2\pi]$$
$$\theta_2 \sim U[0,\ 2\pi]$$

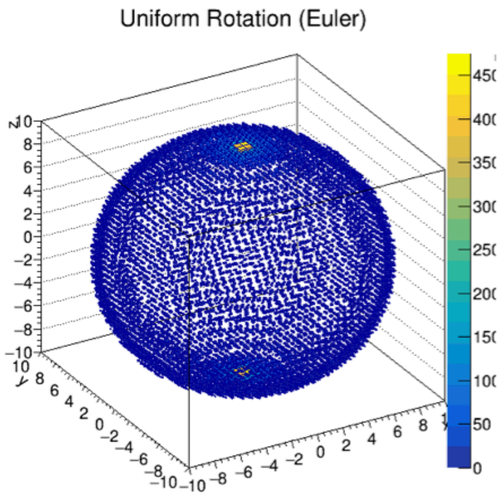
$$\mathbf{q} = \begin{pmatrix} \sqrt{s}\cos\theta_2 \\ \sqrt{1-s}\sin\theta_1 \\ \sqrt{1-s}\cos\theta_1 \\ \sqrt{s}\sin\theta_2 \end{pmatrix} \sim U_{S^3}$$

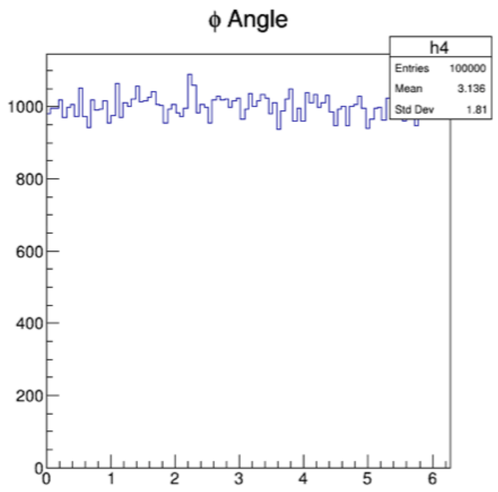


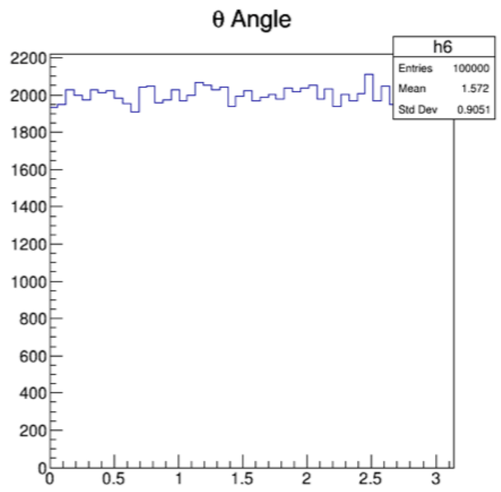


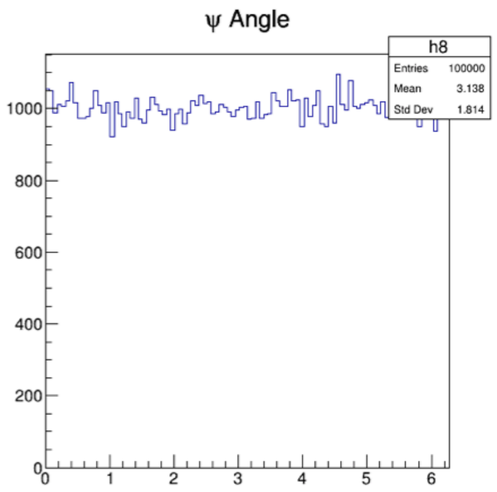






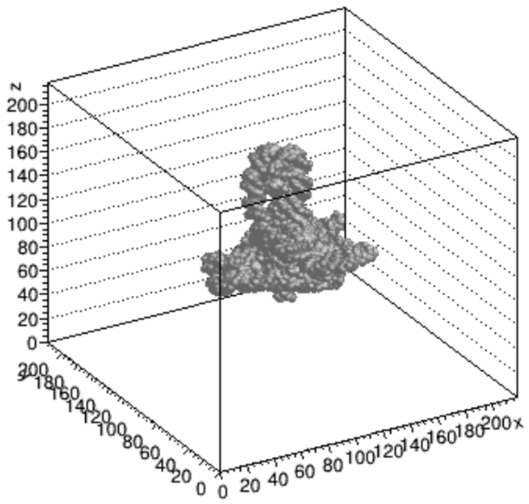




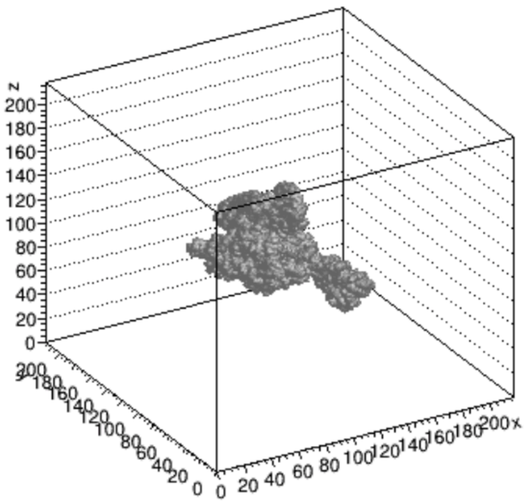


3D Rotation of Protein Structure

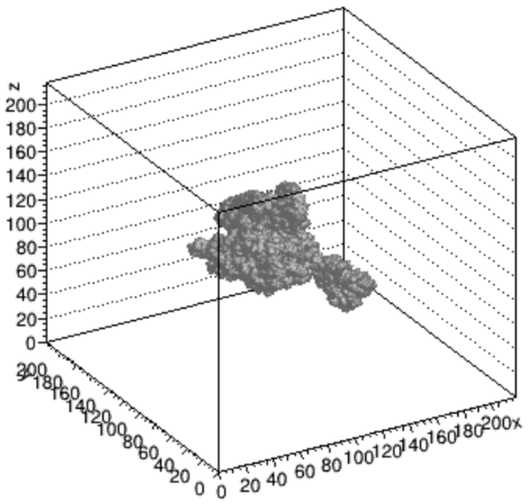
Original



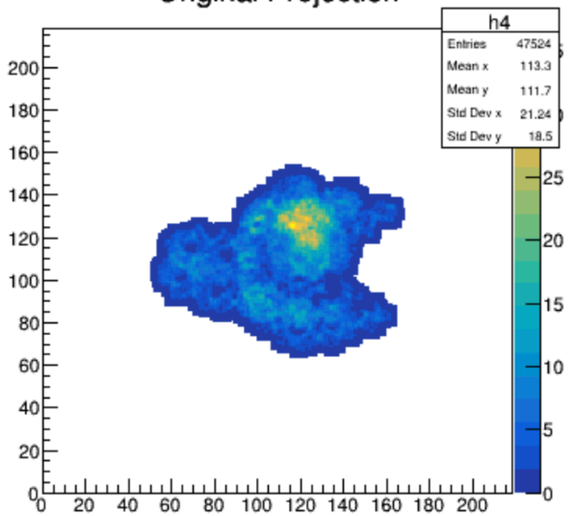
3D Rotation (Euler)



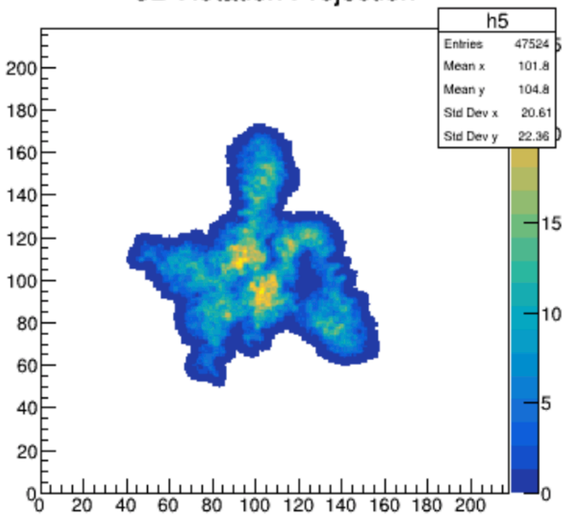
3D Rotation (Quaternion)



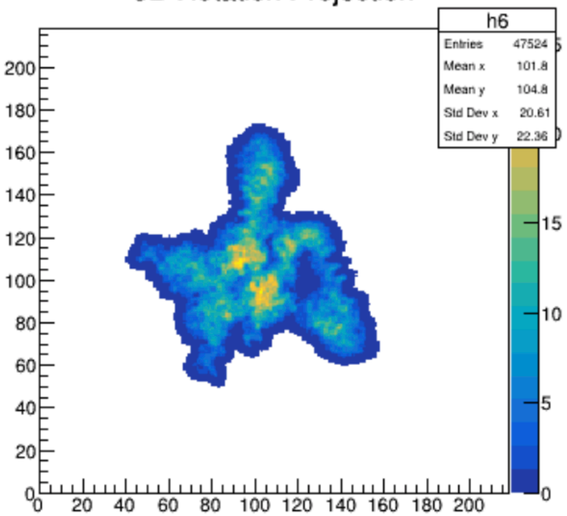
Original Projection



3D Rotation Projection



3D Rotation Projection



Cost of Computing Time

Euler Angle

```
phi   = 1.4*np.pi
theta = 0.3*np.pi
psi   = 1.9*np.pi
start_time = time.time()
vol_offset_rotate1 = rotation3D_Euler( vol_offset, phi, theta, psi )
end_time = time.time()
print( '--- %s seconds ---' % (end_time - start_time) )
```

```
--- 136.03530168533325 seconds ---
```

Quaternion

```
start_time = time.time()
vol_offset_rotate2 = rotation3D_Quat( vol_offset, phi, theta, psi )
end_time = time.time()
print( '--- %s seconds ---' % (end_time - start_time) )
```

```
--- 240.91315579414368 seconds ---
```

Generate Rotation Mapping

```
start_time = time.time()
map_rotate = rotationMap_Euler(218, 1.4*np.pi, 0.3*np.pi, 1.9*np.pi)
end_time = time.time()
print( '--- %s seconds ---' % (end_time - start_time) )
```

```
--- 138.53076076507568 seconds ---
```

```
start_time = time.time()
vol_offset_rotate3 = rotation3D_Euler_Map( vol_offset, map_rotate )
end_time = time.time()
print( '--- %s seconds ---' % (end_time - start_time) )
```

```
--- 18.14928913116455 seconds ---
```

Back Up

Intuition behind a Lower Bound

- The core challenge in employing a branch-and-bound method for cryo-EM is to derive a lower bound that is inexpensive to evaluate but informative about the image alignment error function $E(r, t)$.

- Intuition: *If an image aligns poorly to a structure at low resolution, it will not align well at high resolution*.

- If the low-frequency coefficients already have a given error at a particular pose, the high-frequency coefficients cannot make this error much better or worse.

Derivation of a Lower Bound — I

Image alignment error

$$E(r,t) = \sum_l \frac{1}{2} |\, C_l Y_l(r) - S_l(t) X_l \,|^2$$

Split E into two parts: A and B by radius L

L: a certain radius in Fourier space

$$E(r,t) = \underbrace{\sum_{||l||\le L} \frac{1}{2} |\, C_l Y_l(r) - S_l(t) X_l \,|^2}_{\equiv A(r,t)} + \underbrace{\sum_{||l||>L} \frac{1}{2} |\, C_l Y_l(r) - S_l(t) X_l \,|^2}_{\equiv B(r,t)}$$

Compute A directly (inexpensive when L is small)

$|\, S_l(t) \,| = 1$ and the CTF is real-valued

$$B(r,t) = \underbrace{\sum_{||l||>L} \frac{1}{2} |\, X_l \,|^2}_{\equiv B_1} + \underbrace{\sum_{||l||>L} \frac{1}{2} C_l^2 |\, Y_l(r) \,|^2}_{\equiv B_2} - \underbrace{\sum_{||l||>L} C_l \Re(Y_l(r) * S_l(t) X_l)}_{\equiv B_3}$$

$\Re(z)$: real part of complex value z

$*$: complex conjugation

B_1: The total power of the image at high frequencies.

B_2: The total power of a slice of the model from pose r at high frequencies.

B_3: The correlation between the shifted image X and the slice of the 3D model in the Fourier domain.

Derivation of a Lower Bound — II

$$B(r,t) = \underbrace{\sum_{||l||>L} \frac{1}{2}|X_l|^2}_{\equiv B_1} + \underbrace{\sum_{||l||>L} \frac{1}{2}C_l^2|Y_l(r)|^2}_{\equiv B_2} - \underbrace{\sum_{||l||>L} C_l \Re(Y_l(r) * S_l(t) X_l)}_{\equiv B_3}$$

B_3 *An upper bound on B_3 contributes to a lower bound on $B(r,t)$.*

The cryo-EM image formation model: $X_l = C_l \tilde{X}_l + \epsilon_l$, $\epsilon_l \sim \mathcal{CN}\left(0, \frac{1}{2}\right)$

(True signal, Gaussian noise)

$$B_3 = \sum_{||l||>L} C_l^2 \Re(Y_l(r) * S_l(t)\tilde{X}_l) + \underbrace{\sum_{||l||>L} C_l \Re(Y_l(r) * S_l(t)\epsilon_l)}_{\equiv H}$$

$$\leq \sum_{||l||>L} C_l^2 |Y_l(r)||\tilde{X}_l| + H$$

H is normally distributed with variance $\sigma_H^2 = \sum_{||l||>L} \frac{1}{2}C_l^2|Y_l(r)|^2$

H is a random variable which captures the expectation of the effect of noise on E:

$$\begin{aligned}
H &= \sum_{||l||>L} C_l \Re(Y_l(r) * S_l(t)\epsilon_l) \\
&= \sum_{||l||>L} C_l \Re(Y_l(r) * \epsilon_l) \\
&= \sum_{||l||>L} C_l \Re\left(Y_l(r) * \mathcal{CN}\left(0, \frac{1}{2}\right)\right) \\
&= \sum_{||l||>L} C_l \Re\left(\mathcal{CN}\left(0, \frac{1}{2}|Y_l(r)|^2\right)\right) \\
&= \sum_{||l||>L} \mathcal{N}\left(0, \frac{1}{2}C_l^2|Y_l(r)|^2\right) \\
&= \mathcal{N}\left(0, \sum_{||l||>L} \frac{1}{2}C_l^2|Y_l(r)|^2\right)
\end{aligned}$$

Derivation of a Lower Bound — III

$$B(r,t) = \underbrace{\sum_{||l||>L} \frac{1}{2}|X_l|^2}_{\equiv B_1} + \underbrace{\sum_{||l||>L} \frac{1}{2}C_l^2|Y_l(r)|^2}_{\equiv B_2} - \underbrace{\sum_{||l||>L} C_l \Re(Y_l(r) * S_l(t)X_l)}_{\equiv B_3}$$

$B_2 - B_3$

$$B_2 - B_3 \geq \sum_{||l||>L} \frac{1}{2}C_l^2|Y_l(r)|^2 - \sum_{||l||>L} C_l^2|Y_l(r)||\tilde{X}_l| - H$$

$$= \underbrace{\sum_{||l||>L} \frac{1}{2}\left(C_l^2|Y_l(r)|^2 - 2C_l^2|Y_l(r)||\tilde{X}_l|\right)}_{\equiv Q} - H$$

The maximum of lower bound of Q is attained at $r = \hat{r}$, $\hat{Y}_l \equiv Y_l(\hat{r})$

$\hat{Y}$ is **the slice of model V that has the maximum CTF-modulated total power**.

$$Q \geq -\sum_{||l||>L} \frac{1}{2}C_l^2|\hat{Y}_l|^2 \equiv \hat{Q}$$

Once $\hat{Y}$ is identified, the bound $\hat{Q}$ on Q is fixed.

Each term of Q is a positive-definite quadratic function of $Y_l(r)$.
Therefore Q can be bounded from below:

$$Q \geq \min_{Y_l(r)} Q$$

$$= \min_{Y_l(r)} \sum_{||l||>L} \frac{1}{2}\left(C_l^2|Y_l(r)|^2 - 2C_l^2|Y_l(r)||\tilde{X}_l|\right)$$

$$= \sum_{||l||>L} -\frac{1}{2}C_l^2|\tilde{X}_l|^2$$

$$= \sum_{||l||>L} -\frac{1}{2}C_l^2|Y_l(r^*)|^2$$

$\tilde{X}_l = Y_l(r^*)$ ↓ Unknown true pose

$$\geq \min_{r} \sum_{||l||>L} -\frac{1}{2}C_l^2|Y_l(r)|^2$$

$$= -\max_{r} \sum_{||l||>L} \frac{1}{2}C_l^2|Y_l(r)|^2$$

Derivation of a Lower Bound — IV

Lower bound on $B(r,t)$

$$B(r,t) \geq \sum_{||l||>L} \frac{1}{2}|X_l|^2 - \sum_{||l||>L} \frac{1}{2}C_l^2|\hat{Y}_l|^2 - H$$

Lower bound on $E(r,t)$

$$E(r,t) \geq \sum_{||l||\leq L} \frac{1}{2}|C_l Y_l(r) - S_l(t)X_l|^2 + \sum_{||l||>L} \frac{1}{2}|X_l|^2 - \sum_{||l||>L} \frac{1}{2}C_l^2|\hat{Y}_l|^2 - H$$

Due to the presence of H, the above expression is a **probabilistic** bound on $E(r,t)$, giving the probability of $E(r,t)$ being greater than the value of the expression.

In practice, consider a probability of 0.999936 of H ($4\sigma_H$)

$$H \leq 4\sigma_H \leq 4\sqrt{\max_r \sum_{||l||>L} \frac{1}{2}C_l^2|Y_l(r)|^2}$$

Complete lower bound on $E(r,t)$

$$E(r,t) \geq \sum_{||l||\leq L} \frac{1}{2}|C_l Y_l(r) - S_l(t)X_l|^2 + \sum_{||l||>L} \frac{1}{2}|X_l|^2 - \sum_{||l||>L} \frac{1}{2}C_l^2|\hat{Y}_l|^2 - 4\sqrt{\sum_{||l||>L} \frac{1}{2}C_l^2|\hat{Y}_l|^2} \equiv \beta_L(r,t)$$

The bound is inexpensive to compute for a particular r, t (since only $A(r,t)$ depends on these).