

```
def GetReferenceGaussianParams (map) :  
    # Input  
    # map : cryoEM or X-ray map  
  
    # Output  
    # A : reference gaussian height  
    # B : reference gaussian offset  
  
    # determine max and min value in map M  
    mapValues = map.allValues()  
    maxM = max(mapValues)  
    minM = min(mapValues)  
  
    # determine value 10 standard deviations above mean (capped at maxM)  
    highV = min (average(M) + standard_dev(M)*10, maxM)  
  
    # determine value 1 standard deviations below mean (capped at minM)  
    lowV = max (average(M) - standard_dev(M)*1, minM)  
  
    # determine reference gaussian height, A, and offset, B  
    A = highV - lowV  
    B = lowB  
  
    return A, B  
  
def GetRadialPoints ( atom, mol, R, N ) :  
    # get ~N points evenly distributed around a sphere with radius R  
    # all points should be closest to the atom and not another atom in mol  
    # hence this might take a few tries since some points distributed on a  
    # sphere will have to be thrown away  
  
    # Input  
    # atom : atom  
    # mol : entire molecule from which atom comes  
    # R : radius of sphere on which points should be placed  
    # B : reference gaussian offset  
    # sigma : reference gaussian width  
    # numPts : number of points to use at each radial distance  
  
    # Output  
    # atomQ : Q-score for the atom  
  
    rPoints = None  
  
    for tryN = 0 to 99 # give up after 100 tries and keep possibly fewer than N points  
        rPoints = []  
  
        # this function returns (N + tryN) points evenly distributed on a  
        # sphere of radius R centered at the atom's position  
        spherePoints = SpherePoints ( atom.position, R, N + tryN )  
  
        for P in spherePoints:  
            if P is closer to another atom in mol than to atom  
                # ignore this point  
                pass  
            else  
                # use this point  
                rPoints.append ( rPoint )  
  
            if len(rPoints) >= N :  
                break  
  
    return rPoints
```

```
def CalculateQscoreForAtom (atom, map, mol, A, B, sigma, numPts) :  
    # Input  
    # atom : atom  
    # map : map (cryoEM or X-ray)  
    # mol : entire molecule from which atom comes  
    # A : reference gaussian height  
    # B : reference gaussian offset  
    # sigma : reference gaussian width  
    # numPts : number of points to use at each radial distance  
  
    # Output  
    # atomQ : Q-score for the atom  
  
    referenceGaussianValues = []  
    mapValues = []  
  
    # map value at point P, interpolated from nearby grid points  
    MapValueAtR0 = map.ValueAtPoint(P)  
  
    # replicate it numPts times so that it has the ~same 'weight' as the  
    # ~numPts that will be added at each radial distance below  
    # in the below, * is the replication operator as in python  
    # e.g. [1] * 4 -> [1,1,1,1]  
    mapValues.append ( [MapValueAtR0] * numPts )  
  
    # value of reference gaussian at radial distance of 0  
    RefGvalueAtR0 = A + B  
    referenceGaussianValues.append ( [RefGvalueAtR0] * numPts )  
  
    for R = 0.1 to 2.0 in increments of 0.1 :  
        rPoints = GetRadialPoints ( atom, mol, R, numPts )  
        mapValuesAtPoints = map.ValuesAtPoints ( rPoints )  
        mapValues.append ( mapValuesAtPoints )  
  
        RefGvalueAtR = A * e-(1/2)*(R/sigma)^2 + B  
        RefGvalues = [RefGvalueAtR] * len(rPoints)  
        referenceGaussianValues.append ( RefGvalues )  
  
    atomQ = correlationAboutMean ( mapValues, referenceGaussianValues )  
    return atomQ  
  
def CalcQScores ( map, model ) :  
    # Input  
    # map : map (cryoEM or X-ray)  
    # mol : at atomic model  
  
    # Output  
    # each atom has a Q-score calculated (except Hydrogen atoms)  
    # return average Q-score for map and model  
  
    A, B = GetReferenceGaussianParams ( map )  
  
    sum = 0  
    N = 0  
  
    for atom in model.atoms :  
        # (ignore Hydrogen atoms)  
        if atom is not Hydrogen atom :  
            atom.Q = CalculateQscoreForAtom (atom, map, model, A, B, 0.6, 8)  
            sum += atom.Q  
            N += 1  
  
    return sum / N
```