```
def logAUC(points):
    # constants
    LOGAUC_MAX = 1.0   ## this should not change
    LOGAUC_MIN = 0.001 ## this you may want to change if you database is
large and you have strong early enrichment.
    RANDOM_LOGAUC = (LOGAUC_MAX-
LOGAUC_MIN)/np.log(10)/np.log10(LOGAUC_MAX/LOGAUC_MIN)


    """Compute semilog x AUC minus the perfectly random semilog AUC."""
    # assumes we have previously interpolated to get y-value at x = 0.1%
    # generate new points array clamped between 0.1% and 100%

    npoints = []
    for x in points:
        if (x[0] >= LOGAUC_MIN*100) and (x[0] <= LOGAUC_MAX*100):
            npoints.append( [x[0]/100 , x[1]/100] )

    area = 0.0
    for point2, point1 in zip(npoints[1:], npoints[:-1]):
        if point2[0] - point1[0] < 0.000001:
            continue

        # segment area computed as integral of log transformed equation
        dx = point2[0]-point1[0]
        dy = point2[1]-point1[1]
        intercept = point2[1] - (dy)/(dx) * point2[0]
        area += dy/np.log(10) + intercept*(np.log10(point2[0])-
np.log10(point1[0]))

    print("logAUC", area, area/np.log10(LOGAUC_MAX/LOGAUC_MIN) -
RANDOM_LOGAUC)
    return area/np.log10(LOGAUC_MAX/LOGAUC_MIN) - RANDOM_LOGAUC

def roc(scores, lig_list, decoy_list, nbins=10000):
    """Calculate ROC curve given ranked ids and ligand ids."""

    num_data = len(scores)
    binsize = int(num_data/nbins) + 1
    num_lig = len(lig_list)
    num_dec = len(decoy_list)

    found_ligand = 0
    results = []
    for i in range(num_data):
        if i % binsize == 0:
            results.append([i-found_ligand, found_ligand])
        if scores[i][2].split('.')[0] in lig_list:
            found_ligand += 1
    results.append([num_data - found_ligand, found_ligand])
    results.append([num_dec, num_lig])

    points = []
    for x in results:
        fpr = x[0]*100.0/num_dec
        tpr = x[1]*100.0/num_lig
        points.append([fpr, tpr])
```

```
    return points

def interpolate_curve(points):
    """Interpolate curve where needed to get better graph visualization."""
    i = 0
    while i < len(points) and points[i][0] < 0.1:
        i += 1
    slope = (points[i][1] - points[i-1][1])/(points[i][0] - points[i-1][0])
    intercept = points[i][1] - slope * points[i][0]
    point_one =  [0.100001, (slope * 0.100001 + intercept)]
    npoints = [x for x in points]
    npoints.insert(i, point_one)
    return npoints
```