Nearest neighbor searching

Grady Wright

Contents

In this tutorial we look at how to efficiently solve the following problem:

Let $X = \{\mathbf{x}_j\}_{j=1}^N$ and $Y = \{\mathbf{y}_i\}_{i=1}^M$ be sets of points on the surface of the sphere. For each point $\mathbf{y}_i$, determine its k nearest neighbors in X.

Here we interpret nearest neighbors to mean the closest points as meausured by Euclidean distance.

A Naive solution to this problem is to compute the distance for each point in Y to all the points in X, then sort these distances to deterimine the k nearest neighbors. This computation requires $\mathcal{O}(MN)$ operations, which is too expensive for large N and M.

A much more computationally efficient solution is to first build a KD-tree of the nodes in X and then use it to search for the k nearest neigbors in Y. This operation requires $\mathcal{O}(N \log N)$ operations to build the KD-tree and $\mathcal{O}(M \log N)$ operations to search for the nearest neighbors.

There are functions for building and searching a KD-tree included in the MATLAB Statistics Toolbox. Alternatively, if one does not have this package, then several KD-tree codes are available to download for free from MatlabCentral from the Mathworks website. For example, the KD-tree package written by Andrea Tagliasacchi is particularly nice.

The `findKNearestNeighbors` function in the **rbfsphere** provides an easy way to do nearest neighbor searching with a KD-tree using either functions from the Statistics Toolbox or from the Tagliasacchi package. The default is to use the former if available.

```
LW = 'linewidth'; lw = 1; FS = 'FontSize'; fs = 12;
FC = 'FaceColor'; fc = [0.93 0.93 0.93]; EC = 'EdgeColor'; ec = 'none';
vw = [77 7];
```

Using the `findKNearestNeighbors` function

Here we give an example for finding the k nearest neigbors in a node set X on the sphere to a given point. We use the minimum energy nodes as X and the point $(1, 0, 0)$ as Y

```
x = getMinEnergyNodes(3136);
y = [1 0 0];
```

The 11 nearest neighbors in X to Y can be found as

```
idx = findKNearestNeighbors(x,y,21);
```

`idx` contains indices for the 21 nearest neighbors of `x` to `y`. You can list these as

```
x(idx,:)
```

```
ans =

   0.999447324384150   0.032081875404895   0.008706265093021
   0.999315724871511  -0.030839450799104  -0.020420830025636
   0.998540439767608  -0.032723782506501   0.042966780274756
   0.997929576268947   0.033905468056771  -0.054653271114642
   0.996946104040931   0.030273949313509   0.071985787698497
   0.996056370781362  -0.028870836603052  -0.083893867592942
   0.995321955543694  -0.095564505509654   0.014199651382042
   0.994997414142708   0.096752532727890  -0.024881584757867
   0.994738637697857   0.094989629027157   0.038366822233465
   0.994403077597231  -0.093470681971343  -0.049251912419395
   0.993745545819535  -0.034115901501386   0.106329184274980
   0.992374520898971   0.035306076639353  -0.118094416560958
   0.992248359015711  -0.097040871186256   0.077629011007757
   0.991268395198822   0.098019031097226  -0.088199989924677
   0.990475424351139   0.093126390301472   0.101419471433497
   0.990410915528553   0.028803579873238   0.135116883432014
   0.989439594331779  -0.091146710478655  -0.112701225980396
   0.988708791748287  -0.027108895837548  -0.147377178986814
   0.987338855157845  -0.157967402132447  -0.014432080900626
   0.987204238927591   0.159358136478389   0.005724943801601
   0.985957134322539  -0.159663834734405   0.048948842237509
```

The indices in `idx` are sorted according to the distance each point is in X to Y. These distances can be obtained by calling `findKNearestNeighbors` with a second output argument:

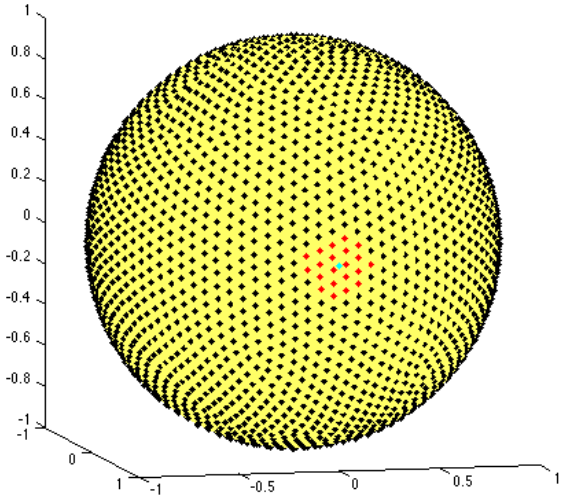
```
[idx,dist] = findKNearestNeighbors(x,y,21);
dist'
```

```
ans =

   0.033246822881315
   0.036993921892356
   0.054028885466794
   0.064349416952334
   0.078152363484023
   0.088810238358403
   0.096726877922382
   0.100025855230454
   0.102580332443825
   0.105800967885641
   0.111843231180659
   0.123494769938071
   0.124512175985240
   0.132148437759801
   0.138018662860219
   0.138485266158155
   0.145330008382444
   0.150274470564451
   0.159129788802444
   0.159973504508773
   0.167587980938142
```

To get a visual picture, we can plot all the nodes in X and the node in Y, then highlight the nearest neighbors.

```
plotSphNodes(x); hold on;
plot3(y(:,1),y(:,2),y(:,3),'c.',MS,ms);       % Plot the node y in cyan
plot3(x(idx,1),x(idx,2),x(idx,3),'r.',MS,ms);
view(vw);
hold off;
```



More than one node can be included in Y when using the `findKNearestNeighbors` function. In this case, each row of idx will correspond to the indices in X closest to the point in each row of Y. Here's an example:

```
y = [[1 0 0];[0 1 0];[0 0 1]];
idx = findKNearestNeighbors(x,y,5);
disp('5 nearest nodes to [1 0 0]');
x(idx(1,:),:)
disp('5 nearest nodes to [0 1 0]');
x(idx(2,:),:)
disp('5 nearest nodes to [0 0 1]');
x(idx(3,:),:)
```

```
5 nearest nodes to [1 0 0]

ans =

   0.999447324384150   0.032081875404895   0.008706265093021
   0.999315724871511  -0.030839450799104  -0.020420830025636
   0.998540439767608  -0.032723782506501   0.042966780274756
   0.997929576268947   0.033905468056771  -0.054653271114642
   0.996946104040931   0.030273949313509   0.071985787698497

5 nearest nodes to [0 1 0]

ans =
```

```
   0.000904571662442   0.999930451325837   0.011759007671380
  -0.000982414173785   0.998697810399154  -0.051007042320908
   0.064404384008626   0.997756709614094  -0.018265369980531
  -0.064313697914007   0.997719289707643  -0.020493101422251
   0.002722183033220   0.997216057084660   0.074516610309764

5 nearest nodes to [0 0 1]

ans =

  -0.026493330339294  -0.021530002648904   0.999417111337139
   0.036215778337423  -0.004680044962085   0.999333034868040
  -0.011913001951517   0.046667133463354   0.998839456088323
   0.018850403854112  -0.072332443186087   0.997202426760619
  -0.072273005791835   0.031475050919676   0.996888125018759
```

To find the 20 nearest neighbors between all points in X we simply use:

```
idx = findKNearestNeighbors(x,x,20);
```

This size of idx in this case is

```
size(idx)

ans =

        3136          20
```

Computing effective resolution of a node set

The effective resolution of a node set on the sphere is typically defined as the maximum distance between all nearest neighbors of the point set. In this case the distance is measured as great circle distance. We can compute this quanity quite naturally using a KD-tree.

```
[idx,dist] = findKNearestNeighbors(x,x,2);
maxdist = max(dist(:,2));
sphdist = acos(1-maxdist^2/2);
fprintf('Effective resolution of N=3136 ME nodes on unit sphere is %f\n',sphdist);

Effective resolution of N=3136 ME nodes on unit sphere is 0.070482
```

To see what this means for Earth we simply need to scale this quantity by the mean radius of Earth: 6371km

```
R = 6371;
sphdist = R*acos(1-mx^2/2);
fprintf('Effective resolution of N=3136 ME nodes on Earth is %f km\n',sphdist);
```

```
Effective resolution of N=3136 ME nodes on Earth is 235.701718 km
```

This can be interpreted as meaning that there is one node every 235.7 km. Here is this quanity for a larger set of MD nodes.

```
N = 192^2;
x = getMaxDetNodes(N);
[idx,dist] = findKNearestNeighbors(x,x,2);
maxdist = max(dist(:,2));
sphdist = R*acos(1-maxdist^2/2);
fprintf('Effective resolution of N=%d MD nodes on Earth is %f km\n',N,sphdist);
```

```
Effective resolution of N=36864 MD nodes on Earth is 134.095363 km
```

Here is this quanity for a even larger set of Icosahedral nodes.

```
x = getIcosNodes(8,0);
N = size(x,1);
[idx,dist] = findKNearestNeighbors(x,x,2);
maxdist = max(dist(:,2));
sphdist = R*acos(1-maxdist^2/2);
fprintf('Effective resolution of N=%d icosahedral nodes on Earth is %f km\n',N,sphdist);
```

```
Effective resolution of N=655362 icosahedral nodes on Earth is 32.927907 km
```

How many quasi-uniformly nodes are need for a resolution of 1km?