Nodes/grids/meshes on the sphere

Grady Wright

Contents

A key difficulty with developing numerical methods for problems in spherical geometries is that it is impossible to uniformly distribute more than 20 points on a sphere, in contrast to trivially placing any number of points uniformly along the periphery of a circle. The essence of the problem is one that has faced cartographers for at least a millennium - the surface of a sphere cannot be mapped to a rectangle without distortions and a singularity in at least one location. In standard spherical coordinates, there are two such singularites at the north and south poles, which is referred to as the "Pole Problem".

Because of this inability to produce a uniform distribution of points on sphere larger than 20, much research has been devoted (and contiues today) to generating point sets that are quasi-uniformly distributed. These are based on mapped rectangular grids, a subdivided polyhedron (usually an icosahedron), or more general "meshless" techniques that do not use an underlying grid or mesh. Several of these methods are available in the **rbfsphere** package.

In this tutorial we give examples of the functions available and illustrate the corresponding node sets. We also highlight some of the utility functions in the **rbfsphere** package for dealing with these node sets.

Note that all the functions for generating the node sets return an N-by-3 array, where N is the toal number of nodes and the columns corresponds to the (x,y,z) coordinates of the nodes.

```
LW = 'linewidth'; lw = 1.2; FS = 'FontSize'; fs = 12;
FC = 'FaceColor'; fc = 'y'; EC = 'EdgeColor'; ec = 'none';
vw = [-12 6];
```

Icosahedral Nodes

Icosahedral (or geodesic) nodes are formed by recursively subdividing the 12 triangular faces of an icosahedron, and projecting the nodes to the surface

of the sphere after each subdivision. These node sets are available using the `getIcosNodes` function. There are two types of these nodes that are typically used in applications. Type=0 nodes are generated by recursive bisection of the edges of the base icosahedron. Type=1 nodes are generated by first trisecting the edges of the base icosahedron and then recursive bisection of the edges of the resulting traingle. Both types are available in the `getIcosNodes` nodes function. Here's an example of the function calls:

```
x0 = getIcosNodes(4,0); % Type 0
x1 = getIcosNodes(2,1); % Type 1.
```

The first argument represents the number of subdivisions that should be performed, while the second is the node type. In the case of Type 0 nodes k subdivisions results in $N = 10 \cdot 4^k + 2$ nodes, while for Type 0 nodes $N = 10 \cdot 3^2 \cdot 4^k + 2$. Here are these values for several k:

```
k = 0:8;
N0 = 10*4.^k+2;
N1 = 10*3^2*4.^k+2;
fprintf('    Type 1     Type 2\n');
fprintf('%8d\t%8d\n',[N0;N1])
```

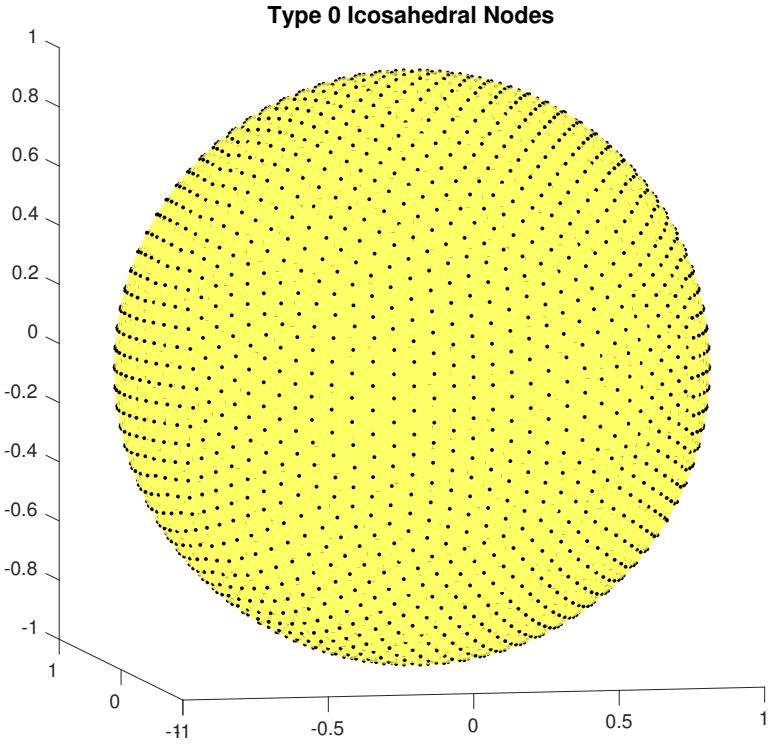
```
    Type 1     Type 2
      12         92
      42        362
     162       1442
     642       5762
    2562      23042
   10242      92162
   40962     368642
  163842    1474562
  655362    5898242
```

An optional output argument for the `getIcosNodes` function is the triangulation of the nodes:

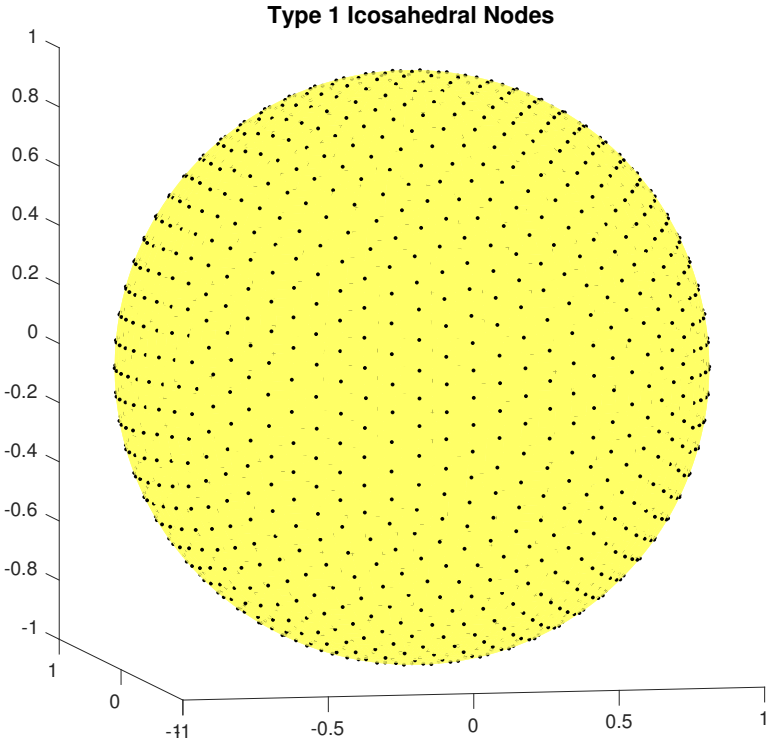
```
[x0,tri0] = getIcosNodes(4,0); % Type 0
[x1,tri1] = getIcosNodes(2,1); % Type 1.
```

The `plotSphNodes` utility function can be used to produce a "visually pleasing" plot of the node sets:

```
plotSphNodes(x0);
title('Type 0 Icosahedral Nodes',FS,fs); view(vw);
```

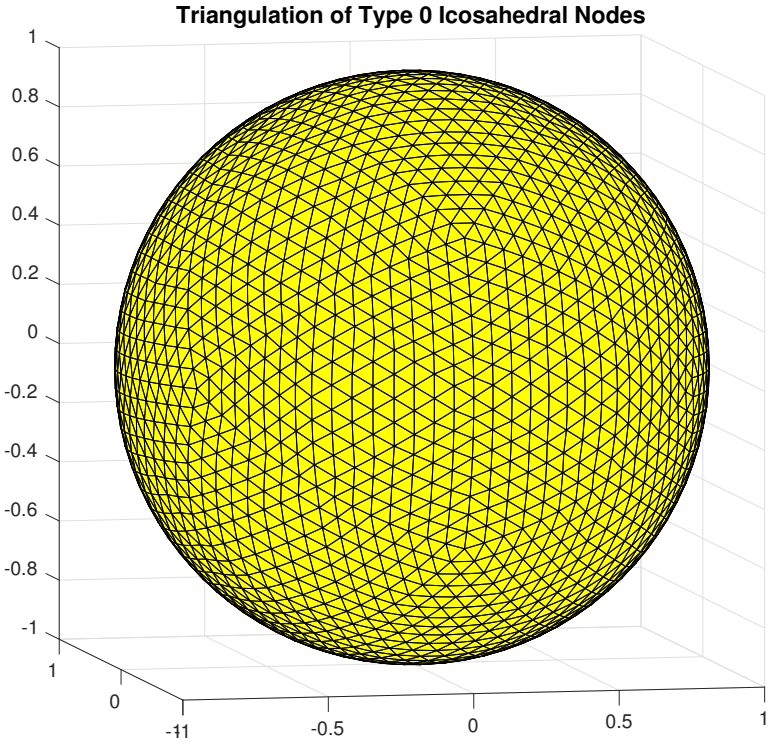



```
plotSphNodes(x1);
title('Type 1 Icosahedral Nodes',FS,fs); view(vw);
```

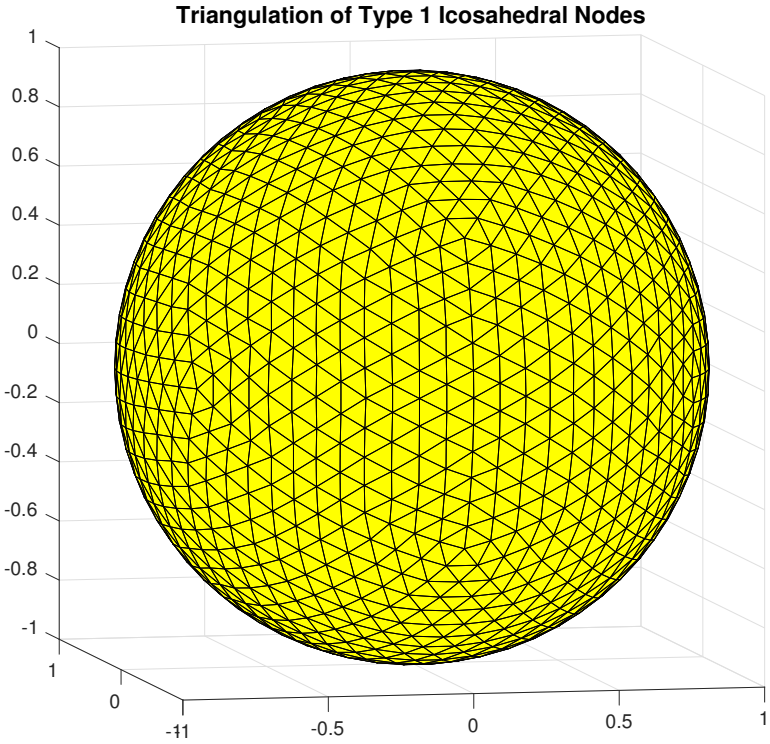


You can get a better idea for how the nodes are arranged by plotting the triangulation of the nodes:

```
trisurf(tri0,x0(:,1),x0(:,2),x0(:,3),FC,fc);
title('Triangulation of Type 0 Icosahedral Nodes',FS,fs);
axis equal; view(vw)
```

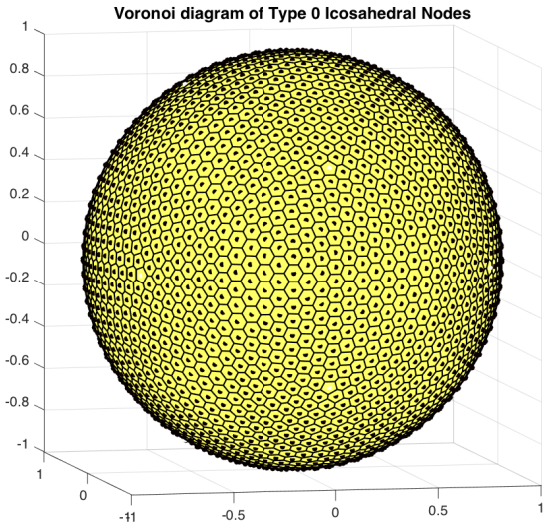



```
trisurf(tri1,x1(:,1),x1(:,2),x1(:,3),FC,fc);
title('Triangulation of Type 1 Icosahedral Nodes',FS,fs);
axis equal; view(vw);
```

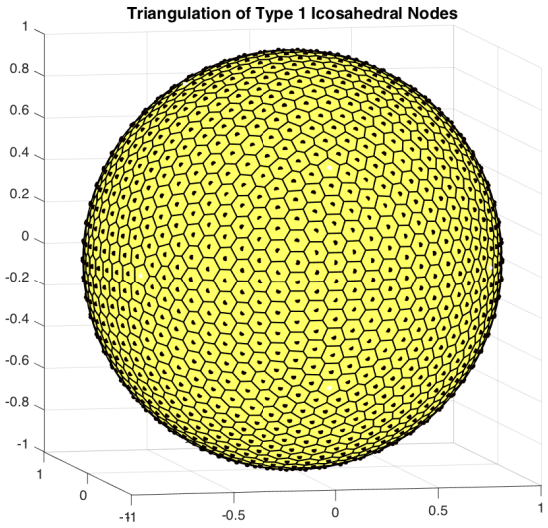


Alternatively, the structure can also be seen by plotting the Voronoi diagram of the nodes (or the dual of the Delaunay triangulation). The function in the **rbfsphere** package for this operation is `voronoiSph`:

```
voronoiSph(x0);
title('Voronoi diagram of Type 0 Icosahedral Nodes',FS,fs), view(vw)
```




```
voronoiSph(x1);
title('Triangulation of Type 1 Icosahedral Nodes',FS,fs), view(vw)
```

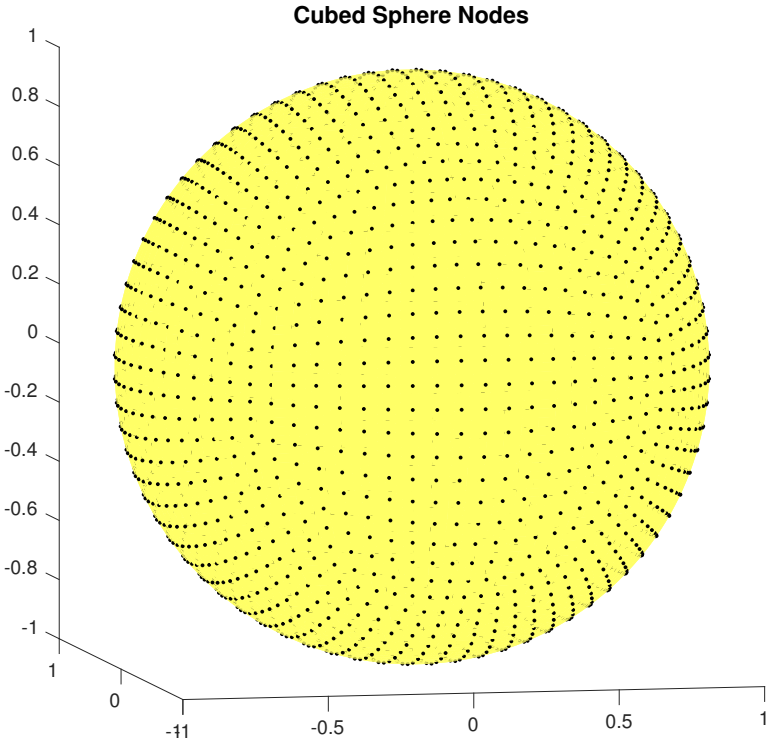


Cubed sphere

The cubed sphere is formed by first forming a tensor product grid on the 6 faces of a cube inscribed in the unit sphere. Then each face is projected to the surface of the sphere. These nodes, which are the vertices of the grid,

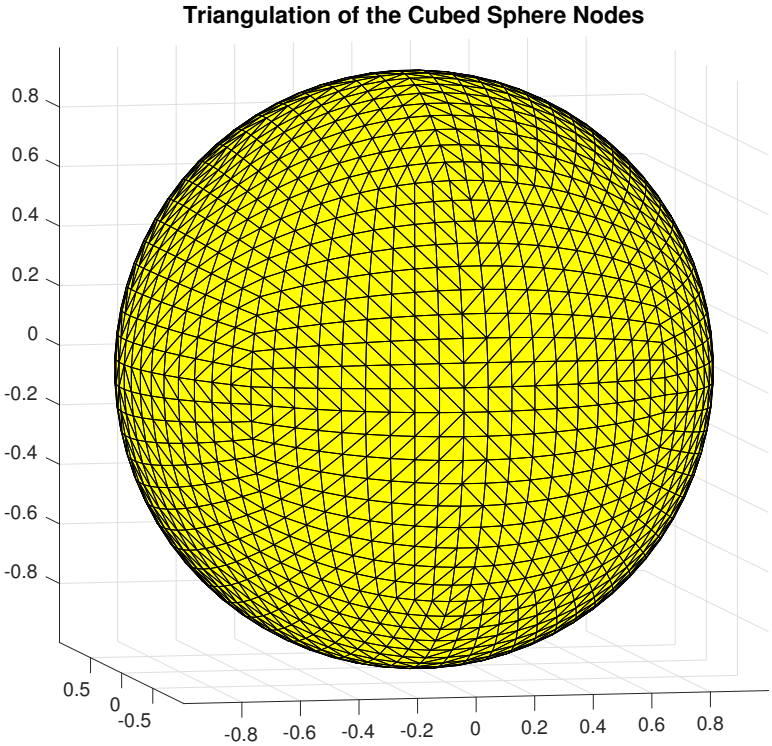
can be obtained from the `getCubedSphNodes` function. This function uses an equiangular projection of the 6 faces. The input to this function is p, the number of nodes in one direction of the grid on each face. This results in a total of $N = 6P^2 - 12P + 8$ nodes. Here is an example for p=20:

```
x = getCubedSphNodes(20);
plotSphNodes(x);
title('Cubed Sphere Nodes',FS,fs); view(vw);
```



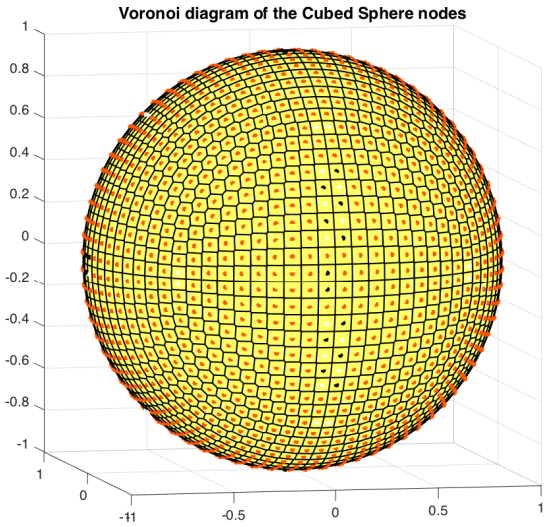
To see the underlying structure of these nodes we can triangulate them and plot the resulting triangles. The utility function `delaunaySph` in the **rbfsphere** package provides a straightforward way to generate such a triangulation:

```
tri = delaunaySph(x);
trisurf(tri,x(:,1),x(:,2),x(:,3),FC,fc);
title('Triangulation of the Cubed Sphere Nodes',FS,fs);
axis equal; view(vw)
```



As above, the Voronoi diagram also reveals the structure:

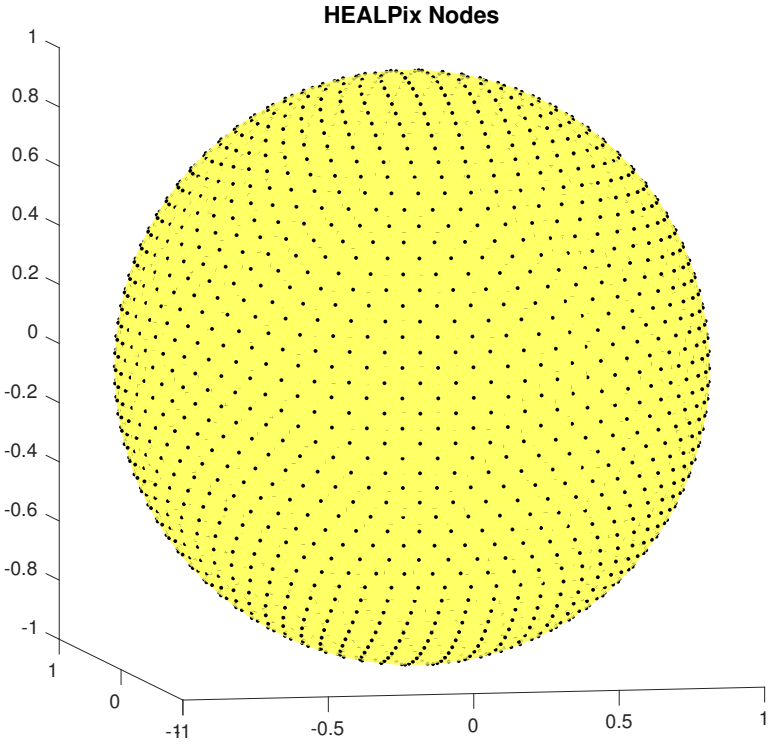
```
voronoiSph(x);
title('Voronoi diagram of the Cubed Sphere nodes',FS,fs);
axis equal; view(vw)
```



HEALPix

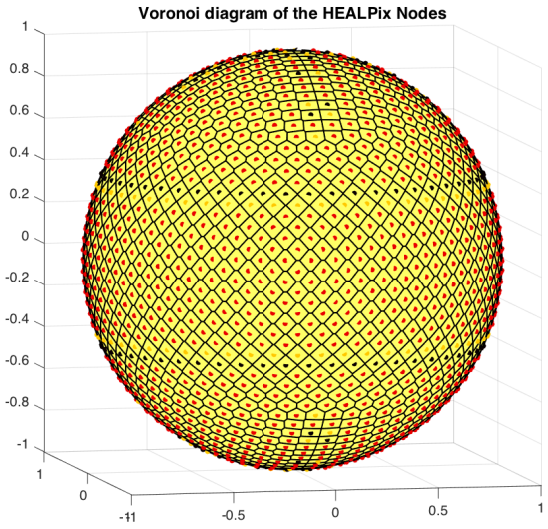
The Hierarchical Equal Area isoLatitude Pixelation (HEALPix) nodes were developed originally by NASA for applications related to satellite missions to measure the cosmic microwave background (CMB), which collect data on a spherical surface. They are a grided set of nodes that provide a quasi-uniform covering. For more details see `http://healpix.sourceforge.net`. These nodes can be obtained from the `getHealPixNodes` function. The number of nodes is parameterized by the input parameter p, with the total number of nodes being $N = 12P^2$. Here is an example for $p = 20$:

```
x = getHEALPixNodes(13);
plotSphNodes(x);
title('HEALPix Nodes',FS,fs); view(vw);
```



As above, the underlying structure of these nodes can be seen from a plot of their Voronoi diagram:

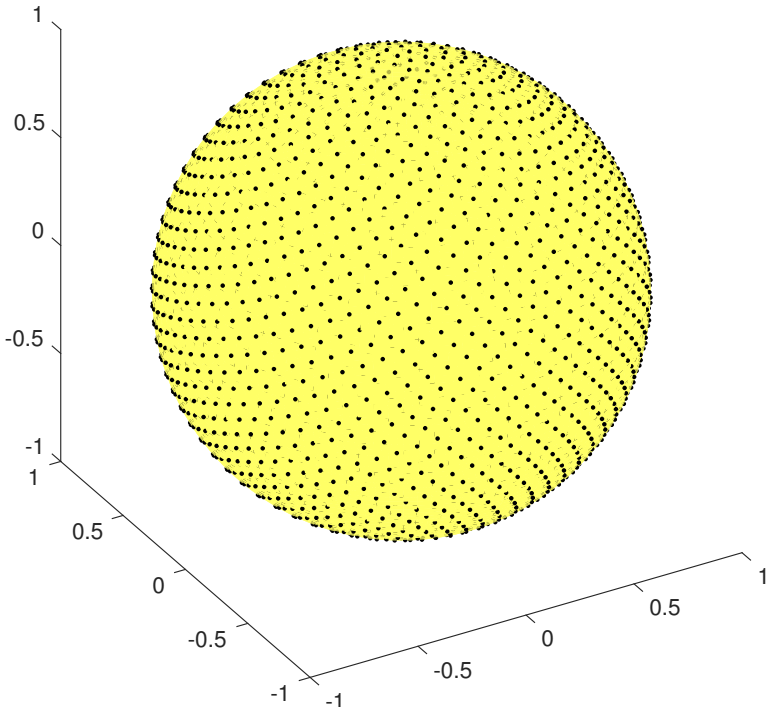
```
voronoiSph(x);
title('Voronoi diagram of the HEALPix Nodes',FS,fs);
axis equal; view(vw)
```



Fibonacci nodes

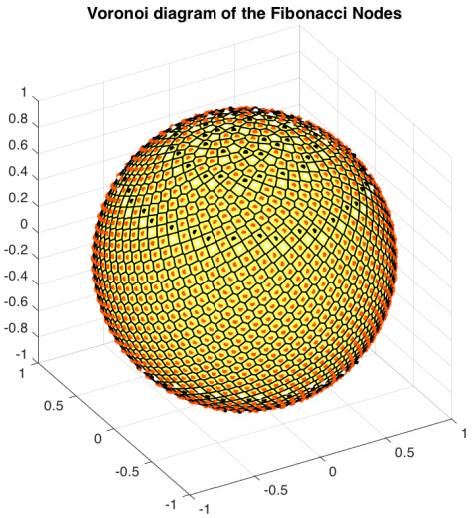
Unlike the three previous node sets the Fibonacci (or phyllotaxis spiral nodes) are not based on a grid. These nodes are developed by a deterministic algorithm that arranges the nodes in a way to mimic certain botanical features (such as the arrangement of seeds on a flower head). These nodes can be obtained from the `getFibonacciNodes` function. The input to this function is N the total number of nodes, which must be an odd number. Here is an example for $N = 2029$ (viewed from as slightly different vantage than the previous examples):

```
x = getFibonacciNodes(2029);
plotSphNodes(x);
title('Fibonacci Nodes',FS,fs); view([-30 30]);
```



There is still structure to these node sets

```
voronoiSph(x);
title('Voronoi diagram of the Fibonacci Nodes',FS,fs);
axis equal; view([-30 30])
```



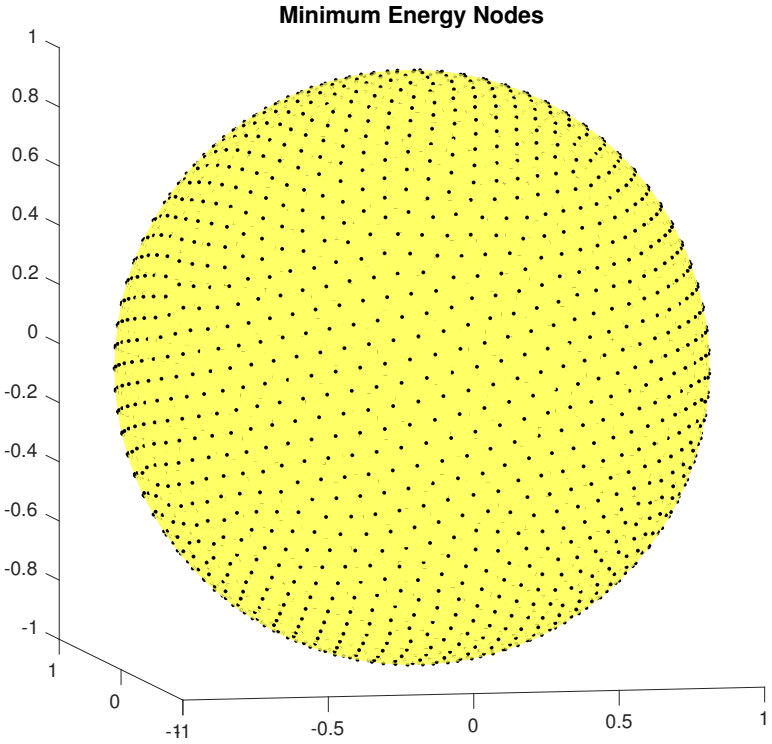
Minimum energry nodes

Imagine randomly placing point charges of the same polarity on the surface of the sphere. The charges will try to maximally separate so that the potenial energy of the system of charges is at minimum. This is the idea behind generating the minimum energy (ME) nodes, which would be the position of the point charges. More specifically, the ME nodes are arranged so that the Reisz energy of the node set $X = \{\mathbf{x}_i\}_{i=1}^N$ with respect to the power q is at a local minimum. The energy is defined as

$$E(\mathbf{X}) = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \|\mathbf{x}_i - \mathbf{x}_j\|^{-q}$$

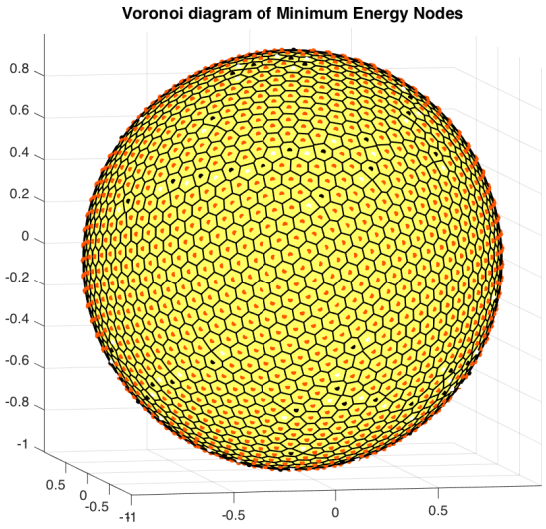
Computing these nodes requires solving a non-linear optimization that can take significant time. The code `getMinEnergyNodes` can be used to get certain numbers of these nodes that have been pre-computed offline and stored (use `help` on this function for more details). The stored nodes are minimized with respect to the power $q = 2$ in the energy functional, and only a local minimum is actually acheived. Here is an example for $N = 2025$:

```
x = getMinEnergyNodes(2025);
plotSphNodes(x);
title('Minimum Energy Nodes',FS,fs); view(vw);
```



Unlike the other node sets discussed above, there is no discernible pattern can be observed for these nodes:

```
voronoiSph(x);
title('Voronoi diagram of Minimum Energy Nodes',FS,fs), view(vw)
```



Maximum determinant nodes

The maximum determinant (MD) nodes are also computed using an optimization technique. The idea is to arrange the nodes so that the determinant of a Grammian matrix related to spherical harmonics is maximized. They can be viewed as a type of Fekete node set for the sphere for the spherical harmonic basis. The code `getMaxDetNodes` can be used to get certain numbers of these nodes that have been pre-computed offline and stored (use `help` on this function for more details). The stored nodes were computed by Prof. Womersley and downloaded from

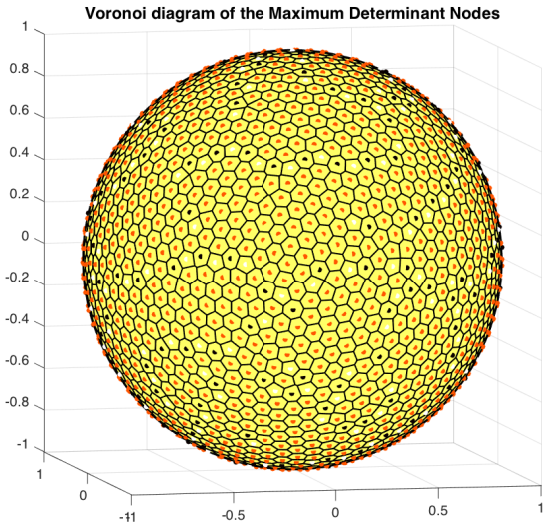
`http://web.maths.unsw.edu.au/~rsw/Sphere/Extremal/New/index.html`

. Here is an example for $N = 2025$:

```
x = getMaxDetNodes(2025);
plotSphNodes(x);
title('Maximum Determinant Nodes',FS,fs); view(vw);
```

Comparing to the ME nodes, we see the MD nodes are less regular, but still quasi-uniformly distributed. As with the ME nodes, there is not a preceptible pattern with the MD nodes:

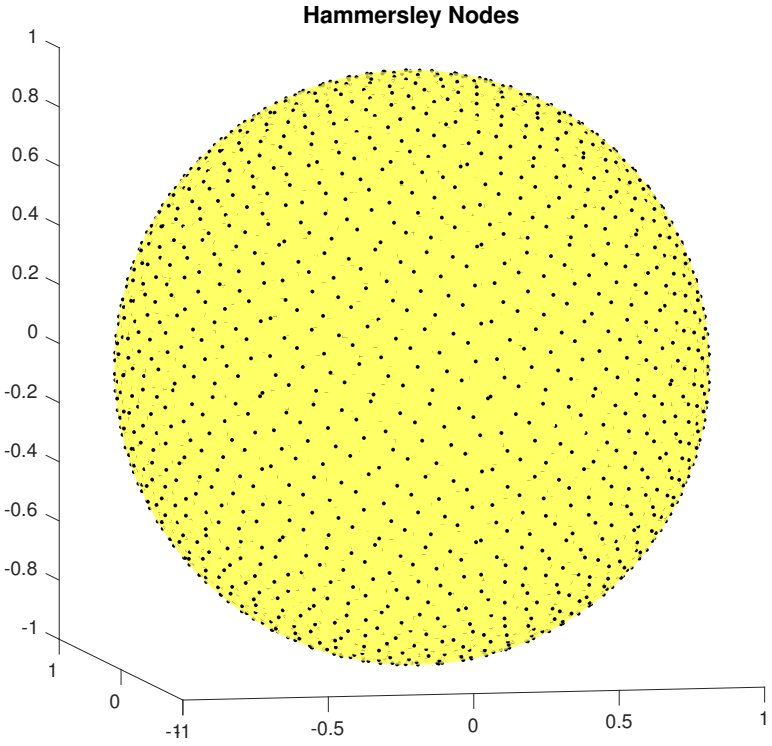
```
voronoiSph(x);
title('Voronoi diagram of the Maximum Determinant Nodes',FS,fs);
axis equal; view(vw);
```



Hammersley nodes

The last set of nodes available from the `rbfsphere` package are not quasi-uniform, but instead provide a "low-discrepancy" sequence for the sphere. These nodes are available from the `getHammersleyNodes` function. Here is an example for $N = 2025$:

```
x = getHammersleyNodes(2025);
plotSphNodes(x);
title('Hammersley Nodes',FS,fs); view(vw);
```


Hammersley Nodes
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
1
0
-1
-0.5
0
0.5
1