# Digital Geometry Processing (236329)

## HW 3

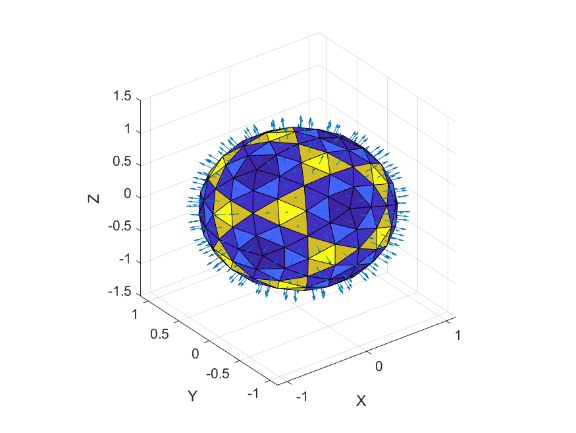
ID1: 204030720

ID2: 203300561

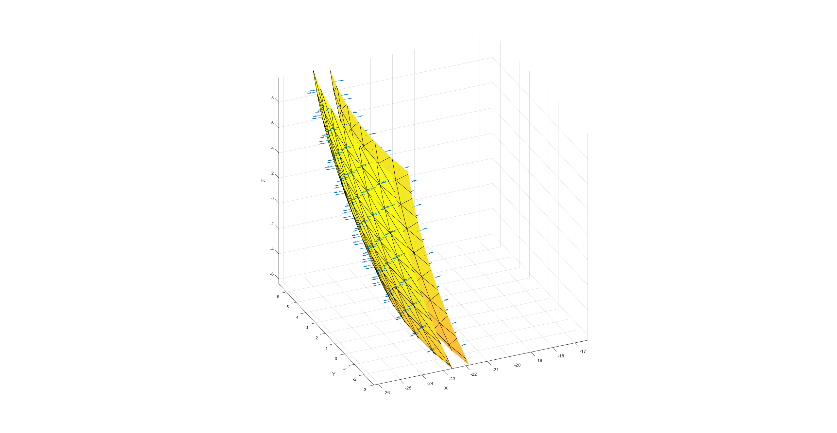
## Git

* All the code can be found in the following Git, matlab branch.
* The python version is under python branch.
* <https://github.com/ManorZ/cs236329.git>

## Q1 – Vector Field Visualization – plot\_vector\_field.m

The function uses our implementation for mesh visualization from HW#2 and adds on top a vector field quiver.  
 Chart

Description automatically generated Chart, surface chart

Description automatically generated Chart, surface chart

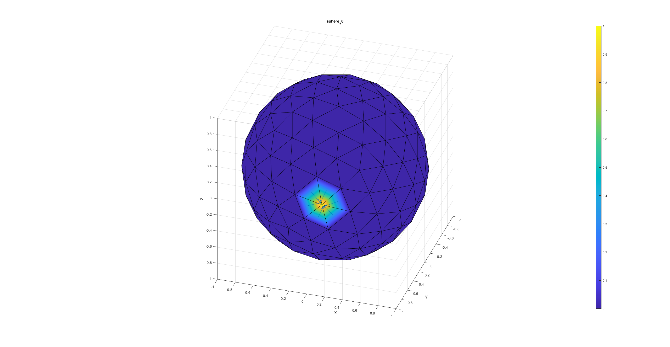
Description automatically generated Chart, surface chart

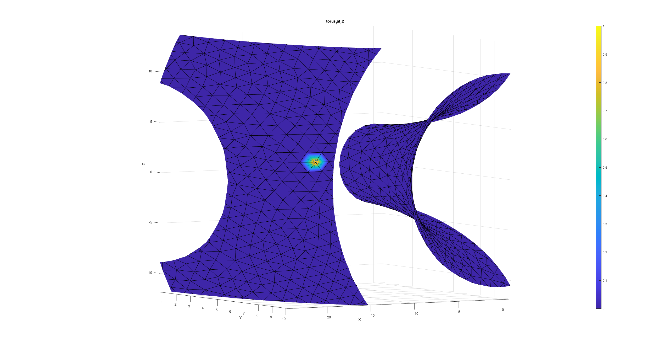
Description automatically generated

## Q2 – Discrete Differential Operators

### Q2.1 – Discrete Grad

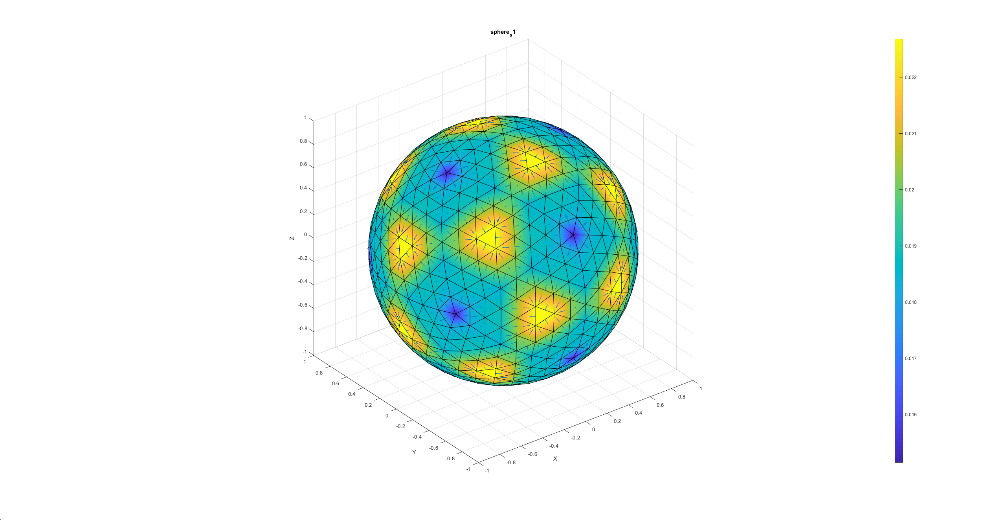
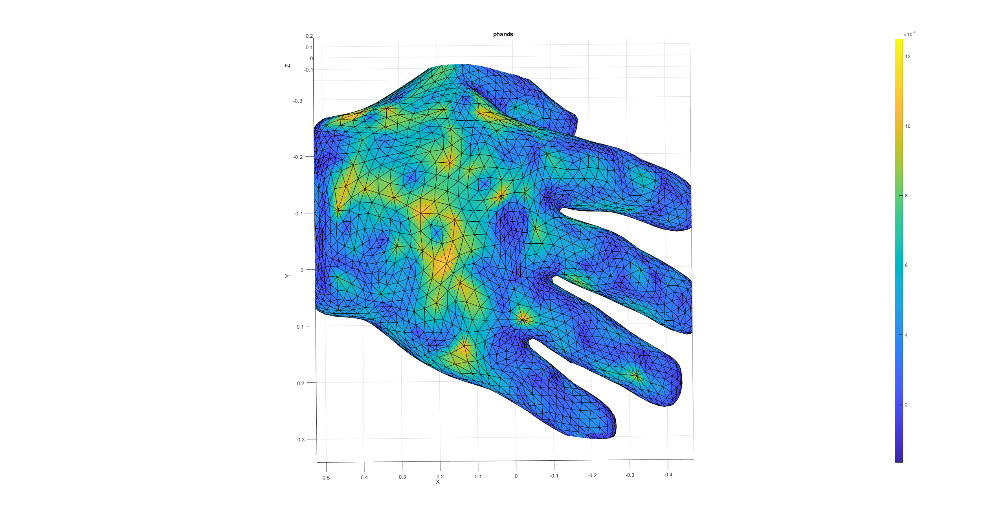
The discrete gradient calculation is implemented here: *calc\_grad.m*  
We experimented with several scalar functions on the vertices:

* Pulse:   
  Chart

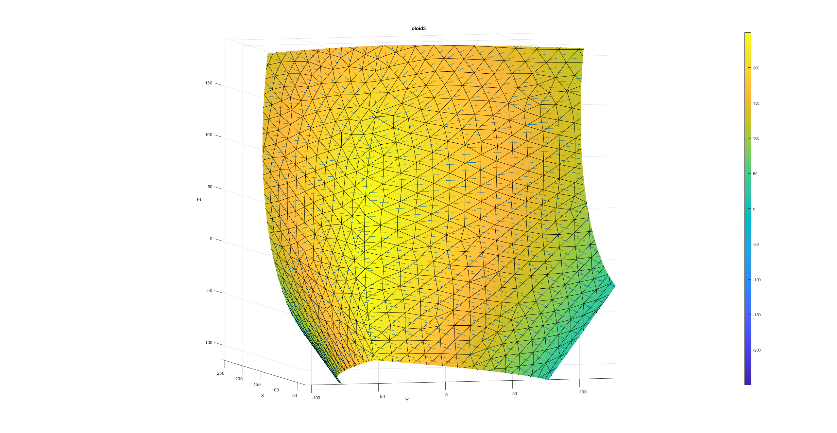
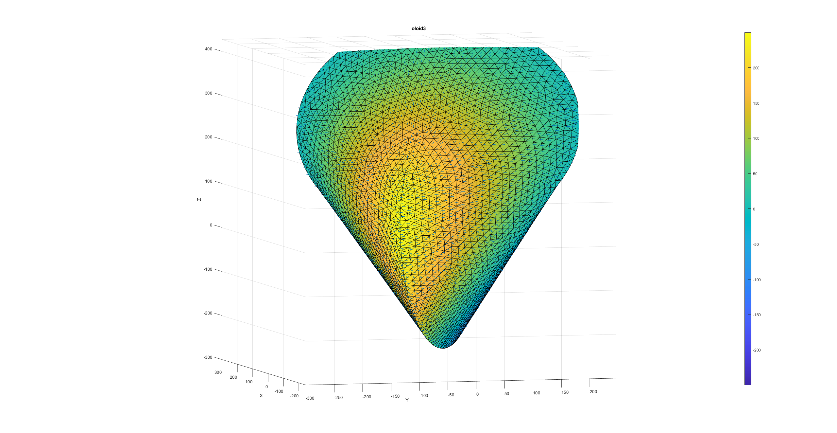
  Description automatically generated with low confidenceA picture containing text

  Description automatically generatedA picture containing chart

  Description automatically generatedChart, radar chart

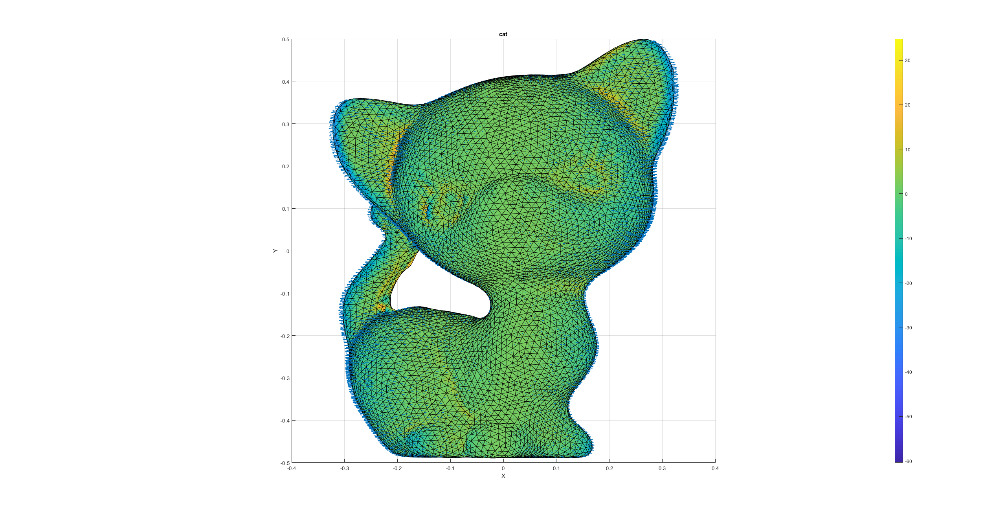
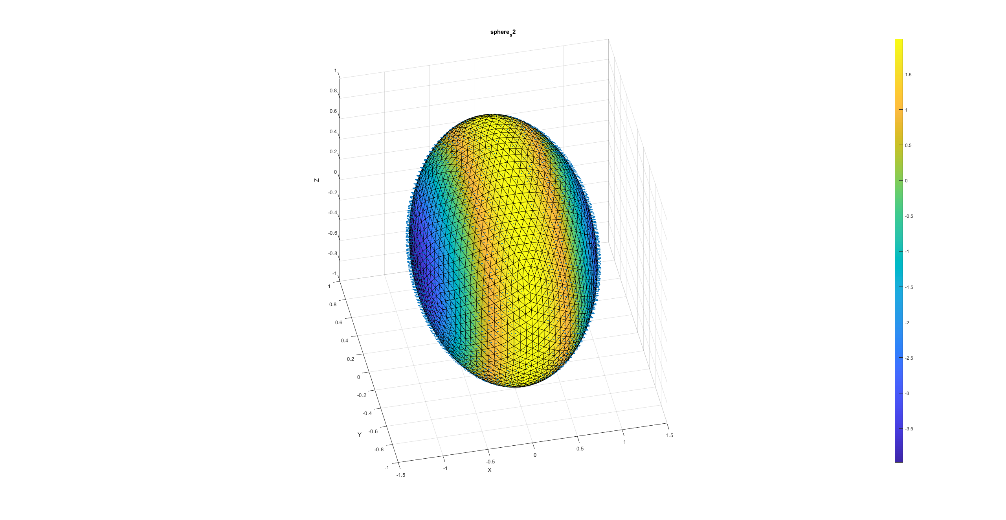
  Description automatically generated
* Barycentric Area:  
  
* X coords:   
  Chart

  Description automatically generatedChart

  Description automatically generated

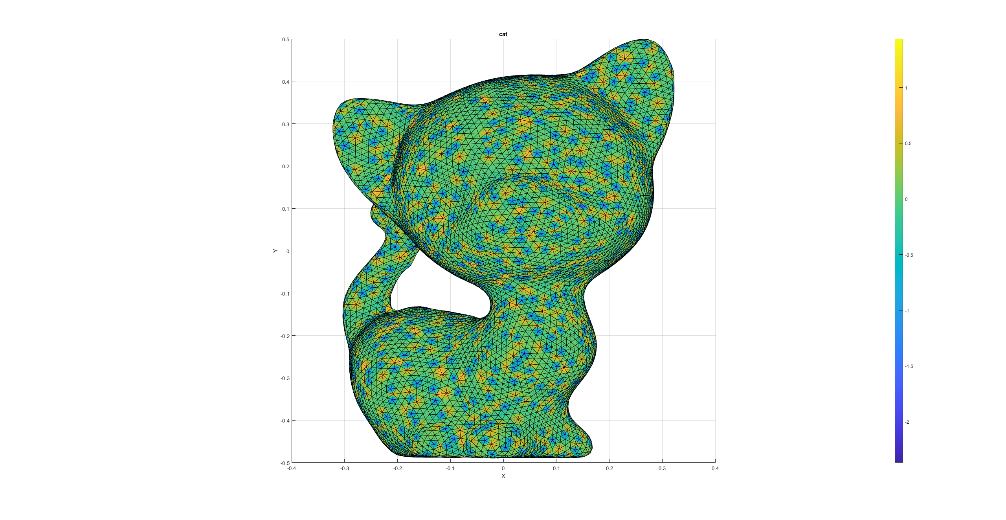
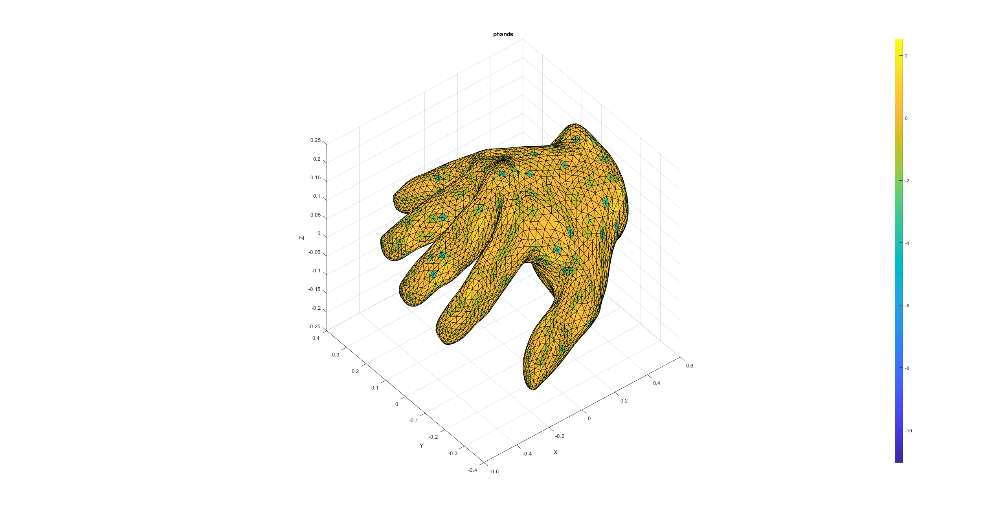
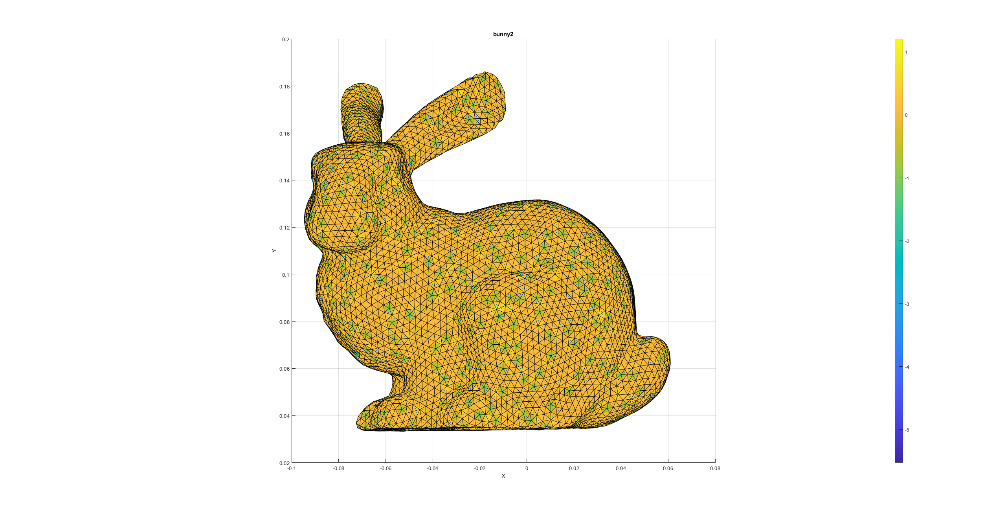
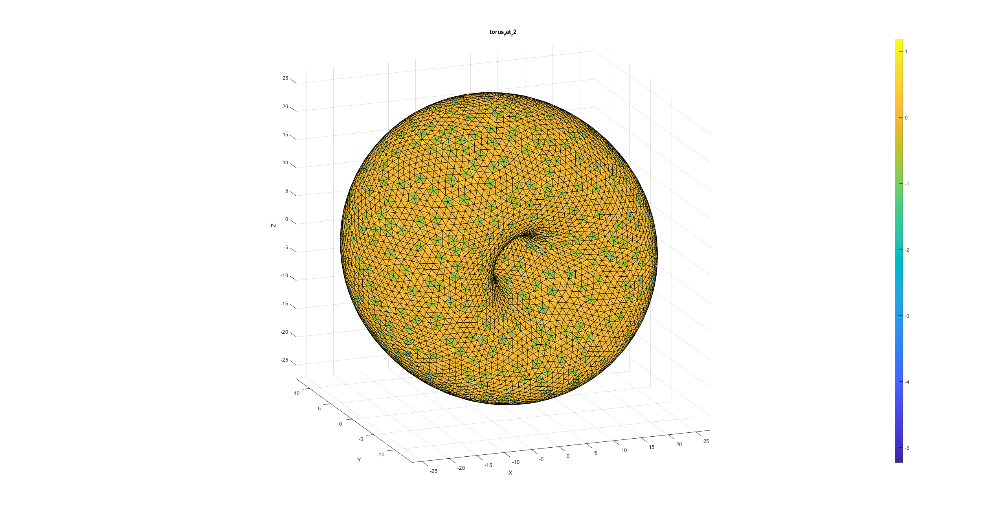
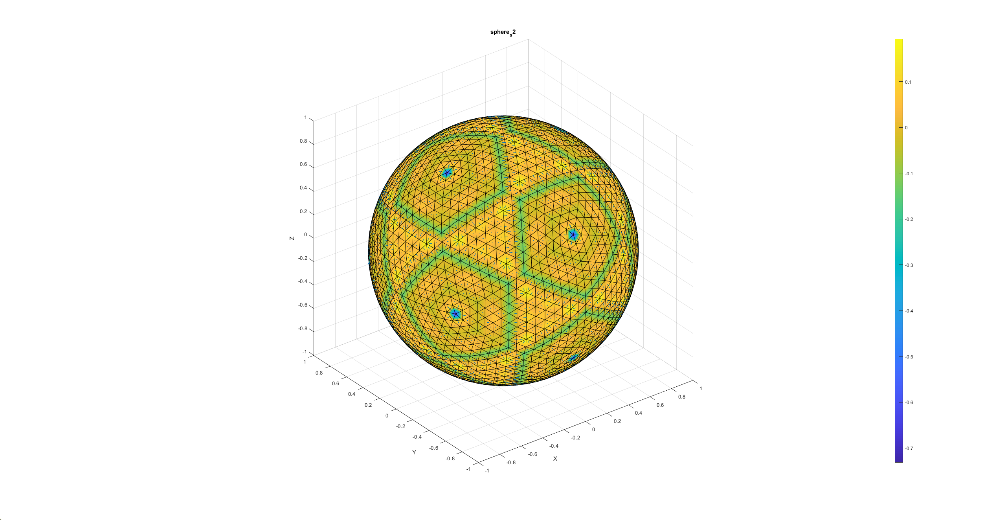
### Q2.2 – Discrete Div

The discrete divergence calculation is implemented here: *calc\_div.m*  
We bring here a visualization for the vector function on faces: where are each face center.



### Q2.3 – Discrete Laplacian

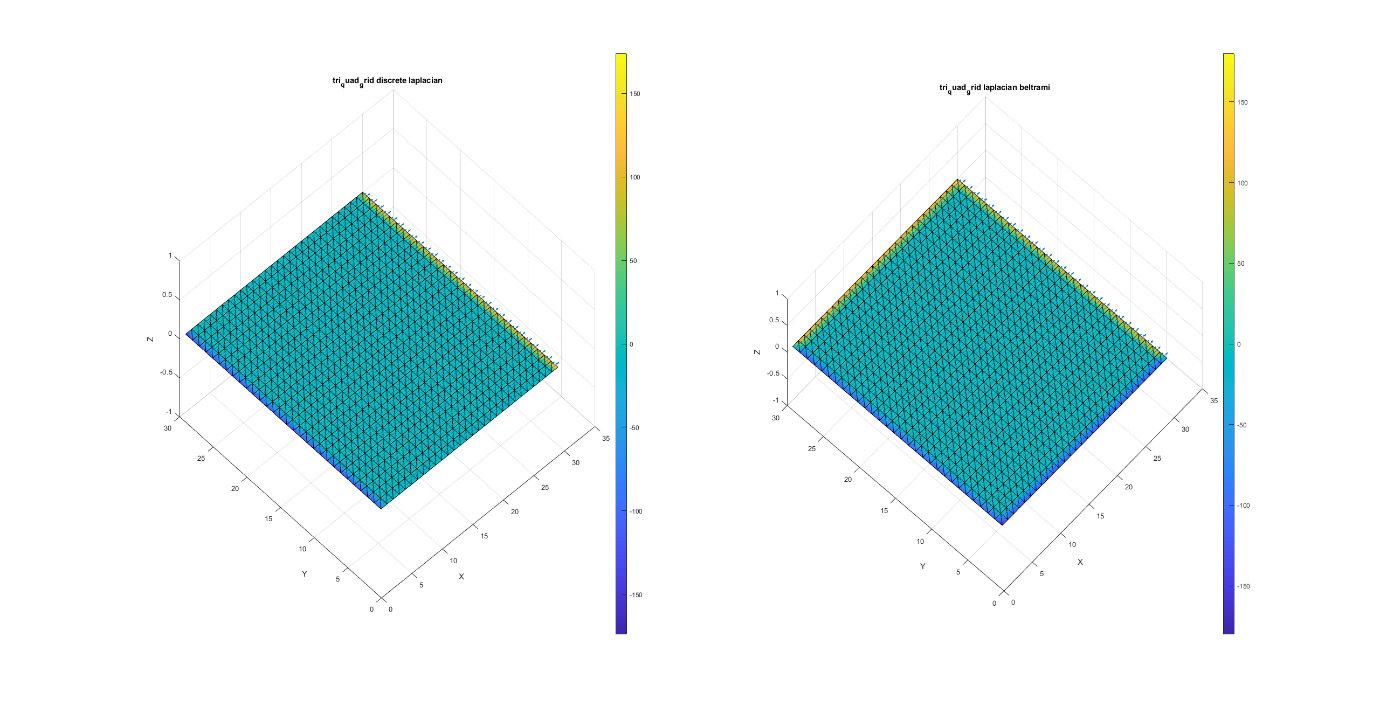
The discrete divergence calculation is implemented here: *calc\_lap.m*  
We bring here a visualization for the gradient function on faces for a barycentric vertex area scalar function on vertices.



### Q2.4 – Laplace Beltrami

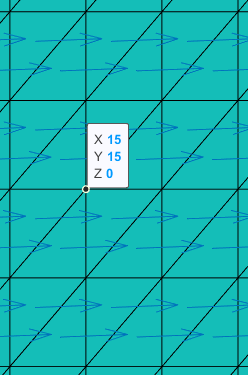
The Laplace Beltrami is calculated in the function: *calc\_lap\_beltrami.m*  
We run it on various mashes and measure the MSE error vs. the discrete Laplacian operator on the scalar function on vertices: .  
  
We noticed that for some reason our algorithm does not work on 2d surfaces (disk, tri\_quad\_grid). We suspect it is because in the Laplacian Beltrami we assume no boundaries.  
In *calc\_lap\_beltrami.m* we assume regular edges only, a.k.a, two adjoint faces per edge.

reading hw2\_data\bunny2.off ... calc lap ... calc lap beltrami ... done. ||Laplacian(F) - Laplacian-Beltrami(F)||^2 = 0.000039  
reading hw2\_data\cat.off ... calc lap ... calc lap beltrami ... done. ||Laplacian(F) - Laplacian-Beltrami(F)||^2 = 0.000004  
reading hw2\_data\oloid3.off ... calc lap ... calc lap beltrami ... done. ||Laplacian(F) - Laplacian-Beltrami(F)||^2 = 0.000000  
reading hw2\_data\phands.off ... calc lap ... calc lap beltrami ... done. ||Laplacian(F) - Laplacian-Beltrami(F)||^2 = 0.000001  
reading hw2\_data\sphere\_s0.off ... calc lap ... calc lap beltrami ... done. ||Laplacian(F) - Laplacian-Beltrami(F)||^2 = 0.000000  
reading hw2\_data\sphere\_s1.off ... calc lap ... calc lap beltrami ... done. ||Laplacian(F) - Laplacian-Beltrami(F)||^2 = 0.000000  
reading hw2\_data\sphere\_s2.off ... calc lap ... calc lap beltrami ... done. ||Laplacian(F) - Laplacian-Beltrami(F)||^2 = 0.000000  
reading hw2\_data\torus\_fat\_r2.off ... calc lap ... calc lap beltrami ... done. ||Laplacian(F) - Laplacian-Beltrami(F)||^2 = 0.000000  
reading hw2\_data\disk.off ... calc lap ... calc lap beltrami ... done. ||Laplacian(F) - Laplacian-Beltrami(F)||^2 = 49661.878919  
reading hw2\_data\tri\_quad\_grid.off ... calc lap ... calc lap beltrami ... done. ||Laplacian(F) - Laplacian-Beltrami(F)||^2 = 944.259498

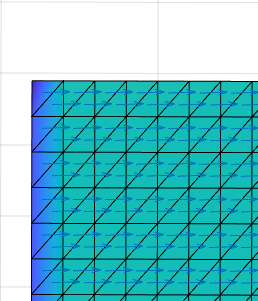
The following plot assure our assumption regarding the boundaries. For each non-boundary vertex, we get 0 Laplacian, as expected. On the edges, we get something else, and it is different between the Laplace-Beltrami and the Discrete Laplacian.  


### Q2.5

Each couple of non-boundary vertices, forms two **rights triangles**, with in each side (see figure below). Therefore, the weights are zeroed, as so the Laplacian.

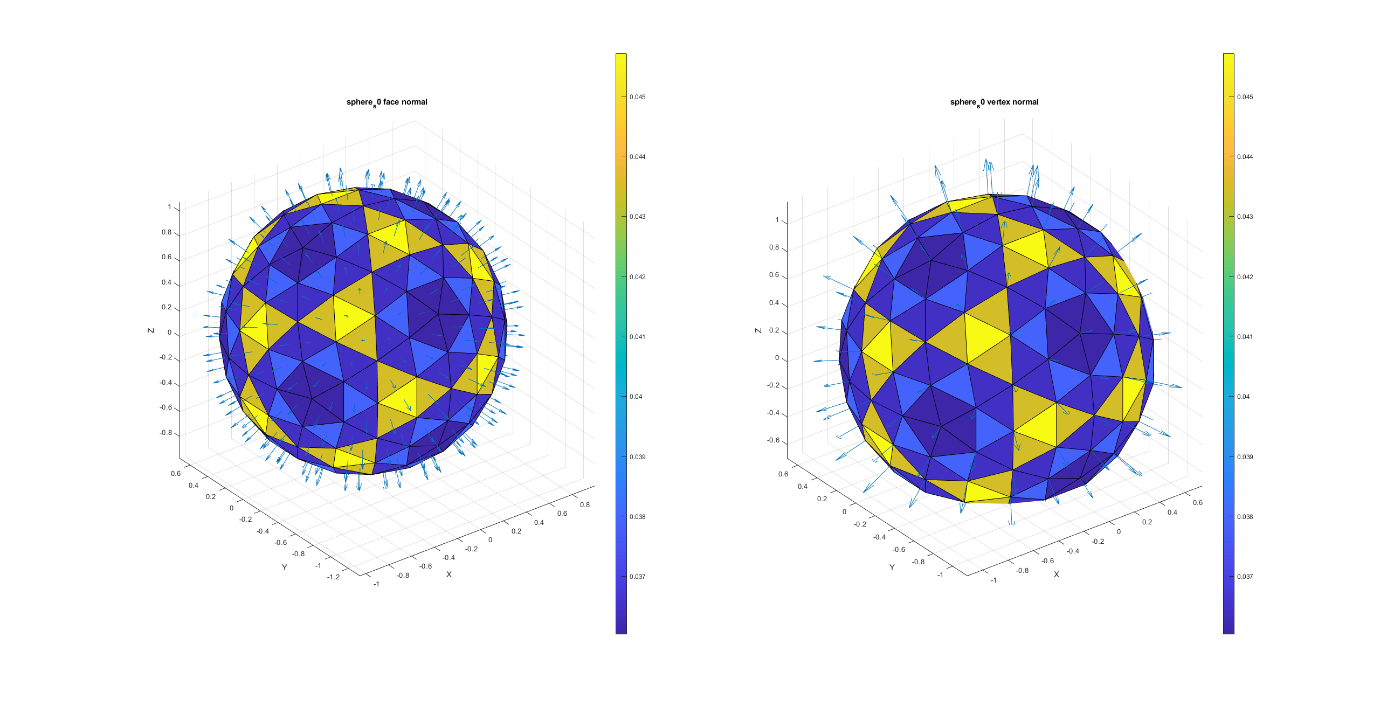
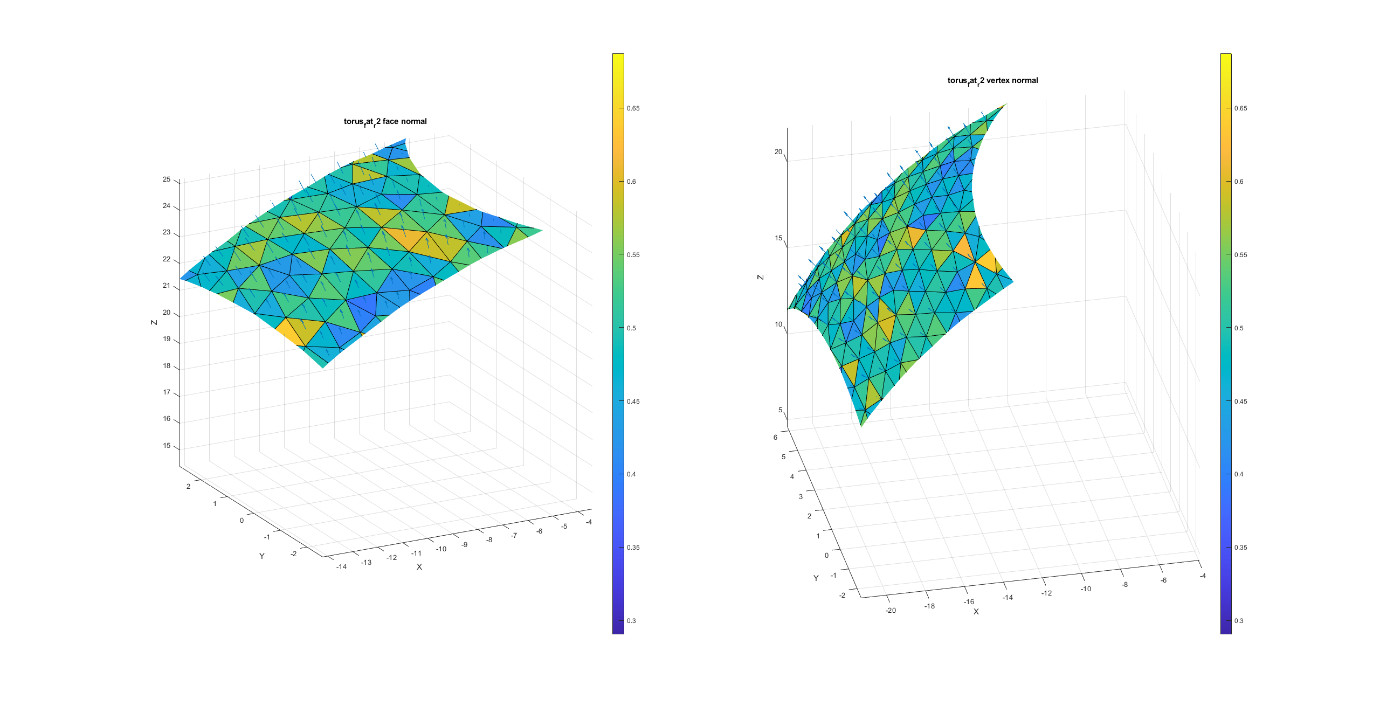




For boundary vertices, the Laplacian-Beltrami formula is not well defined, for example:  


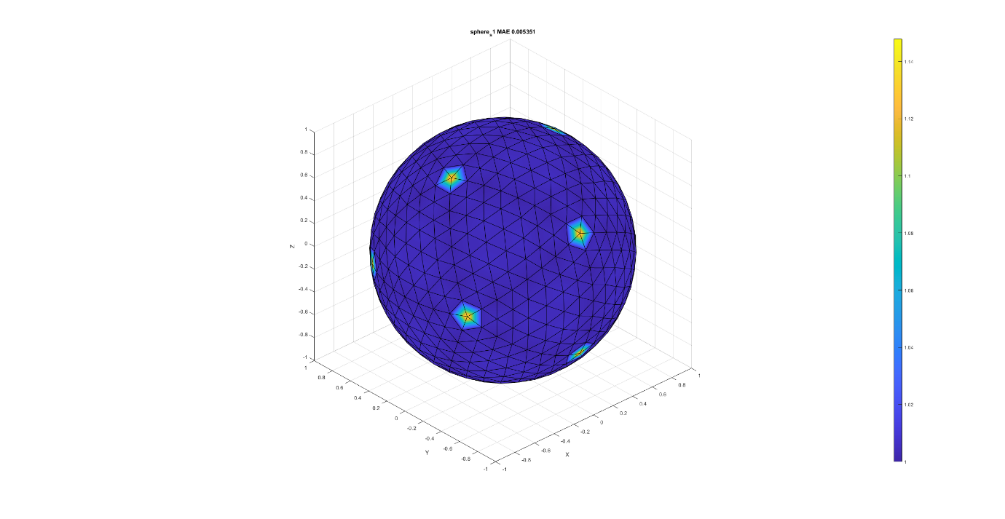
And therefore, we’ve got weird results in section Q2.4 for meshes with boundary.

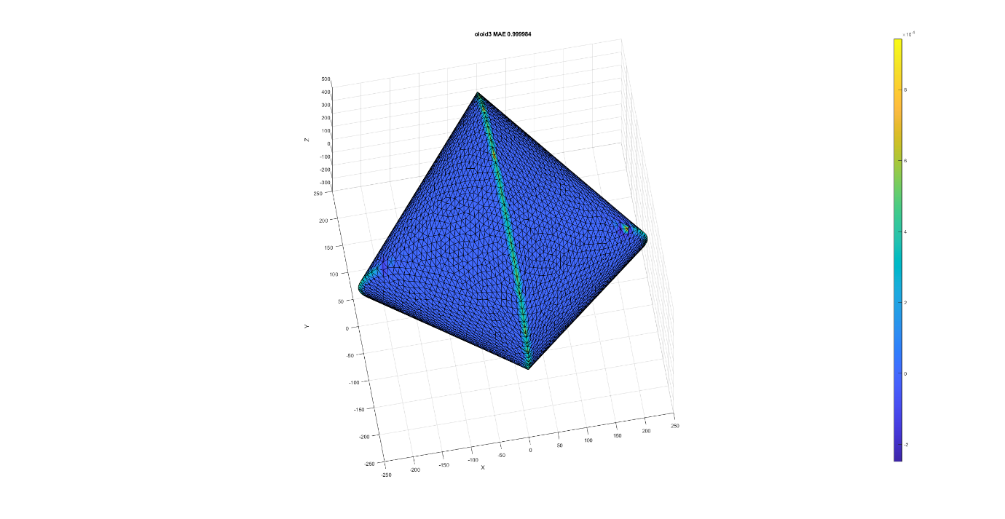
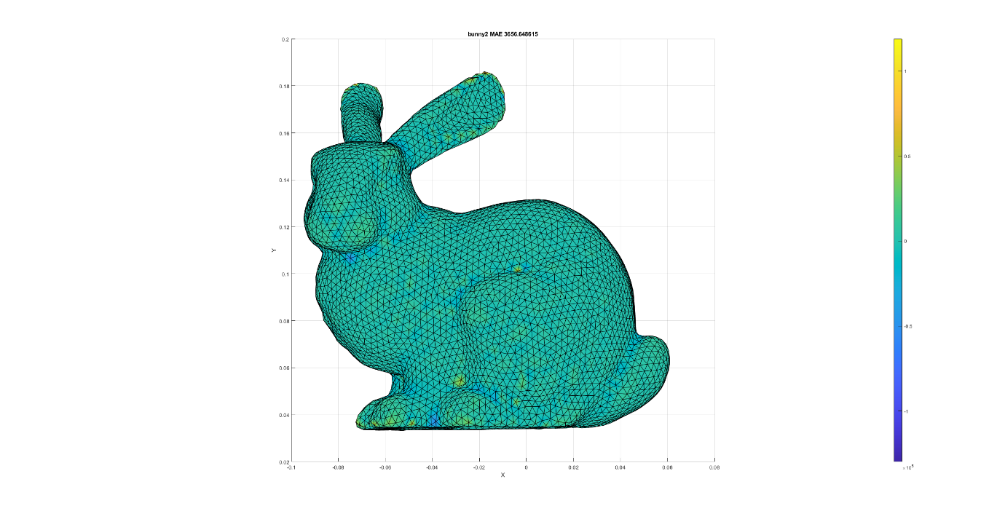
## Q3 – Vertex Normal



## Q4 – Curvature

### Q4.1 Gauss Curvature

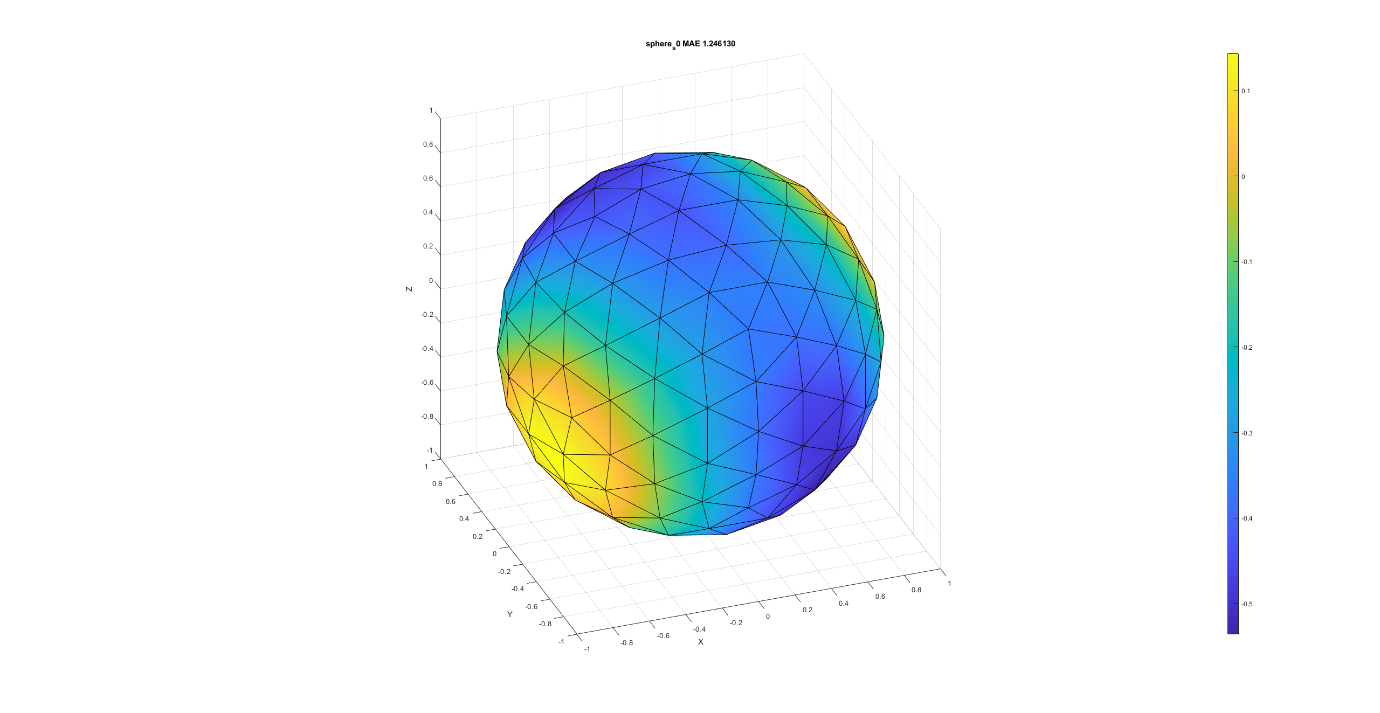
The code is in *calc\_gauss\_curvature.m*.  
We tested our algorithm by calculating the curvature on a unit sphere, expecting 1 everywhere.  
And indeed, we get almost 1 everywhere:  


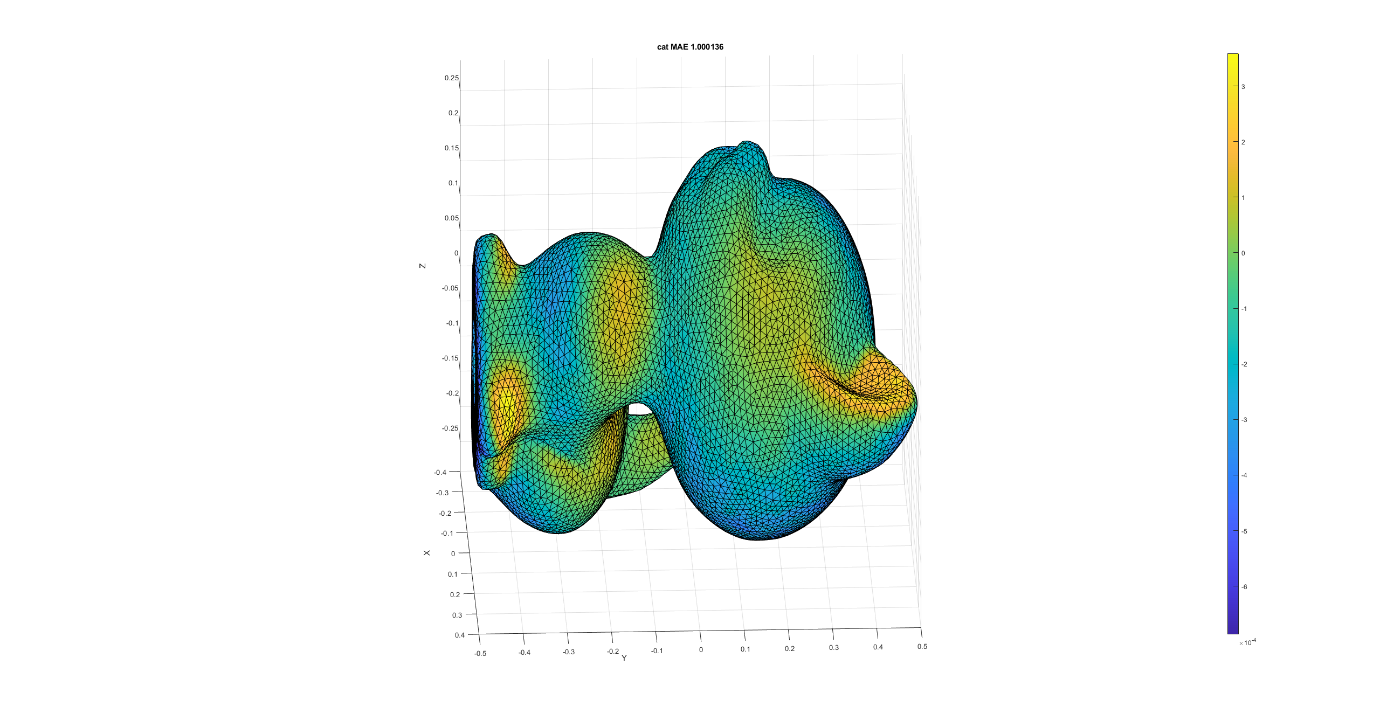
Chart

Description automatically generatedA picture containing logo

Description automatically generated

### Q4.2 Mean Curvature

The code is in *calc\_mean\_curvature.m*.  
We use the Laplacian-Beltrami, but instead of a scalar function we use the coordinates of each vertex. The idea is to measure ‘how fast change in each vertex from its neighbors’.  
We then calculate each vertex normal, and inner-producting it with the Laplacian.  
The results on a unit sphere are somewhat weird, not sure why.  
We expect it to be 1 everywhere on a unit sphere, like the Gauss Curvature.  
In a 2nd thought, it might be ok, up to a constant multiplicative scaling factor, since for other meshes, the cat for instance, we get higher values for ‘curvy’ areas, and lower values for ‘flat’ areas.  
  
A picture containing chart

Description automatically generatedChart, surface chart

Description automatically generated