Finite Element Methods for the Poisson Equation and its Applications

Charles Crook

July 30, 2013

Abstract

The finite element method is a fast computational method that also has a solid mathematical theory behind it. We visualize the finite element approximation to the solution of the Poisson equation on different type of domains and observe the corresponding order of convergence. As a real-world application of the Poisson equation, we use the finite element approximation to distinguish different cartoon characters. Furthermore, we use the weighted Poisson equation arising from the axisymmetric Poisson equation to distinguish axisymmetric three-dimensional objects as well.

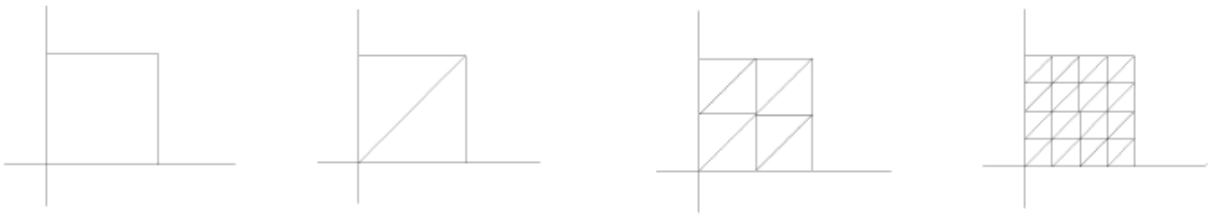
1 Poisson Equation with Homogenous Dirichlet Boundary Conditions

Given the problem, find the function u that satisfies the following,

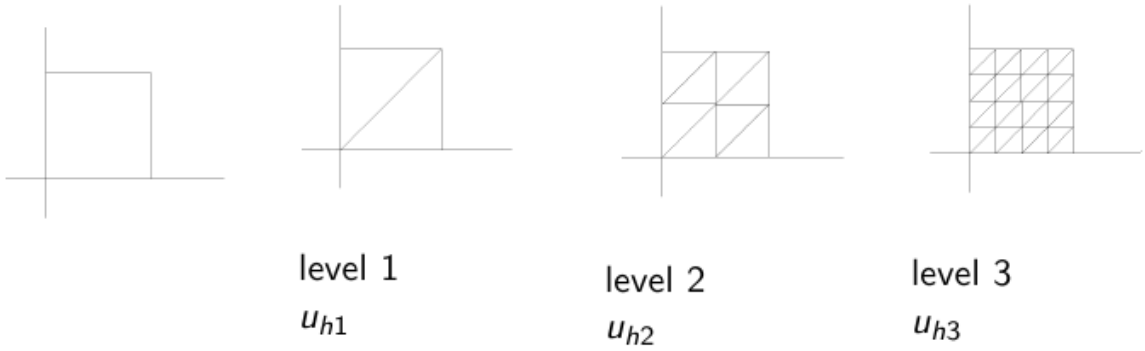
$$\begin{aligned} -\Delta u &= F \text{ in } \Omega \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned}$$

where F is given data. The goal is to find a 'good' approximation to the exact solution u.

2 The Finite Element Method



The Finite Element Method is called so, since it starts by taking the inputted domain and performing calculation to make the domain into now finitely man elements. The elements in my case are triangles.



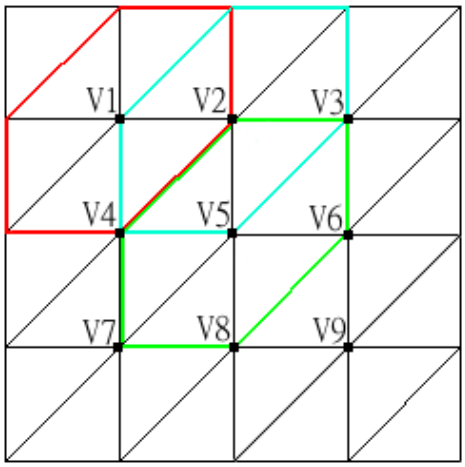
With each mesh generated, that mesh has exactly one finite element approximation u_h. As the meshes become more refined, the u_h changes to better approximate the exact solution. The main idea of the finite element method is that it changes the infinite dimension problem into a finite dimension problem. With the problem now converted into a finite dimension, it can now be changed to a matrix system, using basis functions, for finding the solution.

3 Finite Elements Subspace P_1^0

We now define the space we are working in to be P_1^0, which means after the domain has had the finite element method applied to it, each element inside has a plane above it represented by the function $ax+by+c$, and the collection of all functions over the domain has to be continuous and have zero boundary conditions as well. The dimension of this space, P_1^0, is the total number of

interior nodes.

4 Basis of P_1^0



Now, we construct a Φ_i for each interior node v_i using the formula below,

$$\Phi_i(v_j) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$$

For example, to construct Φ_1, you would have to plug in each interior node and find the value at each. At V_1, $\Phi_1 = 1$, and at $V_2, V_3, ..., V_9$, $\Phi_1 = 0$. Also, with the given initial condition, $\Phi_1 = 0$ on the boundary. Now connecting each of the nodes together requires the use of the plane function, $ax + by + c$, from earlier.

5 Infinite Dimensional Problem $\Rightarrow$ Finite Dimensional Problem

The infinite dimension problem, shown below

Find $u \in H_0^1(\Omega)$ s.t.

$$\int_\Omega \bigtriangledown u \bigtriangledown v = \int_\Omega Fv \qquad \forall v \in H_0^1(\Omega)$$

where $H_0^1 = \{f : \int_\Omega f^2 < \infty,\ \int_\Omega [\bigtriangledown f]^2 < \infty,\ f = 0 \text{ on } \partial\Omega\}$ is now converted to the finite dimensional problem, through the use of the Finite Element

Subspace, is shown below,

Find $u_h \in P_1^0$ s.t.

$$\int_\Omega \bigtriangledown u_h \bigtriangledown v_h = \int_\Omega F v_h \qquad \forall v_h \in P_1^0$$

6 Finite Dimensional Problem $\Rightarrow A\vec{x} = \vec{b}$

We convert the new finite problem down to a matrix system by using $u_h = \sum_{i=1}^{n} c_i \Phi_i$.

$$\int_\Omega \bigtriangledown u_h \bigtriangledown v_h = \int_\Omega F v_h \qquad \forall v_h \in P_1^0$$

$$\int_\Omega \bigtriangledown (\sum_{i=1}^{n} c_i \Phi_i) \bigtriangledown \Phi_j = \int_\Omega F \Phi_j \qquad \forall j = 1, 2, \ldots n$$

$$\sum_{i=1}^{n} c_i \int_\Omega \bigtriangledown \Phi_i \bigtriangledown \Phi_j = \int_\Omega F \Phi_j \qquad \forall j = 1, 2, \ldots n$$

$$\sum_{i=1}^{n} \underline{c_i} \quad \underline{\int_\Omega \bigtriangledown \Phi_i \bigtriangledown \Phi_j} = \underline{\int_\Omega F \Phi_j} \qquad \forall j = 1, 2, \ldots n$$

$$\underline{A}\ \underline{\vec{c}} = \underline{\vec{b}}$$

Where, the underlined red portion corresponds to the i^{th} entry of $\vec{c}$.
The underlined blue portion indicates the ij^{th} entry of the matrix A.
The underlined green portion is j^{th} entry of $\vec{b}$.

To find more a exact solution, requires a finer mesh. With this, the matrix A will become larger with each mesh. So to find a more exact solution, you will have wait longer for the program to finish.

7 Numerical Example

7.1 Example One

The problem for example one was,

Find a function, u, with the given function, F, such that

$-\Delta u = F$ in Ω

$u = 0$ on $\partial\Omega$

I picked a 'nice' F such that $u = xy(x-1)(y-1)$. By 'nice', I mean where the $u = 0$ on $\partial\Omega$ and u is a polynomial.

mesh	$\parallel u - u_h \parallel$	OoC	$\parallel \bigtriangledown u - \bigtriangledown u_h \parallel$	OoC
1	0.0186579226	——	0.1074034163	——
2	0.0057148929	1.7070	0.0588911882	0.8669
3	0.0015087244	1.9214	0.0301758912	0.9647
4	0.0003824646	1.9799	0.0151826243	0.9910
5	0.0000959512	1.9950	0.0076032633	0.9977
6	0.0000240088	1.9987	0.0038031293	0.9994

The L2 error in the chart is reducing by about 1/4 each mesh. The order of convergence is the ratio of two meshes with respect to the largest edge length. Since the mesh size is decreasing by 1/2 each time, the order of convergence converges to 2.

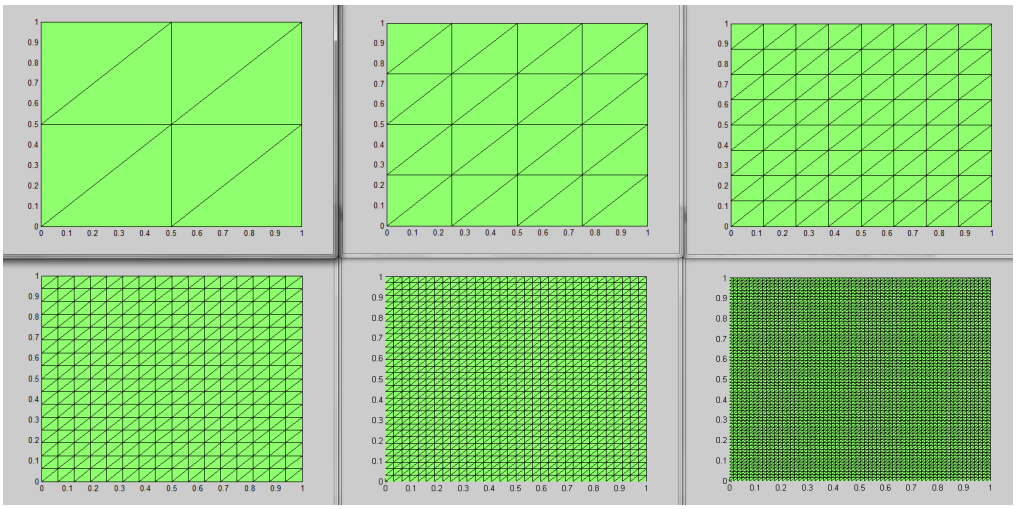


Figure 1: Top View of the six meshes

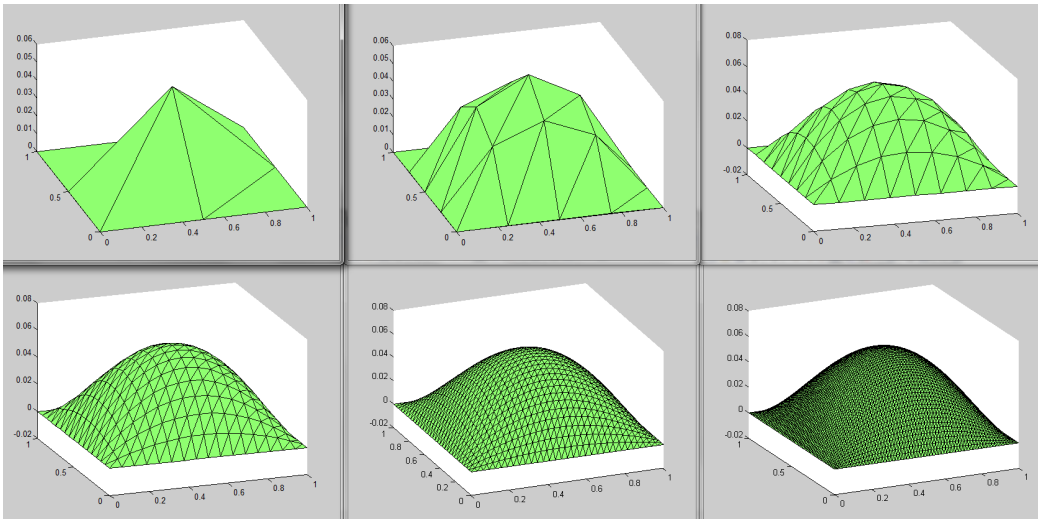


Figure 2: Tilted View of the six meshes

7.2 Example Two

In Example 2, I decided to change the domain to be more complicated, by removing part of the domain.

$u = xy(x-1)(y-1)(x-0.25)(y-0.25)(y-0.75)$

mesh	$\| u - u_h \|$	OoC	$\| \bigtriangledown u - \bigtriangledown u_h \|$	OoC
2	0.0003341186	———	0.0042327447	———
3	0.0001088289	1.6183	0.0024042537	0.8160
4	0.0000293177	1.8922	0.0012455389	0.9488
5	0.0000074747	1.9717	0.0006285406	0.9867
6	0.0000018781	1.9927	0.0003150047	0.9966

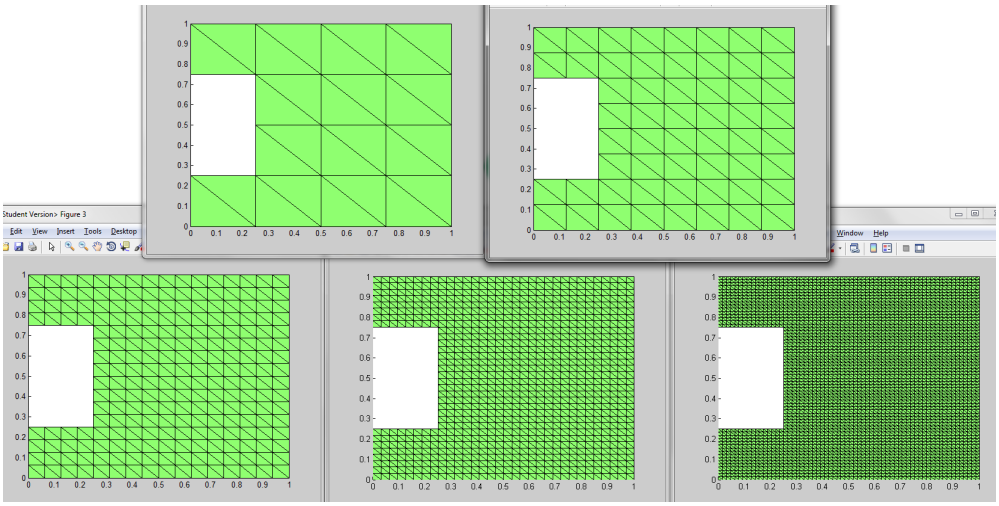


Figure 3: Top View of the five meshes

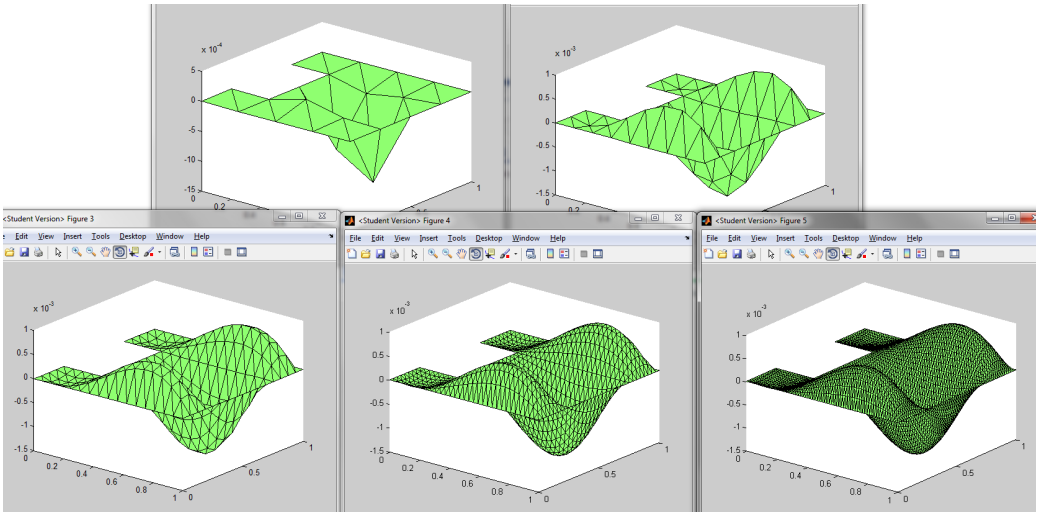


Figure 4: Tilted View of the five meshes

7.3 Example Three

For Example 3, I changed the domain to be more complicated by adding a hole inside it.

$$u = xy(x-1)(y-1)(x-0.25)(y-0.25)(y-0.75)(x-0.5)(x-0.75)$$

mesh	$\| u - u_h \|$	OoC	$\| \bigtriangledown u - \bigtriangledown u_h \|$	OoC
3	0.0000069270	——	0.0001657242	——
4	0.0000021446	1.6915	0.0000916471	0.8546
5	0.0000005696	1.9127	0.0000471180	0.9598
6	0.0000001447	1.9769	0.0000237300	0.9896

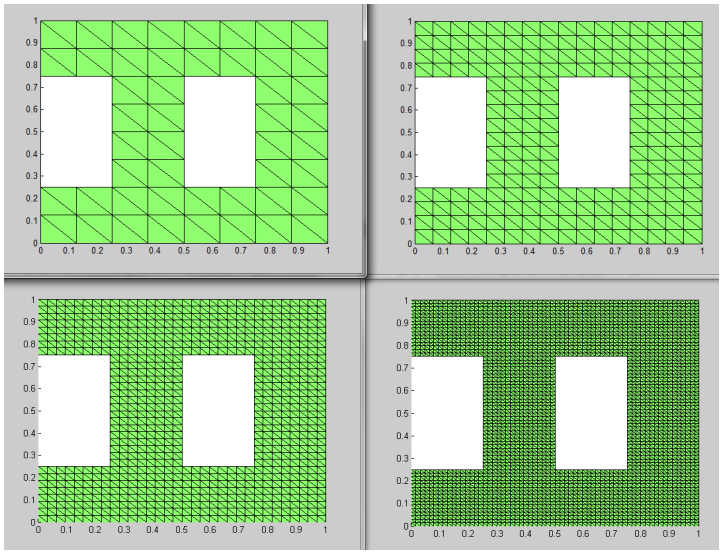


Figure 5: Top View of the four meshes

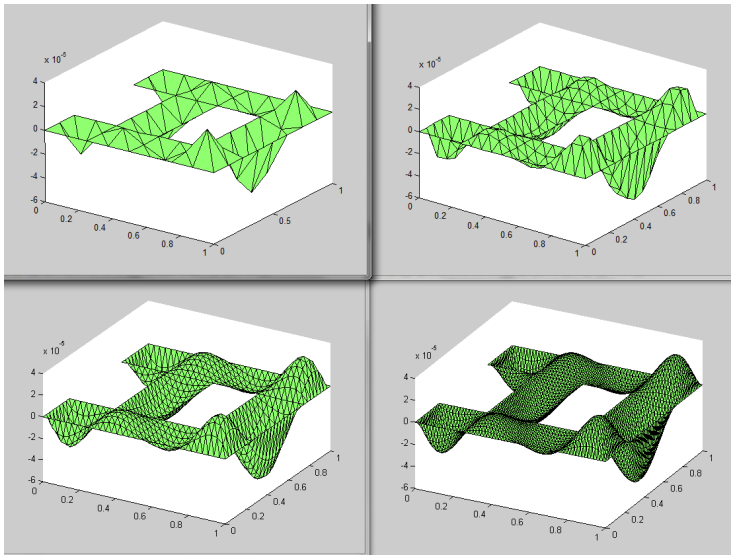


Figure 6: Tilted View of the four meshes

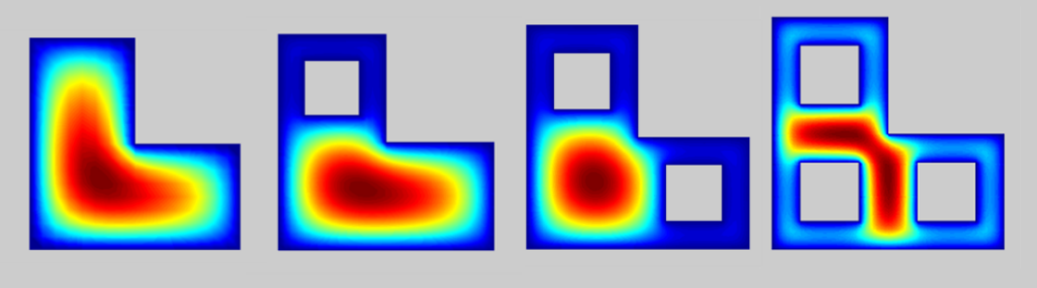
7.4 Conclusion

For examples 1 through 3, even though the domain got more complicated, the order of convergence still converged to 2. The reason why this happens is that for each of the domains, I picked a 'nice' F such that u is a polynomial and all initial conditions hold true.

8 Order of Convergence relative to the Domain

In this section, we will show that as the domain gets more complex, when $F = 1$, the Order of Convergence would not approach 2, but a number less instead.

$$\begin{aligned} -\triangle u &= 1 \text{ in } \Omega \\ u &= 0 \text{ on } \partial\Omega \end{aligned}$$



Meshes	OoC	Meshes	OoC	Meshes	OoC	Meshes	OoC
1,2,3	1.700569	1,2,3	1.725416	1,2,3	1.701380	2,3,4	1.650589
2,3,4	1.745914	2,3,4	1.663195	2,3,4	1.644536	3,4,5	1.569222

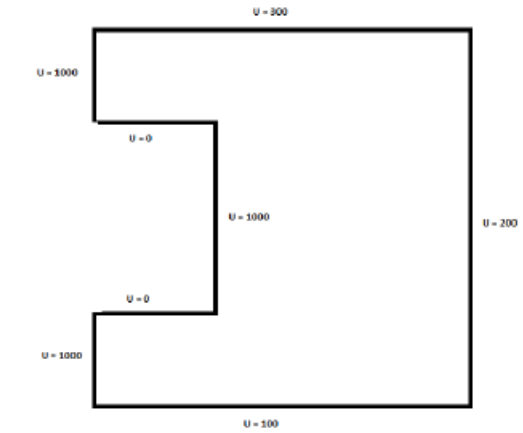
$$\text{Order of Convergence} = \log_2 \frac{||u_{h1} - u_{h2}||}{||u_{h2} - u_{h3}||}$$

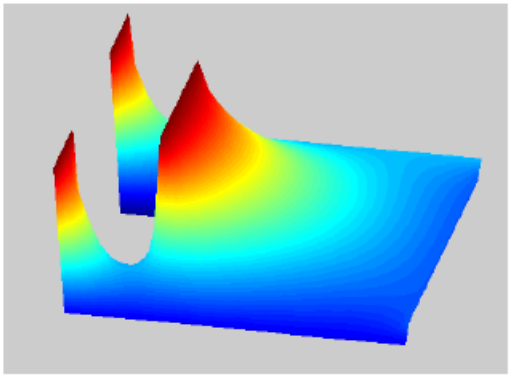
Please note that the above formula is relative to meshes 1,2,3. The formula can be modified accordingly to fit the desired meshes.

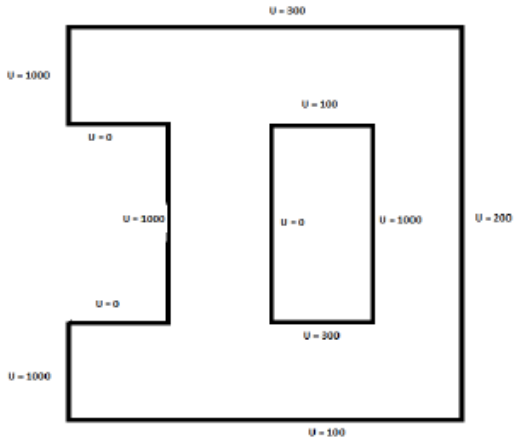
Notice that as the domain becomes more complicated, the exact solution also becomes more complicated, and this is why the Order of Convergence goes down.

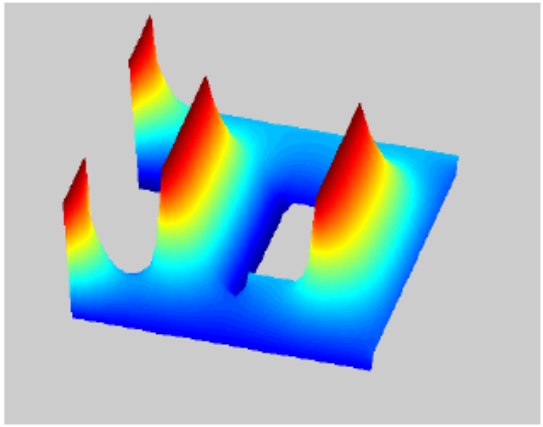
9 Exact Solutions with Nonzero boundary conditions

Example 4 shows that for the meshes, you don't have to have zero-boundary conditions. Below, the key is to the left showing the boundary conditions for the contour map on the right.









10 Shape Recognition using the Finite Element Approximation of the Poisson Equation

In this section, I started to apply techniques learned in the paper "Shape Representation and Classification Using the Poisson Equation." [Gorelick, et al: 2006]

Mainly, I focused on using their formula for Φ, $\Phi = U + U_x^2 + U_y^2$, and applying to my domains, and then learning the special properties it upholds. For this section, I decided to use the block letters of the alphabet, H, J, L, T, and U.

Φ can be used to locate the corner of the domain where u_h is changing most rapidly or slowly, or where Φ is at its maximum or minimum. And if I wanted to extract certain parts of the domain from their respective heights, the $log(\Phi)$ is calculated and graphed. From now onwards, any extracted piece of the domain is taken from the $log(\Phi)$.

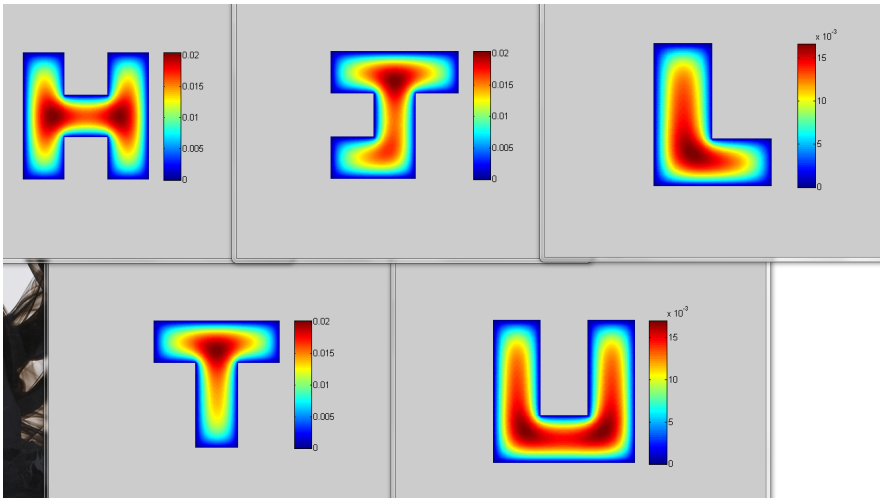


Figure 7: Contour Maps of the u_hs

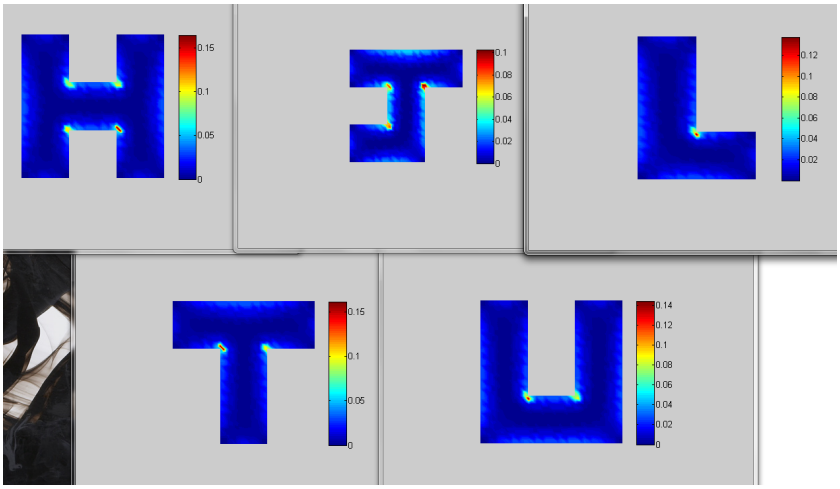


Figure 8: Contour Maps of $U_x^2 + U_y^2$

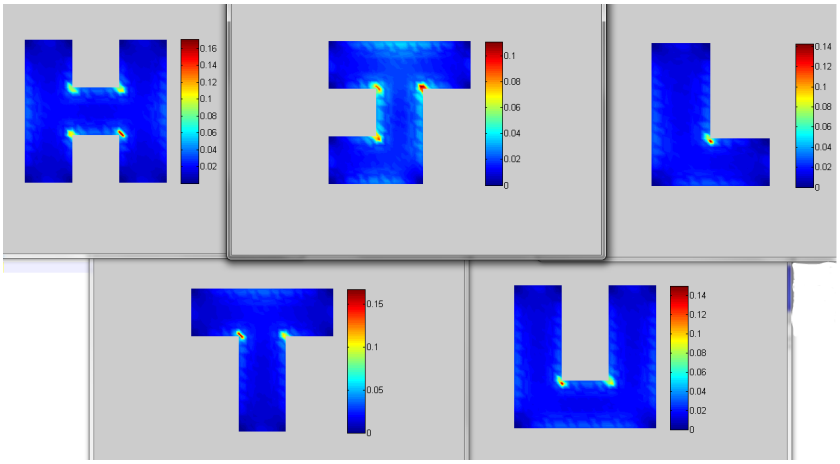


Figure 9: Contour Maps of Φ

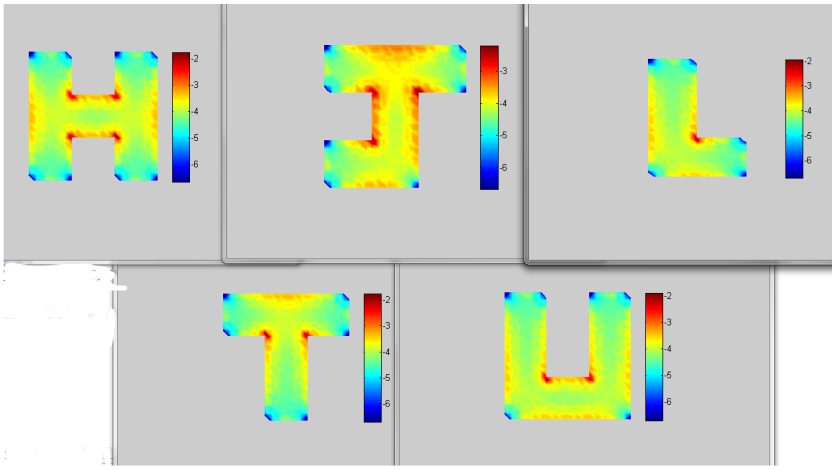


Figure 10: Contour Maps of $log(\Phi)$

11 Weighted Case of the Poisson Equation

I learned and applied the weighted case of the Poisson Equation to each of the block letters from the previous section. The weighted case is found by multiplying x to each side of the formula, and now it becomes

$$\sum_{i=1}^{N} c_i \int_{\Omega} \nabla \Phi_i \nabla \Phi_j x = \int_{\Omega} F \Phi_j x$$

This weighted Poisson equation arises when performing dimension reduction to the axisymmetric 3D Poisson equation.

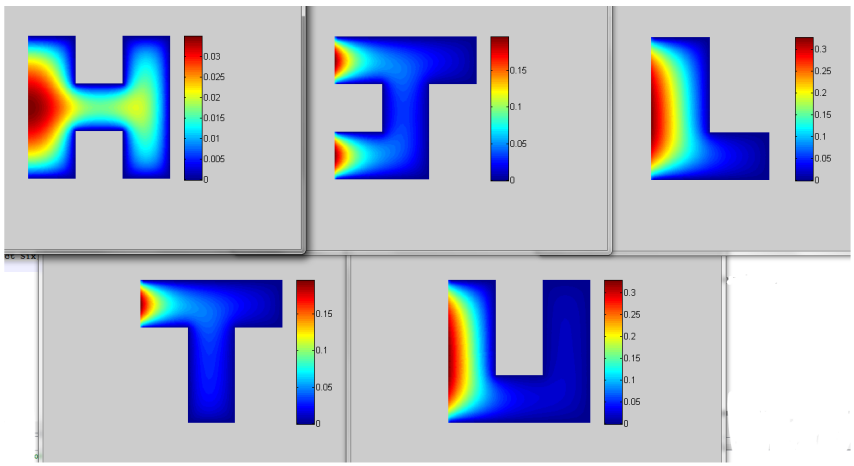


Figure 11: Contour Maps of u_h

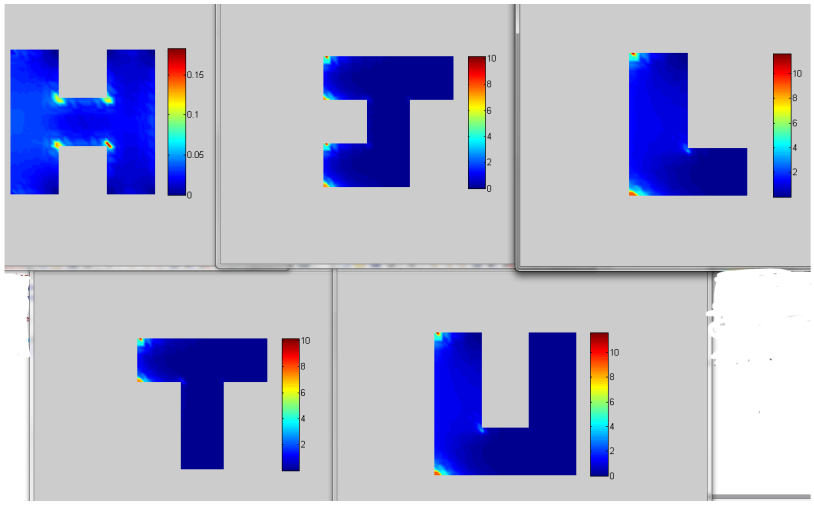


Figure 12: Contour Maps of Φ

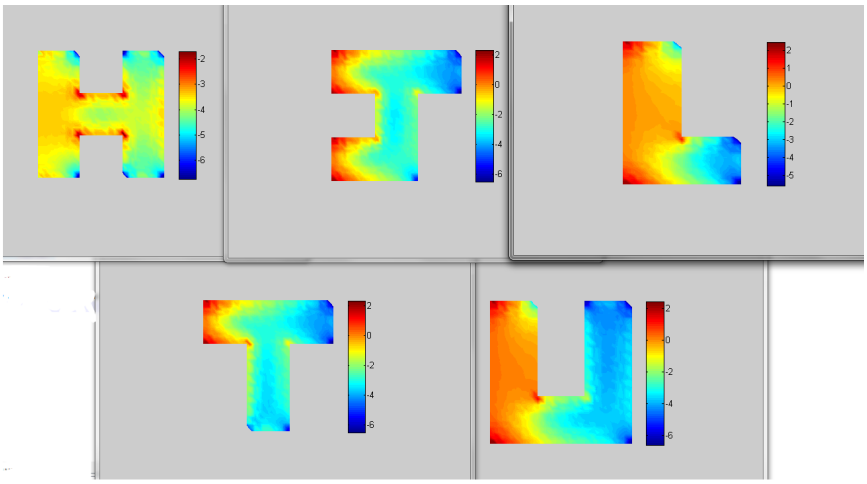


Figure 13: Contour Maps of $log(\Phi)$

12 Extraction of Parts of Domains using Φ

In this example, We now want to extract a certain percent of the graph out and plot that. In my case, I used the domains of somewhat well known cartoon characters, Goku from Dragonball Z, Sonic the Hedgehog, and The Ice King from Adventure Time. I did the normal process of graphing the contour maps of u_h, Φ, and $log(\Phi)$. I wanted to observe if exactly one specific range of Φ is ideal for identifying key features of any domain, and I found a specific range, the bottom 35%

With each of these graphs, there contained some 'noise' points. To remove these noise points, a simple removal formula was used, find the second largest containment of points, and take 10% of that. If any cluster of points are less than that value, they were removed.

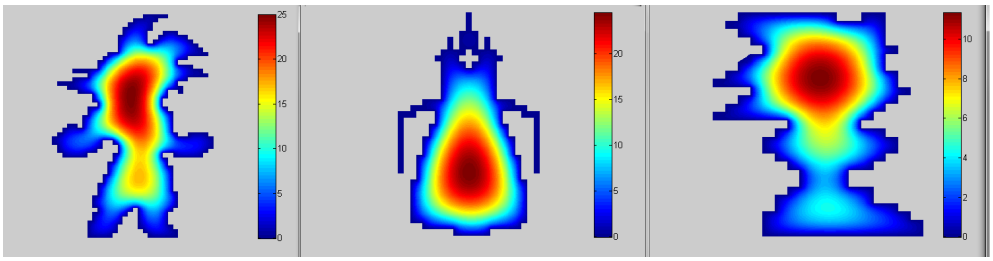


Figure 14: Contour Maps of u_h

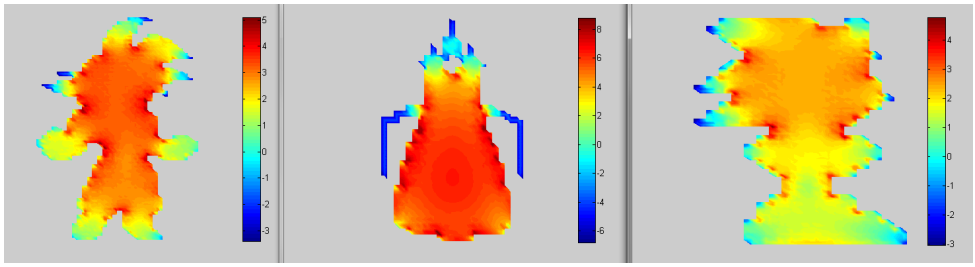


Figure 15: Contour Maps of $log(\Phi)$

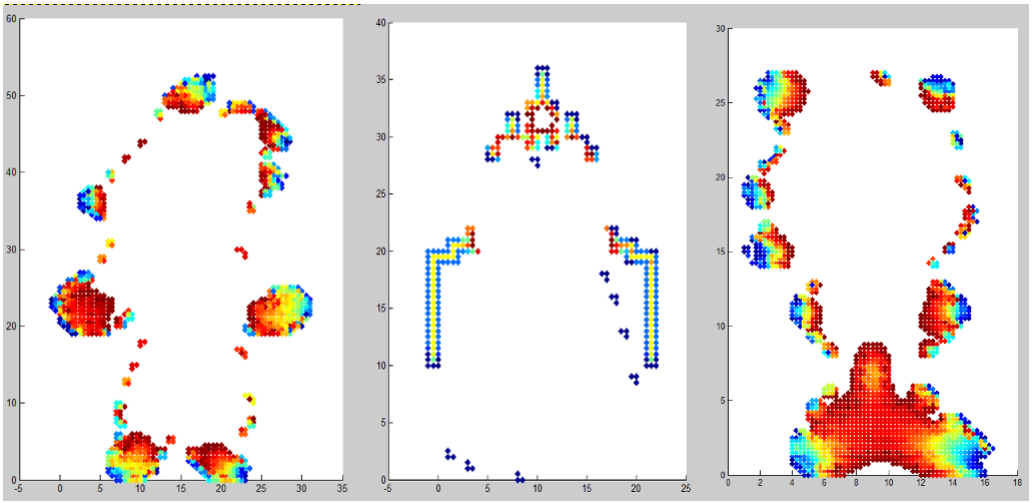


Figure 16: Contour Maps of $log(\Phi)$ with noise

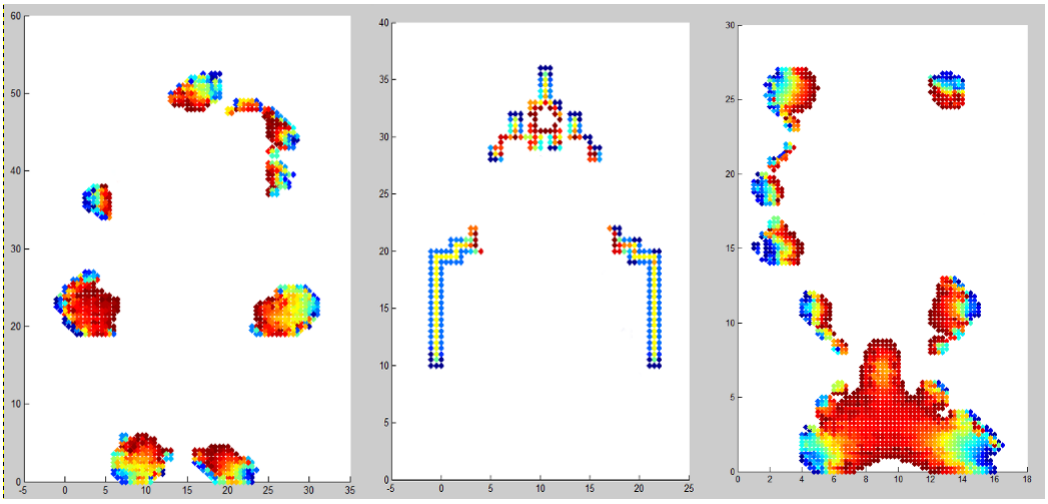


Figure 17: Contour Maps of $log(\Phi)$ without noise

The purpose of this is to show that I can take any domain and extract, then graph, the key features from it.

13 Extension to Axisymmetric 3D Domain

Now, I wanted to see if I could apply the previous extraction technique to the case of Axisymmetric 3D examples. So to start out, I have a dish or satellite dish graph shown first. I used dimension reduction to change this from 3D to 2D so it will be more computationally efficient. Then, I applied the extraction from the previous example, and rotated around the y-axis to convert the problem back to 3D.

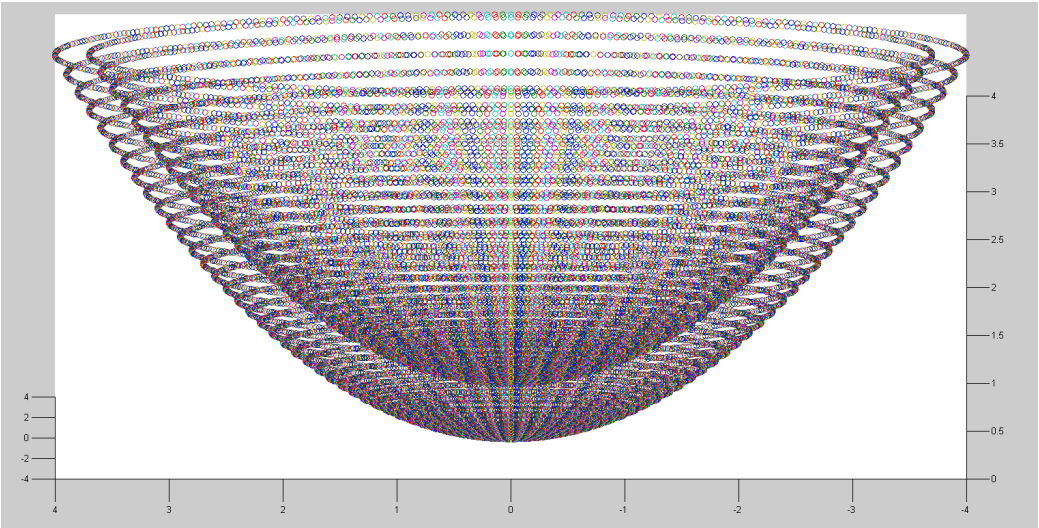


Figure 18: 3d rotation of the axisymmetric satellite dish

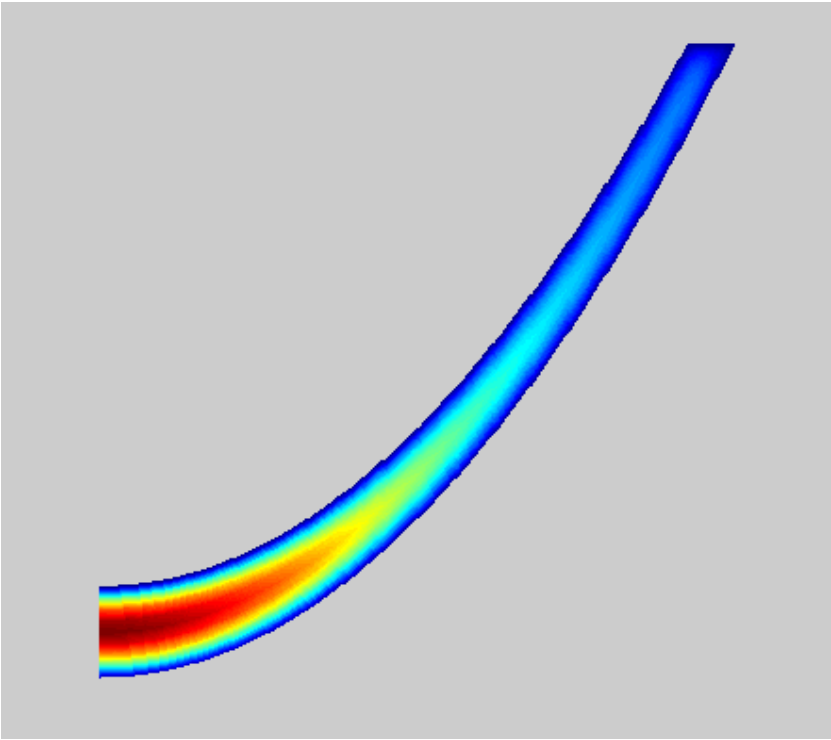


Figure 19: 2d Contour Map of the axisymmetric satellite dish

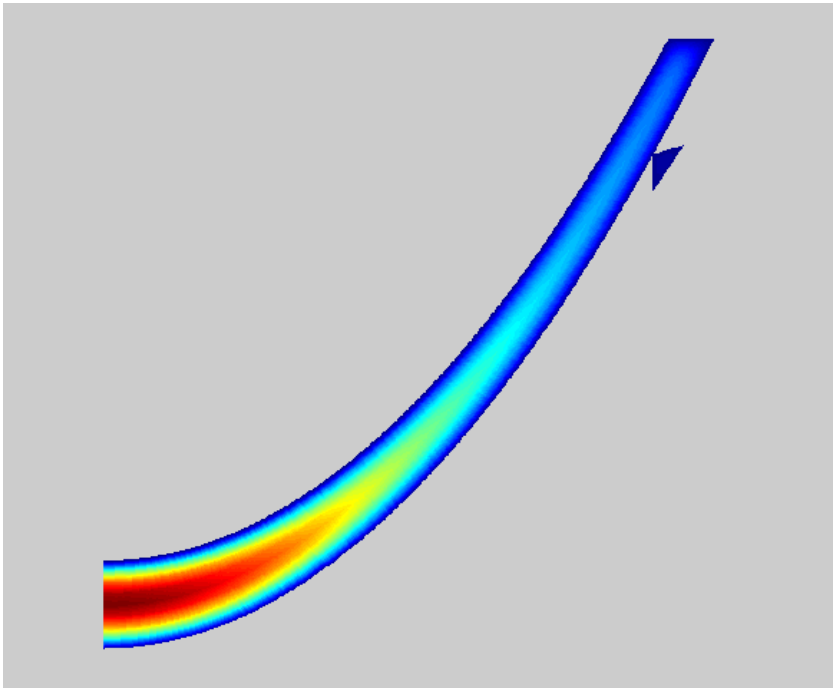


Figure 20: 2d Contour Map of the axisymmetric satellite dish with a triangle added

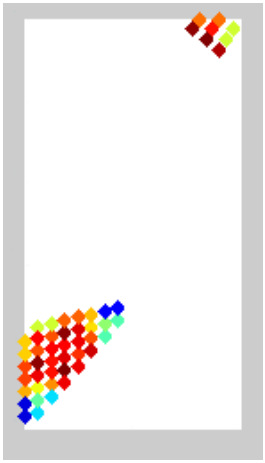


Figure 21: 0-35% of the previous figure

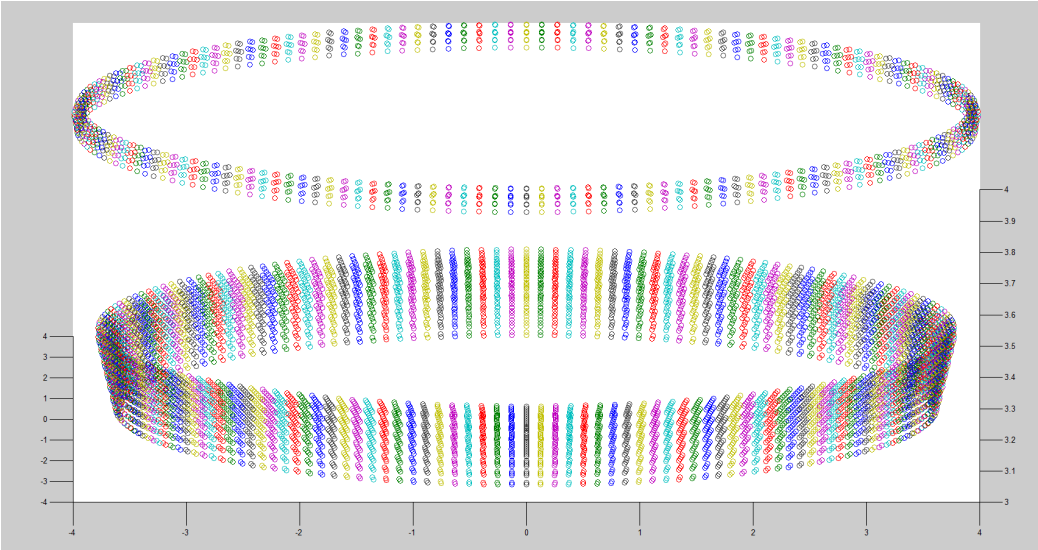


Figure 22: 3d rotation of the 0-35% of Φ


JMU