Università della Svizzera italiana

Institute of Computing CI

High-Performance Computing Lab **Institute of Computing**

Student: Dennys Mike Huber Discussed with: Alberto Finardi & Leon Ackermann

Solution for Project 6

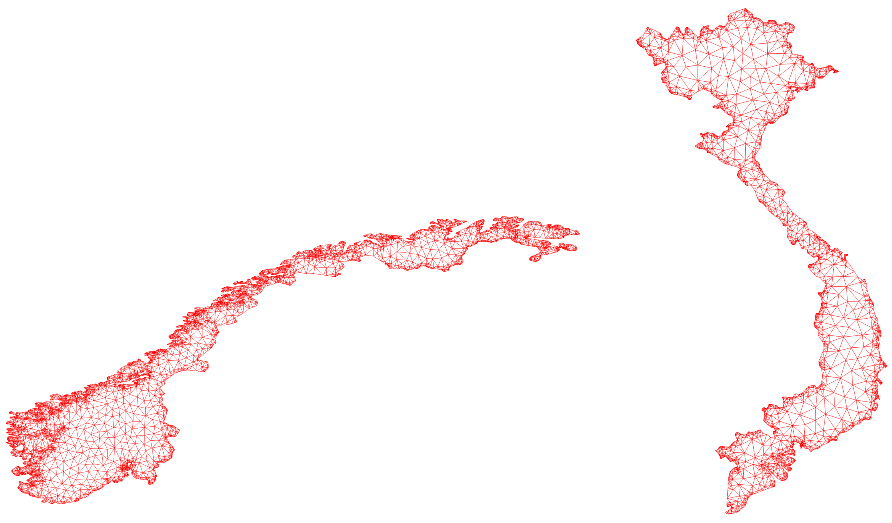
In this project, we will investigate the graph partitioning problem within the context of domain decomposition for high-performance computing (HPC) applications.

1. Construct adjacency matrices from connectivity data [10 points]

For this exercise the goal was to read the csv file of each graph and create a sparse adjacency matrix which must symmetric and sparse. To ensure the first property we applied the code seen in Listing 1, where we check if the matrix is symmetric and if not do the necessary adjustment. To ensure the second property we use the `sparse` function to create a sparse matrix.

```
W = sparse(from,to, 1, nodes, nodes);
if ~issymmetric(W)
  W = (W+W')/2;
  disp('The adjacency matrix has been symmetrized.');
end
```

Listing 1: Ensure symmetric property



(a) Norway (b) Vietnam

Figure 1: Country graphs

After creating the adjacency matrix we save it and the coordinates matrix in folder `./Datasets/Countries_Ma`
As the final step we load the matrices for Norway and Vietnam and visualize it with the provided `gplotg` function. The resulting graphs can be seen in Figure 1.

2. Implement graph partitioning algorithms [25 points]

For this exercise the task was to Implement two different graph bisection algorithms. The first algorithm can be seen in Listing 2 and is called Spectral bisection. The algorithm leverages the spectral properties of the Laplacian Matrix, which can be computed by $L = D - A$, where D is the degree matrix and A is the adjacency matrix. Subsequently, we use this Laplacian matrix to compute its eigen values and eigen vectors. In Matlab we can use the `eigs` function to that for sparse matrices. More precisely we are interested in the second smallest eigenvector, also commonly referred to as Fiedler vector. Using the median of the Fiedler vector we then assign the vertices to one of the partitions depending on if the corresponding entry in the Fiedler vector is smaller or bigger than the median.

```matlab
function [part1,part2, eig_vec] = bisection_spectral(A,xy,picture)

  % 1. Construct the Laplacian.
  D = diag(sum(A, 2));
  L = D - A;
  % 2. Calculate its eigensdecomposition.
  [eig_vec,~] = eigs(L,3,1e-10);
  % 3. Label the vertices with the components of the Fiedler vector.
  fiedler_vec = eig_vec(:,2);
  % 4. Partition them around their median value, or 0.
  fiedler_med = median(fiedler_vec);
  n = size(A,1);
  map = zeros(n,1);
  map(fiedler_vec <= fiedler_med) = 0;
  map(fiedler_vec > fiedler_med) = 1;
  [part1, part2] = other(map);

  if picture == 1
    gplotpart(A,xy,part1);
    title('Spectral bisection (actual) using the Fiedler Eigenvector');
  end
end
```

Listing 2: Spectral graph bisection implementation

The second algorithm that we Implemented was inertial bisection 3, which leverages geometric properties to subdivide the graph into two subpartitions. Inertial bisection does this by computing the center mass in both x and y. Then the matrix M is constructed to reflect the distribution of points relative to the center of mass. This Matrix M is then used to compute its own smallest normalized eigen vector, which represents the direction of minimum variance. This eigen vector is used to define a line that passes through the center of mass. This line is then used to partition the vertices into two disjointed groups.

```matlab
function [part1,part2] = bisection_inertial(A,xy,picture)
  % Steps
  % 1. Calculate the center of mass.
  xy_mean = mean(xy, 1);
  x_mean = xy_mean(1);
  y_mean = xy_mean(2);

  % 2. Construct the matrix M.
  x_shift = xy(:,1) - x_mean;
  y_shift = xy(:,2) - y_mean;
  Sxx = sum(x_shift.^2);
  Syy = sum(y_shift.^2);
  Sxy = sum(x_shift.*y_shift);
  M = [Syy, Sxy; Sxy, Sxx];

  % 3. Calculate the smallest eigenvector of M.
  [eig_vec,~] = eigs(M,1,1e-10);

  % 4. Find the line L on which the center of mass lies.
  eig_vec = eig_vec/norm(eig_vec);

  % 5. Partition the points around the line L.
  [part1, part2] = partition(xy, eig_vec);

  if picture == 1
    gplotpart(A,xy,part1);
    title('Inertial bisection (actual) using the Fiedler Eigenvector');
  end
end
```

Listing 3: Inertial graph bisection implementation

As a final step we compare the newly implemented algorithms to the Coordinate bisection and Metis bisection methods using a variety of different mashes. The goal is to figure out, which algorithms creates the lowest number of edge-cuts to create the two partitions. The results in Table 1 clearly show that there is not a clear algorithm that always creates the lowest number of edge-cuts and the choice of bisection algorithm has to be made on an individual problem basis.

Mesh	Coordinate	Metis 5.0.2	Spectral	Inertial
mesh1e1	18	18	18	20
mesh2e1	37	34	36	47
mesh3e1	19	20	18	19
mesh3em5	19	20	24	19
airfoil1	94	93	132	93
netz4504_dual	25	19	23	27
stufe	16	17	16	16
3elt	172	96	117	257
barth4	206	111	127	208
ukerbe1	32	36	32	28
crack	353	220	233	384

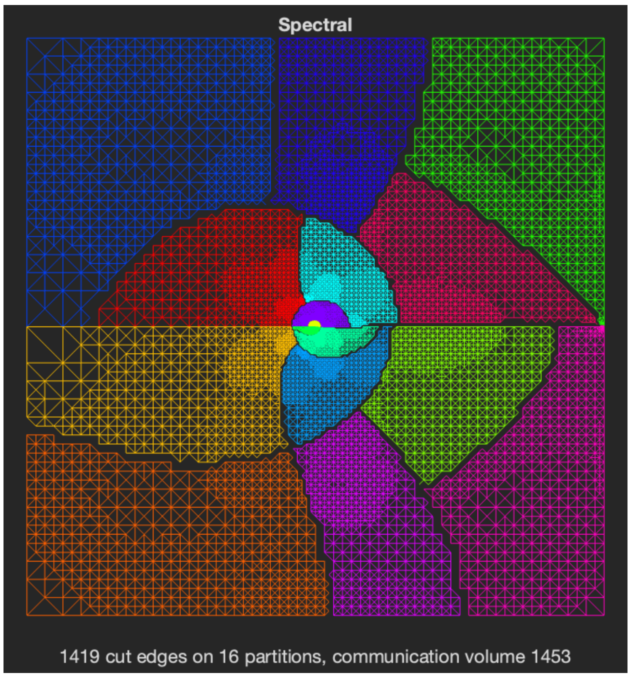
Table 1: Bisection Results

3. Recursively bisecting meshes [15 points]

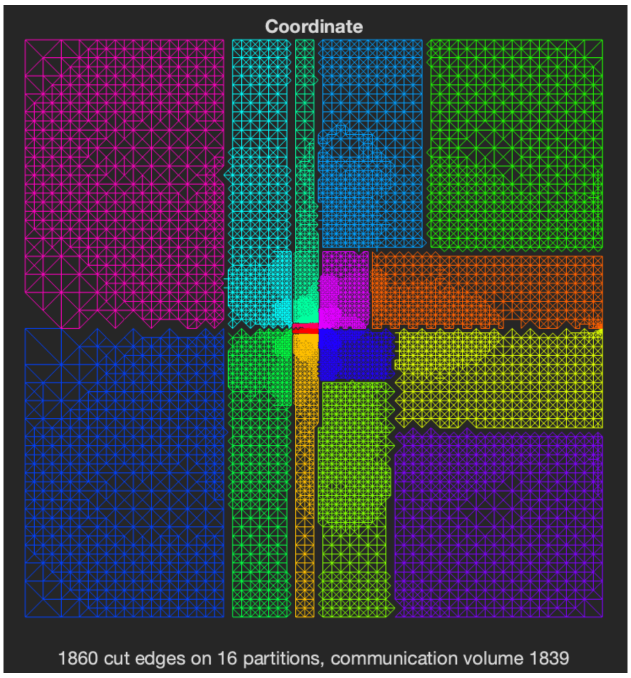
In this exercise we recursively partition 2D graphs derived by structural engineering matrices provided by Nasa. For that we use the provided function `Recursion`, which takes a bisection algorithm and graph as arguments and then partitions the graph into 2^l subpartitions. In our case we chose l to be 3 and 4 to create 8 and 16 partitions respectively. The resulting edge-cuts for both partition sizes can be found in table 2. Based on said table we can observe that Metis consistently produces the smallest edge cut for both 8 and 16 partitions, outperforming the other mesh cut methods. Furthermore an example for the visual result of different algorithms for the mesh `crack` using 16 partitions can be seen in Figure 2.

Mesh	Edge Cut (8 Partitions)				Edge Cut (16 Partitions)			
	Spectral	Metis 5.0.2	Coordinate	Inertial	Spectral	Metis 5.0.2	Coordinate	Inertial
airfoil1	397	318	516	670	629	561	819	1081
netz4504_dual	112	96	127	165	183	159	198	271
stufe	129	108	123	320	246	193	227	606
3elt	469	418	733	814	752	699	1168	1230
barth4	549	470	875	977	835	743	1306	1492
ukerbe1	781	147	225	340	888	245	374	499
crack	883	808	1343	1351	1419	1275	1860	1884

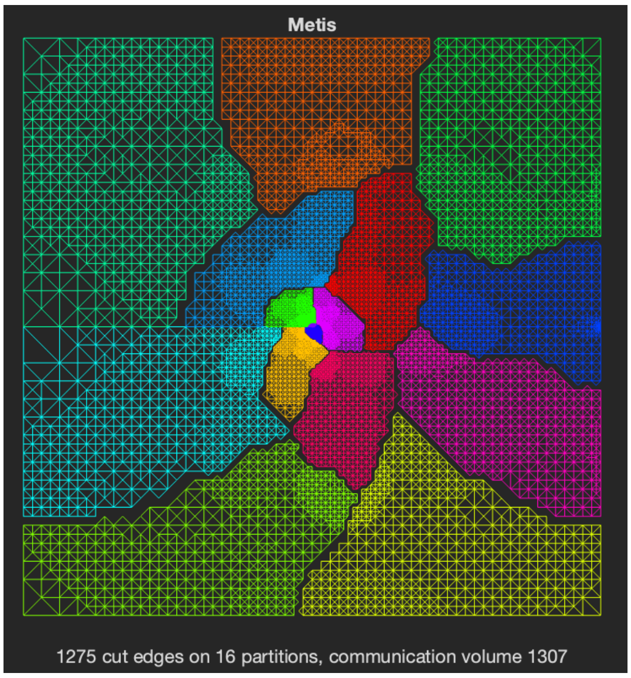
Table 2: Edge cut results for recursive bi-partitioning using 8 and 16 partitions.



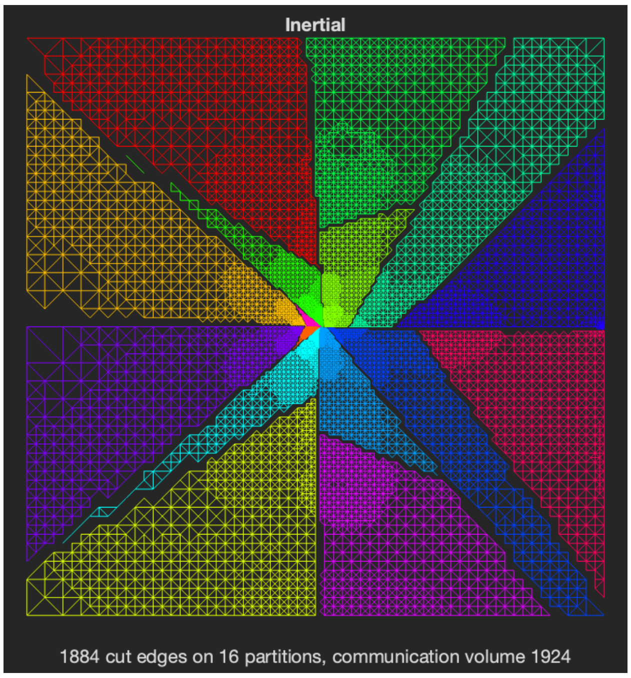
(a) Spectral bisection



(b) Coordinates bisection



(c) Metis bisection



(d) Inertial Bisection

Figure 2: Recursive bisecting of mesh "crack" into 16 partitions with different algorithms.

4. Comparing recursive bisection to direct k-way partitioning [10 points]

In this exercise the goal is to compare and analyze the performance of recursive bisection and direct k-way partitioning by looking at the edge cut result. Both methods are using Metis and the goal is to partition the graphs into 16 (Table 3) and 32 (Table 4) partitions. Comparing the two methods we can clearly see that that direct k-way partitioning consistently outperforms the recursive bisection by creating a smaller edge cut. This is the case because recursive bisection has no global information when doing partitioning decisions, k-way on the other hand overcomes this limitation by coarsening the graph first and the then refining the partitioning. Knowing this coarsening and refinement steps it was anticipated that k-way partitioning would outperform recursive bisection. Another observation that can be made is that the difference in edge cut between the methods increases when comparing 16 to 32 partitions. Indicating that k-way multiway partitioning also scales better in terms of number of partitions.

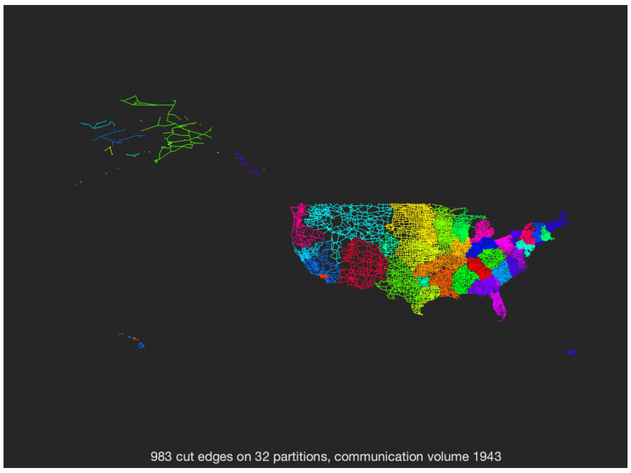
Partitions	Luxemburg	USRoads	Greece	Switzerland	Vietnam	Norway	Russia
16	191	585	318	685	270	271	572
32	317	983	500	1067	445	509	941

Table 3: Cut edges for recursive bisection.

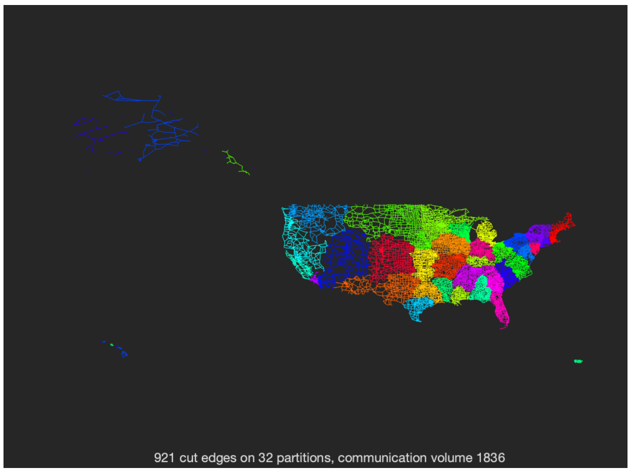
Partitions	Luxemburg	USRoads	Greece	Switzerland	Vietnam	Norway	Russia
16	185	545	301	665	231	241	551
32	304	921	486	1013	418	436	931

Table 4: Cut edges for direct multiway partitioning in Metis 5.0.2.

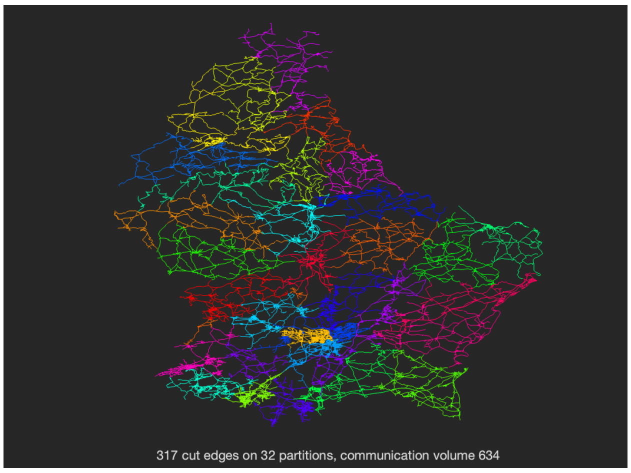
Figure 3 showcases how various examples of graphs are partitioned differently when using recursive or k-way partitioning.



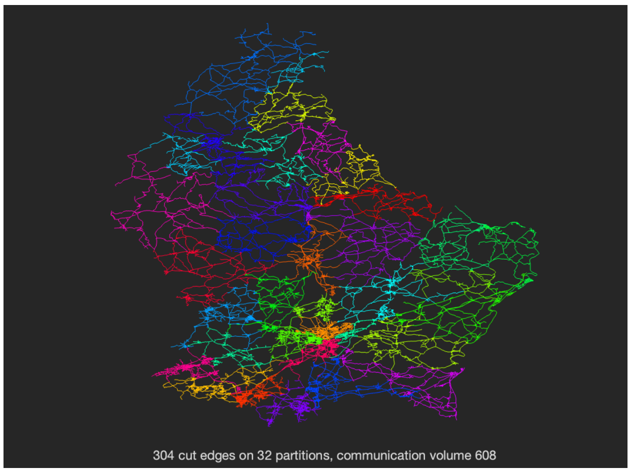
(a) USA with recursive bisection



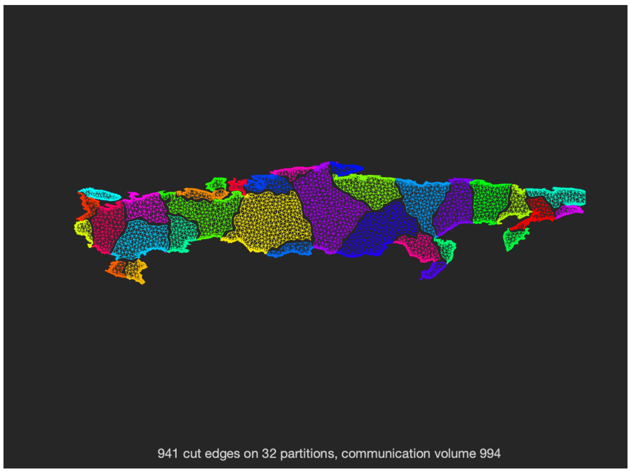
(b) USA with direct multiway partitioning



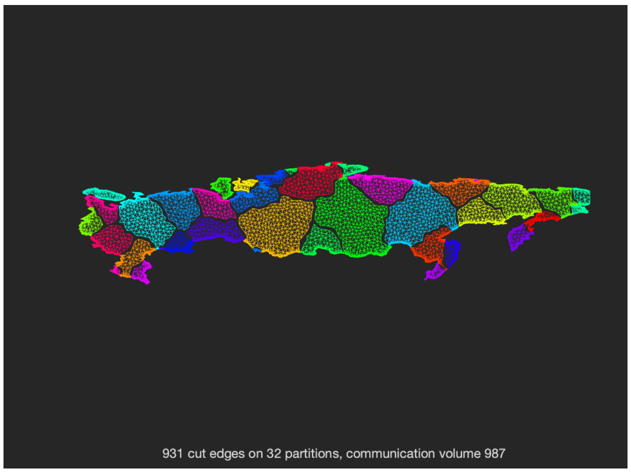
(c) Luxembourg with recursive bisection



(d) Luxembourg with direct multiway partitioning



(e) Russia with recursive bisection

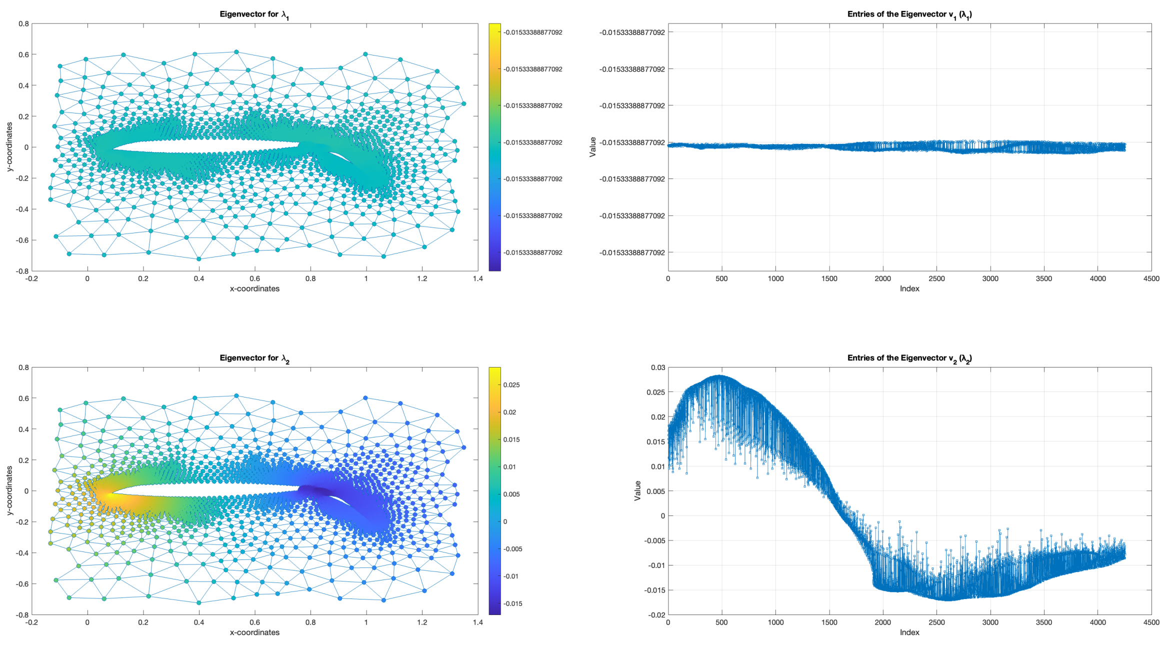


(f) Russia with direct multiway partitioning

Figure 3: Partitioning results for the graphs of USA, Luxemburg, and Russia for 32 partitions

5. Utilizing graph eigenvectors [25 points]

The entries of the eigenvector associated with the first (λ_1) and second (λ_2) smallest of the Laplacian L are plotted in Figure 4. The behaviour of the λ_1's eigenvector v_1 is expected to have only 1's as its elements due to the property of the L. In our case the entries are not 1 but the normalized vector which is $\frac{v_1}{\sqrt{n}}$, where n is the length of the eigenvector [1]. The Fiedler vector v_2 of the second smallest eigen value reflects the structure of the graph. Vertices are strongly connected will have similar eigenvector values, while vertices will show more distinct values, when they are further apart. When looking at plot for v_2 in the Subfigure 4b you can see that the vector shows a smooth transition from positive to negative indicating that they are transitioning from one partition to another. Subfigure 4a then shows this grouping using coloring of the vertices.



(a) The vertices colored according to their corresponding eigenvector element

(b) The entries of the eigenvector plotted against their indices.

Figure 4: Entries of the eigenvector associated with the first (λ_1) and second (λ_2) smallest eigenvector.

As the next step we project each solution on the coordinate system space for the graph mesh3e1, barth4, 3elt and crack The result of this can be seen in Figure 5. Looking at the shape of the values of the fiedler vector we can see that the for 3elt and barth4, they are concentrated in a vertical transition. While for the other two, they have more of a planar structure, creating a clear horizontal separation. Furthermore all graphs appear to have a smooth transition, seen by the transition of color from dark blue to red.

As the final step show the graph with its partitions and edge cut by not using the coordinate space, but by using the eigenvectors values of the graph Laplacian. In Figure 6 one can see the partitioning results in two ways. On the top using the eigenvector values and on the bottom the Euclidean coordinates. For the barth4 mesh for example in Subfigure 6b we are able to visualize the relation between nodes a lot more effectively using the eigenvector values as our coordinates, in comparison to the cartesian version, where the groups are barely visible.

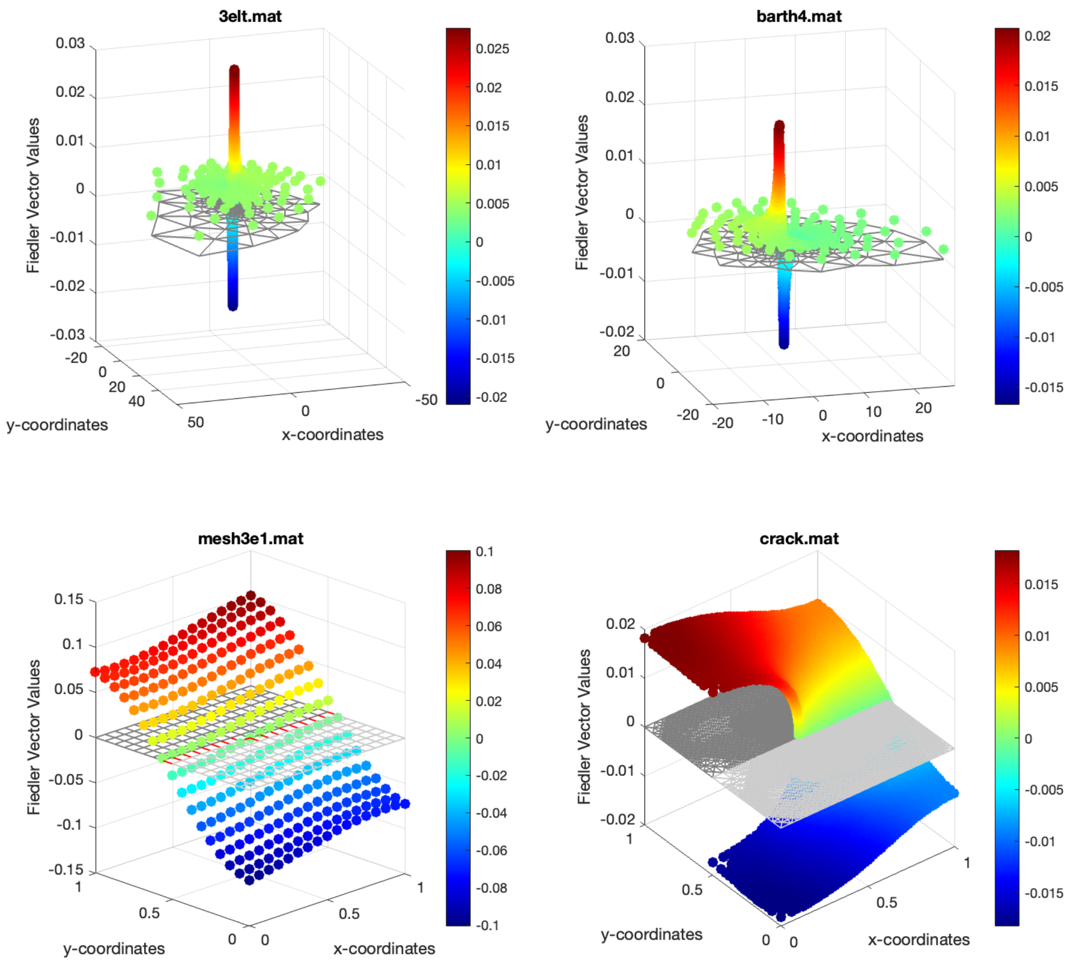
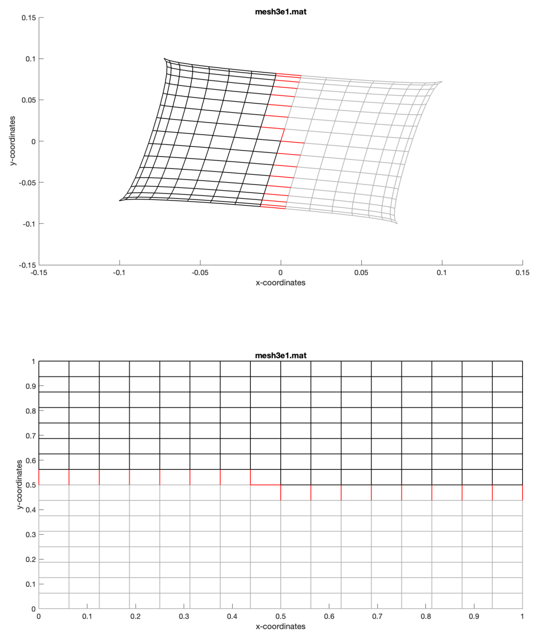
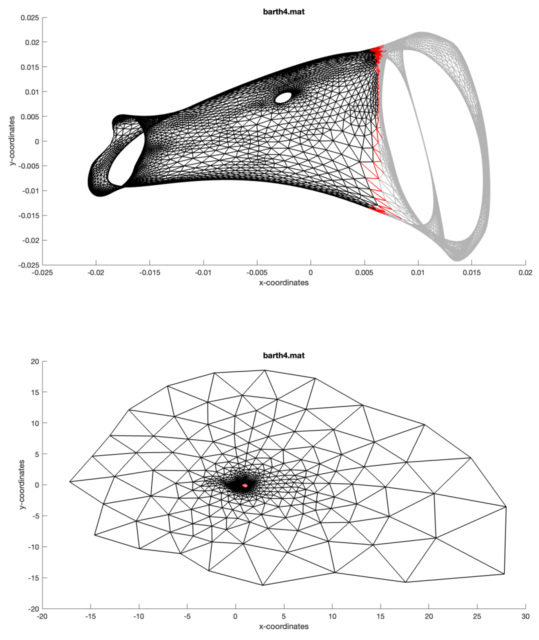


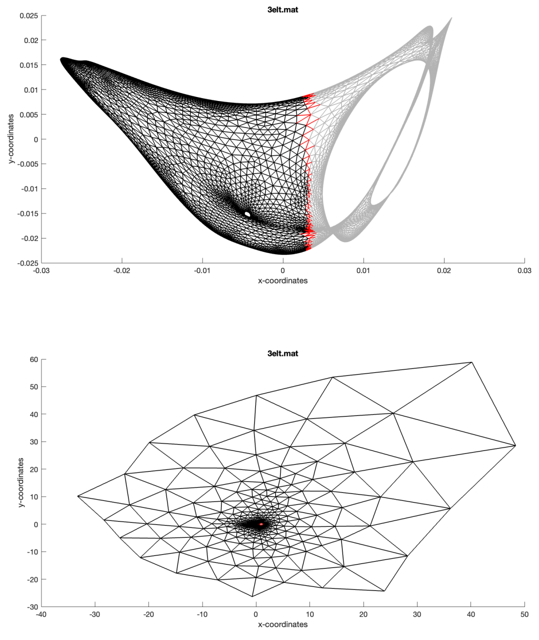
Figure 5: Entries of the eigenvector associated with the second smallest eigenvalue λ_2 of the Graph Laplacian matrix L. The two partitions are depicted in dark gray and light gray, while the cut edges is red respectively. The z-axis represents the value of the entries of the eigenvector.



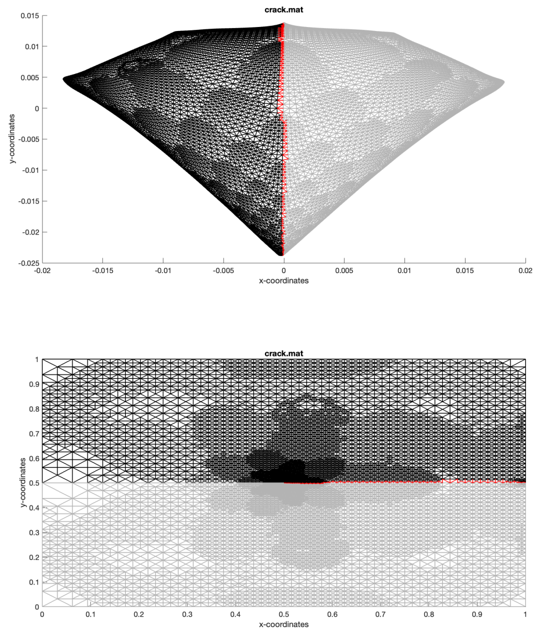
(a) mesh3e1



(b) barth4



(c) 3elt



(d) crack

Figure 6: Spectral bi-partitioning results using the eigenvectors to supply coordinates (top) and using cartesian coordinates (bottom).

References

[1] *The Smallest Eigenvalues of a Graph Laplacian*. URL: `http://blog.shriphani.com/2015/04/06/the-smallest-eigenvalues-of-a-graph-laplacian/` (visited on 12/16/2024).