Graph Partitioning using Eigenvectors

10 points

In this problem, we consider the Fiedler algorithm, which partitions graphs by finding eigenvectors of a matrix associated with the graph. Your main task will be to implement Lanczos to find a particular eigenvector.

Given a graph G, we can partition elements into groups by calculating the Fiedler vector of its Laplacian matrix $\boldsymbol{L}$:

- if $i \neq j$,
    - $l_{ij} = -1$ if the i^{th} vertex of G is connected to the j^{th} vertex
    - $l_{ij} = 0$ otherwise
- $l_{ii} =$ number of connections from vertex i (degree of vertex i)

The row-sums of the symmetric matrix $\boldsymbol{L}$ are zero, so it has a 1-dimensional null-space, which corresponds to its smallest eigenvalue, zero. The Fiedler vector of a graph G is the eigenvector with the second smallest eigenvalue. From this vector, the graph can be partitioned by taking vertices corresponding to eigenvector elements with an entry less than the median in one partition and placing the remaining vertices in the other partition. As an intuition for why the Fiedler vector provides a partitioning with a small cut, we can consder a graph that has two disjoint partitions. The Fiedler eigenvector then contains ones for all vertices in one partition, and zeros for all vertices in the other partition, with an eigenvalue of zero (in this case the null space of $\boldsymbol{L}$ is 2-dimensional).

Since we are interested in an extremal eigenvalue (second smallest) of a symmetric matrix, we will employ the Lanczos algorithm. Since we are not interested in the trivial 1-dimensional null-space of $\boldsymbol{L}$, we will orthogonalize the starting Krylov vector to the vector of ones.

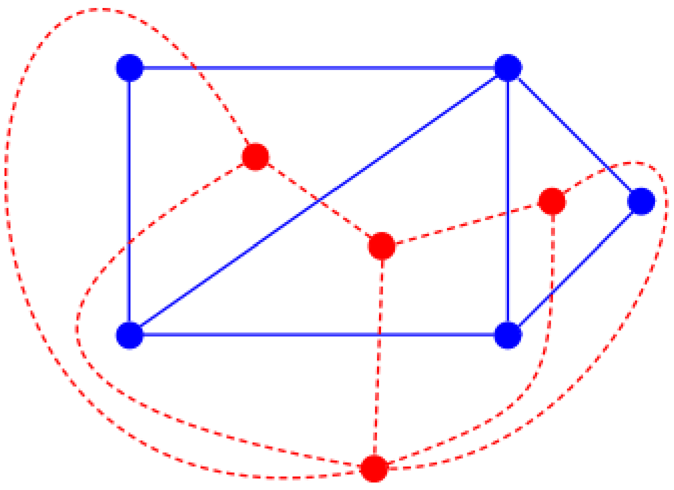
Application of Graph Partitioning for Mesh Splitting (click to view)

Computational models often requires spatial discretization. Normally this is done by using a mesh. A mesh consists of vertices and edges that connect the vertices to form a contiguous union of non-overlapping triangles (or potentially squares, hexahedrons, tetrahedrons, etc. depending on preference). Naturally, a mesh can be represented as a graph.

For very large problems, we can parallelize mesh computation by splitting the mesh and processing each part concurrently on separate processors. Ideally, the mesh should be split so that each processor has an equal amount of work and the communication between processors is minimal. Because our mesh can be represented as a graph, we can think of this space decomposition as a graph partitioning problem.

From the Laplacian matrix, one can describe the connectivity and adjacency of vertices in a graph G. However, we are more interested in the adjacency of elements rather than vertices because we want to avoid splitting a single element (triangle) into two different partitions. We can accomplish this by performing the partitioning on the "dual graph" (https://en.wikipedia.org/wiki/Dual_graph).

In the figure shown below the original graph is in blue and the graph shown in red is the dual graph:



(Source: Bneder2k14, Wikimedia Commons, CC0).

Code is provided for you for the following tasks:

- Read in a mesh from a file - `readmesh` function.
- Generate the Laplacian of the dual graph of G - `mesh2dualgraph` function.
- Fiedler's algorithm which uses the Lanczos iteration to calculate the graph partitions - `fiedler` function.
- Plotting original and parititioned meshes - `plotmesh` function.

You can read the docstrings in the above functions in the problem set-up code and starter code sections to find out more about what they do. These functions are callable from the submission box. If you decide to test your code locally, you can download the mesh files below. Once downloaded, call the `readmesh` function with the path to the mesh as an argument to read it.

- mesh.1 (/course/cs450-s19/file-version/dd9a1d98076c43c7f314f5733421d849d1acad38/data/mesh.1)
- mesh.2 (/course/cs450-s19/file-version/dd9a1d98076c43c7f314f5733421d849d1acad38/data/mesh.2)

Your task is to implement the Lanczos algorithm in a function called `lanczos` and to extract the vector with the smallest Ritz value in the function `fiedler_ritz`. The `lanczos` function takes three arguments:

- `L` : Laplacian $\boldsymbol{L}$ of the dual graph (`scipy.sparse` matrix of shape: $n \times n$)
- `x0` : Starting vector for Lanczos iteration (`numpy.array` of length n)
- `k` : number of Lanczos iterations to be performed.

Your completed `lanczos` function should implement the Lanczos algorithm after removing the components of the $\mathbf{1}$ vector (vector of ones) from the initial guess `x0` via orthogonalization. It should return two matrices:

- `Q` : Contains first `k` orthogonal Lanczos vectors (`numpy.array` of shape: $n \times k$)
- `T` : Tridiagonal matrix having eigenvalues and eigenvectors similar to the original Laplacian matrix (`L`) provided. (`numpy.array` of shape: $(k \times k)$)

These matrices are then given as arguments to the `fiedler_ritz` function, which should return an approximation to the Fiedler eigenvector:

- `fiedlerVec` : vector of dimension n corresponding to the Ritz vector (i.e., approximate eigenvector) with the smallest Ritz value

Problem set-up code (click to view)

Starter code (click to view)

Answer*

```
    w,v = la.eig(T)
    #Implement the body of this function
    return Q, T

def fiedler_ritz(Q,T):
    #Implement the body of this function
    w,v = la.eigh(T)
    #idx = -1
    #smallv = 1000000
    #for i in range(len(w)):
    #    if w[i]<= smallv:
    #        smallv = w[i]
    #        idx = i
    idx = np.argsort(w)
    fiedlerVec = Q@v[:,idx[0]]
    return fiedlerVec

def fiedler(G, k):
    """
    Calculate the fiedler vector of the graph Laplacian matrix
    'G' using 'k' niter of Lanczos algorithm.
    """
```

Press F9 to toggle full-screen mode. Set editor mode in user profile (/profile/).

Your answer is correct.

Here is some feedback on your code:

- 'partitionVec' looks good
- Results look good. Good job!
- Execution time: 3.6 s -- Time limit: 15.0 s

Your code ran on relate-06.cs.illinois.edu.

Your code produced the following plots:

Figure1

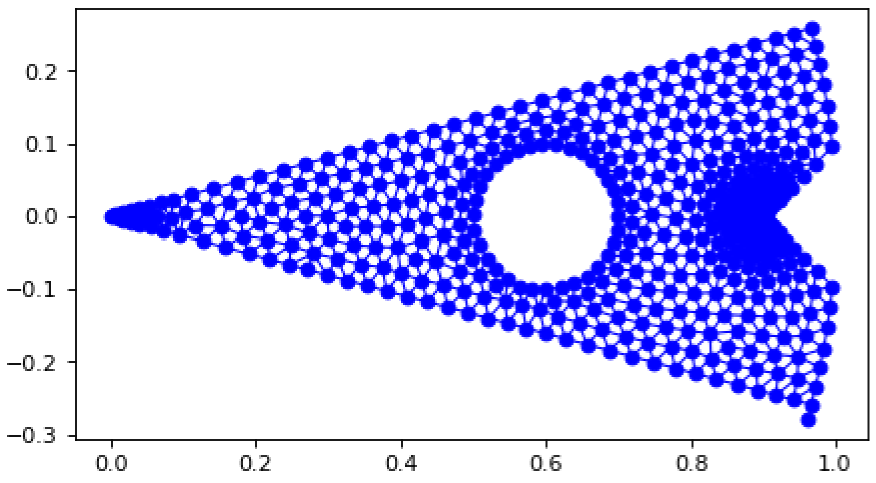
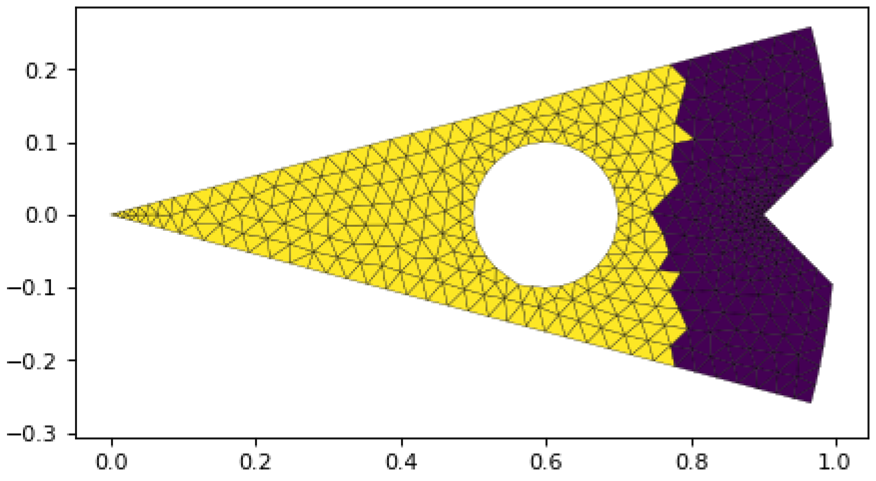


Figure2



The following code is a valid answer:

```python
import numpy as np

def lanczos(L, x0, k):
    n, m = L.shape
    s = x0.shape[0]

    assert (n == m), "Matrix should be square !!"
    assert (n == s), "Initial vector has wrong shape !!"

    Q = np.zeros((n, k))

    ones = np.ones(n)
    ones = ones/np.linalg.norm(ones)

    x0 = x0 - ones * np.inner(ones,x0)

    ## Initialize q0 to zero vector and beta_0 to 0
    q0 = np.zeros_like(x0)

    ## Initialize q1 to normalized initial vector
    q1 = np.divide(x0, np.linalg.norm(x0))

    ## Lists to keep track of main diagonal and super-sub diagonal
    alpha = np.zeros((k,))
    beta = np.zeros((k + 1,))

    ## Loop through the size of the matrix
    for l in range(k):
        Q[:, l] = q1[:]
        ## Matrix-vector product
        u = L.dot(q1)

        ## Calculate alpha_l
        alpha[l] = np.dot(q1.T, u)

        ## Update u
        u = u - beta[l] * q0 - alpha[l] * q1

        ## Calculate beta_{l+1}
        beta[l + 1]  = np.linalg.norm(u)

        ## Set q_{l+1}
        q0[:] = q1[:]
        q1 = np.divide(u, beta[l + 1])

    T = np.diag(alpha, 0) + np.diag(beta[1:-1], -1) + np.diag(beta[1:-1], 1)

    return Q, T

def fiedler_ritz(Q, T):
    eVal, eVec = np.linalg.eig(T)
    idx = eVal.argsort()
    eVal = eVal[idx]
    eVec = eVec[:, idx]
    fiedlerVec = np.dot(Q, eVec[:, 0])
    return fiedlerVec
```

```
def fiedler(G, k):
    """
    Calculate the fiedler vector of the graph Laplacian matrix
    'G' using 'k' iterations of Lanczos algorithm.
    """
    n, m = G.shape

    assert (n == m), "Matrix should be square !!"

    x0 = np.linspace(1, n, num = n)

    ## You should complete this Lanczos function
    Q, T = lanczos(G, x0, k)

    ## You should complete this Fiedler vector computation function
    fiedlerVec = fiedler_ritz(Q,T)

    partitionVec = np.zeros_like(fiedlerVec)
    mfeidler = np.ma.median(fiedlerVec)

    for i in range(n):
        if (fiedlerVec[i] >= mfeidler):
            partitionVec[i] = 1
        else:
            partitionVec[i] = -1

    return partitionVec

points, triangles = readmesh("mesh.1")
plotmesh(points, triangles)
G = mesh2dualgraph(triangles)
partitionVec = fiedler(G, 150)
```