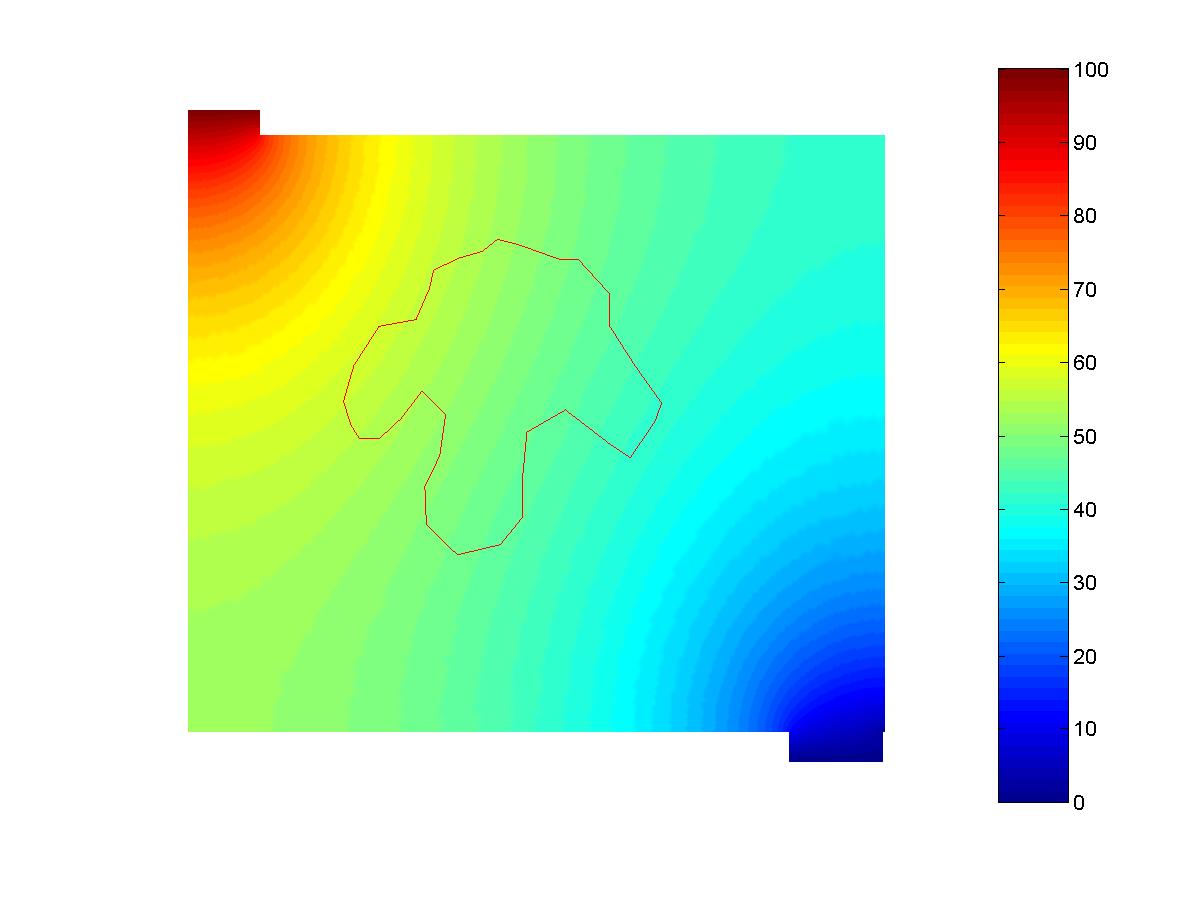
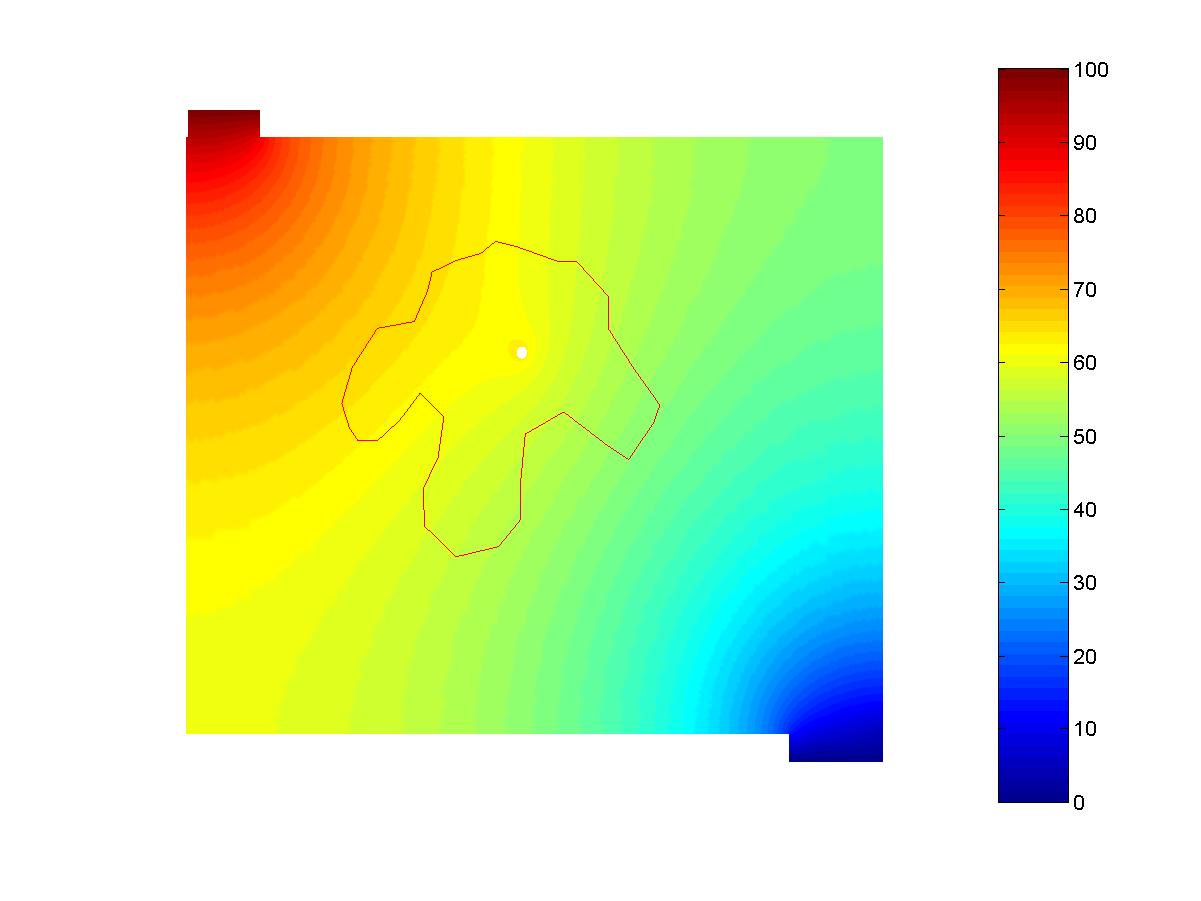
**BME 7310 Computational Laboratory #2**

Due: 9/15/2023

**Problem 1: Concepts in Divergence.**

ab

**Electro-potential distributions (a) Scenario #1, (b) Scenario #2.**

Often electrostatic potential theory is used to represent the distribution **of current within the brain**. In the two scenarios above, a very basic distribution of potential (i.e. voltage) is illustrated. Laplace’s equation was used for each problem. It highlights the flow of current from the top left to the bottom right of the images. Within the domain, an arbitrary contour is shown. The image to the right shows an additional structure which influences the domain.

You have been provided 2 files for each scenario above. The two files for each scenario are: POINT\_LIST\_#.DAT, and LINE\_LIST\_#.DAT .... where the # is associated with each scenario.

POINT\_LIST\_#.DAT has the format: [point index] [x position] [y position] [**Jx**] [**Jy**]

LINE\_LIST\_#.DAT has the format: [**contr. line index**] [**contr. node 1**] [**contr. node 2**]

For reference, the contour nodes are arranged **such that for each contour line segment**, **it is arranged as traversing the boundary in a counter-clockwise manner.**

a. Your task is to **calculate the following line integral:**



and **report the values for each of the scenarios above**. Be sure to submit any MATLAB code used to achieve your results.

**Solution:**

A diagram of a mathematical equation

Description automatically generated with medium confidence

Figure . Summary of the approach for the integration

Line integration for **pointlist1**: -0.040104

Line integration for **pointlist2**: -10.327510

The same code can be used, just assign line\_list and points\_list to the respective line and point list that match the scenario:

|  |
| --- |
| line\_list\_1 = importdata('LINE\_LIST\_1.DAT');  line\_list\_2 = importdata('LINE\_LIST\_2.DAT');  point\_list\_1 = importdata('POINT\_LIST\_1.DAT');  point\_list\_2 = importdata('POINT\_LIST\_2.DAT');  % create edges  line\_list = line\_list\_2;  point\_list= point\_list\_2;  % -------------------------------------  line = [line\_list(1,2)];  for n = 2: length(line\_list(:,1))  index = find(line\_list(:,2) == line(end));  line(n) = line\_list(index,3);  end  line = line';  wrapped\_line = [line(end);line;line(1)];  XY = point\_list(line, 2:3);  wrappedXY = point\_list(wrapped\_line, 2:3);  figure(1), clf  plot(wrappedXY(:,1), wrappedXY(:,2))  deltaXY = wrappedXY(3:end,:) - wrappedXY(1:end-2,:);  n\_v = deltaXY ./ sqrt(deltaXY(:,1).^2 + deltaXY(:,2).^2) ;  n\_v = [n\_v(:,2), -n\_v(:,1)];  quiver(XY(:,1), XY(:,2), n\_v(:,1), n\_v(:,2))  title('normal vector')  % hold on  for n = 1:length(XY(:,1))  if mod(n, 10)==0  text(XY(n,1),XY(n,2), string(n))  end  end  quiver(XY(:,1), XY(:,2), J\_field(:,1), J\_field(:,2))  title('J vector field')  % The direction of the vector field were plotted at each points and it  % points from the bottom right to the top left  J\_field = point\_list(line, 4:5);  J\_field\_projection\_for\_each\_point = dot(J\_field, n\_v, 2);  J\_field\_projection = [J\_field\_projection\_for\_each\_point;  J\_field\_projection\_for\_each\_point(1)];  % calculate lengeth between each 2 points / 2  new\_wrapped\_line = wrapped\_line(2:end);  new\_wrappedXY = point\_list(new\_wrapped\_line, 2:3);  differenceXY = new\_wrappedXY(2:end,:) - new\_wrappedXY(1:end-1,:);  length\_segments = sqrt(differenceXY(:,1).^2 + differenceXY(:,2).^2);  line\_integration = sum(J\_field\_projection(1:end-1).\*(length\_segments/2) ...  + J\_field\_projection(2:end).\*(length\_segments/2));  fprintf('Line integration for pointlist: %f', line\_integration) |

b. After getting your results, what can you say about your solution with respect to your understanding of the **divergence operator**? Also, if the results were not exactly what you expected, can you suggest a reason why? What is the influence of the **additional structure in the additional domain**?

What can you say about your solution with respect to your understanding of the divergence operator

* Here we have J as a vector field. Divergence, or the integration we calculated, represents the **flux of a vector field** across the curve.

Also, if the results were not exactly what you expected, can you suggest a reason why?

* The direction of the normal vector and J\_field at each point were plotted. The J\_field points from the bottom right to the top left, and a big part of the line has the normal vector and J opposite each other. Thus, we get more negative values from the dot products, and thus the negative value for integration.

|  |  |
| --- | --- |
|  |  |

What is the influence of the **additional structure in the additional domain**?

* The additional structure in the domain causes more rapid change from high to low potential through the curve. As expected, magnitude-wise, b) has a higher magnitude compared to a).

c. In class, we performed an integration **over a singl­­­e contour line** and produced the following:



If I were to change the integral to , what would the result be **for all points**. If I were to change the integral to  **for all points**, what would the result be? For this problem assume the same 2-node domain that was in the lecture with the “rooftop” function.

**Solution:** We can write out the integration formula by hand as below.

**A close-up of math equations

Description automatically generatedA close-up of a paper with mathematical equations

Description automatically generatedA notebook with writing on it

Description automatically generated**

For point list1:

Line integration with **phi1** for pointlist: -0.016334

Line integration with **phi2** for pointlist: -0.038600

For point list2:

Line integration with **phi1** for pointlist: -5.164692

Line integration with **phi2** for pointlist: -13.773760

All code attached below:

|  |
| --- |
| IF we phave phi1 in the integration  line\_list\_1 = importdata('LINE\_LIST\_1.DAT');  line\_list\_2 = importdata('LINE\_LIST\_2.DAT');  point\_list\_1 = importdata('POINT\_LIST\_1.DAT');  point\_list\_2 = importdata('POINT\_LIST\_2.DAT');  % create edges  line\_list = line\_list\_1;  point\_list= point\_list\_1;  % -------------------------------------  line = [line\_list(1,2)];  for n = 2: length(line\_list(:,1))  index = find(line\_list(:,2) == line(end));  line(n) = line\_list(index,3);  end  line = line';  wrapped\_line = [line(end);line;line(1)];  XY = point\_list(line, 2:3);  wrappedXY = point\_list(wrapped\_line, 2:3);  deltaXY = wrappedXY(3:end,:) - wrappedXY(1:end-2,:);  n\_v = deltaXY ./ sqrt(deltaXY(:,1).^2 + deltaXY(:,2).^2) ;  n\_v = [n\_v(:,2), -n\_v(:,1)];  J\_field = point\_list(line, 4:5);  J\_field\_projection\_for\_each\_point = dot(J\_field, n\_v, 2);  J\_field\_projection = [J\_field\_projection\_for\_each\_point;  J\_field\_projection\_for\_each\_point(1)];  % calculate lengeth between each 2 points / 2  new\_wrapped\_line = wrapped\_line(2:end);  new\_wrappedXY = point\_list(new\_wrapped\_line, 2:3);  differenceXY = new\_wrappedXY(2:end,:) - new\_wrappedXY(1:end-1,:);  length\_segments = sqrt(differenceXY(:,1).^2 + differenceXY(:,2).^2);  line\_integration = sum(J\_field\_projection(1:end-1).\*(length\_segments/3) ...  + J\_field\_projection(2:end).\*(length\_segments/6));  fprintf('Line integration with phi1 for pointlist: %f', line\_integration)  For line integration with phi2  line\_list\_1 = importdata('LINE\_LIST\_1.DAT');  line\_list\_2 = importdata('LINE\_LIST\_2.DAT');  point\_list\_1 = importdata('POINT\_LIST\_1.DAT');  point\_list\_2 = importdata('POINT\_LIST\_2.DAT');  % create edges  line\_list = line\_list\_2;  point\_list= point\_list\_2;  % -------------------------------------  line = [line\_list(1,2)];  for n = 2: length(line\_list(:,1))  index = find(line\_list(:,2) == line(end));  line(n) = line\_list(index,3);  end  line = line';  wrapped\_line = [line(end);line;line(1)];  XY = point\_list(line, 2:3);  wrappedXY = point\_list(wrapped\_line, 2:3);  deltaXY = wrappedXY(3:end,:) - wrappedXY(1:end-2,:);  n\_v = deltaXY ./ sqrt(deltaXY(:,1).^2 + deltaXY(:,2).^2) ;  n\_v = [n\_v(:,2), -n\_v(:,1)];  J\_field = point\_list(line, 4:5);  J\_field\_projection\_for\_each\_point = dot(J\_field, n\_v, 2);  J\_field\_projection = [J\_field\_projection\_for\_each\_point;  J\_field\_projection\_for\_each\_point(1)];  % calculate lengeth between each 2 points / 2  new\_wrapped\_line = wrapped\_line(2:end);  new\_wrappedXY = point\_list(new\_wrapped\_line, 2:3);  differenceXY = new\_wrappedXY(2:end,:) - new\_wrappedXY(1:end-1,:);  length\_segments = sqrt(differenceXY(:,1).^2 + differenceXY(:,2).^2);  line\_integration = sum(J\_field\_projection(1:end-1).\*(length\_segments/1) ...  + J\_field\_projection(2:end).\*(length\_segments/3));  fprintf('Line integration with phi2 for pointlist: %f', line\_integration) |

**Problem #2.** **Understanding Current:** **Conservation of cortical current** is governed by the PDE

(1)

where  is the **electrical current density**. Often  is expressed with respect to **tissue potential changes**, this can be expressed as the **gradient** of **a scalar potential Φ**, and electrical conductivity *s.*

(2)

Hence equation (1) can be recast in terms of Φ as,

(3)

under **homogeneous electrical conductivity**, it can be written as (Laplace equation):

(4)

Your job is to compute *point iterative solutions­* of (4) when discretized by **center finite differences** under the following scenarios:

**(a)** Even though you will be solving equation (4) below with respect to the computational model, in this part write out the **full FD description approach for equation (3)**. Be sure to address the material property.

**(b)** Given the problem **domain** and **boundary conditions** below, **compute the solution of Φ** using ***Jacobi* iteration** with an initial solution vector **of Φ=0 everywhere**.

* Iterate until reaching an ***absolute L∞ norm-based error*** of successive iterates of **less than 1x10-5** and report the number of iterations needed to reach this convergence criterion.

(Answer below)

* Plot contours of your solution over the computational domain and report the actual numerical value of Φ for at the point x=0.7 and at y=0.7 Estimate the spectral radius of the Jacobi iteration matrix during the course of the iterations and compare with the theoretically expected value.

(Answer below)





cos(y)





x=y=0.05







**(c)** Repeat part **(a) using Gauss-Seidel**. Be sure to report the solution value requested in part (a) in order to verify that your solution is essentially unchanged. Is the speed-up in terms of convergence rate relative to Jacobi in agreement with theory? If the convergence critierion were extended to 1x10-6 how many more iterations would you ***predict*** would be needed with Gauss-Seidel? Is your prediction in reasonable agreement with practice?

**Answer:** for both b) and c):

>> For Tol=1e-5 we need:

**787 iterations** for Jacobi,

Jacobi V(x==0.7, x==0.7): 0.870189353342

theroretical\_spectral\_r\_Jacobi 0.987662994499

numerical\_spectral\_Jacobi 0.993177463840

**425 iterations** for GaussSeidel,

Gauss Seidel V(x==0.7, x==0.7): 0.870212161185

theroretical\_spectral\_r\_Gauss 0.975325988997

numerical\_spectral\_r\_Gauss 0.979454652024

A comparison of colored circles

Description automatically generated

Figure . Contour plots with number of iterations for Jacobi (left) and Gauss Seidel (right) for Tol=1e-5

For Tol=1e-6 we need

**973 iterations** for Jacobi

Jacobi V(x==0.7, x==0.7): 0.870226300688

theroretical\_spectral\_r\_Jacobi 0.987662994499

numerical\_spectral\_Jacobi 0.993177463840

**518 iterations** for GaussSeidel

Gauss Seidel V(x==0.7, x==0.7): 0.870228577832

theroretical\_spectral\_r\_Gauss 0.975325988997

numerical\_spectral\_r\_Gauss 0.979454652024

A comparison of a diagram

Description automatically generated with medium confidence

Figure . Contour plots with number of iterations for Jacobi (left) and Gauss Seidel (right) for Tol=1e-6

Is the speed-up in terms of convergence rate of GaussSeidel relative to Jacobi in agreement with theory?

* In both cases of Tol=1e-5 and Tol=1e-6 we saw a close to half reduction in the number of iterations for GaussSeidel vs. Jacobi.

From theory, Jacobi has a convergence rate of O(h^4) whereas GaussSeidel has O(h^2), thus the fact GaussSeidel saw a close to x2 times speed-up for both cases agrees with theory.

If the convergence critierion were extended to 1x10-6 how many more iterations would you ***predict*** would be needed with Gauss-Seidel? Is your prediction in reasonable agreement with practice?

We have the theoretical spectral radius as:

where *h* is our step size and *a* is the size of 1 edge of the domain. Thus, from the formula Tolerance does not necessarily influence the spectral radius. In other words, we cannot really estimate a good number of iterations from the theoretical formula above.

But once we numerically calculated spectral radius based on the largest error ratio through all iterations, we can estimate the number of iterations by using

, which differs quite a bit from the actual number of iterations used for Jacobi and Gauss in both Tol limits.

**For Tol=1e-5**

* Theoretical number of iterations for Jacobi at Tol=0.000010: **185**.486843133081
* Estimated number of iterations for Jacobi at Tol=0.000010: **336**.344336190154
* Theoretical number of iterations for GaussSeidel at Tol=0.000010: **92**.164172437790
* Estimated number of iterations for GaussSeidel at Tol=0.000010: **110**.918031224449

**For Tol=1e-6**

* Theoretical number of iterations for Jacobi at Tol=0.000001: **185**.486843133081
* Estimated number of iterations for Jacobi at Tol=0.000001: **336**.344336190154
* Theoretical number of iterations for GaussSeidel at Tol=0.000001: **92**.164172437790
* Estimated number of iterations for GaussSeidel at Tol=0.000001: **110**.918031224449

We can see the theoretical and estimated number of iterations do not change with respect to Tol value. We can see the estimation differes from the actual number of iterations reported earlier.