Assignment 3:
The Eikonal Equation & Spectral Geometry

TA in charge of the exercise: Ido Imanuel (ido.imanuel@gmail.com)
Final submission date: The 12th of January (12/1/2020)
Relevant course material: Lectures 1-9, Recitations 1-9

1. **Homework scale:**
   Single/Pair submission - This assignment holds two questions which are both mandatory.
2. **Submission Format.** Please submit your solutions to the appropriate homework submission box on Moodle as a ***zip*** file – with the file name ***HW3_ID.zip*** for single submission and ***HW3_ID1_ID2.zip*** for pair submission (where IDs are you and your partner's ID number).
   Your solution zip should contain:
   - A single PDF file with the name ***HW3_ID_Solution.pdf*** for single work and ***HW3_ID1_ID2_Solution.pdf*** for pair submission. Your report should be **typed** and presented in **English**. No other presentation format will be accepted. The report should conclude all results and figures you obtain in the wet parts.
   - Source code implementation for you wet part. Please make sure the code runs directly after uncompressing the zip file. Code must be written **solely** in the Python language unless directly stated otherwise. You may use any Python module you like, but we expect to see mostly your own coding here.
   - Additional miscellaneous (such as GIF files, images etc.) as requested in the assignment.
3. **Grading.** Please clearly show your work. Partial credit will be awarded for ideas *toward* the solution, so please submit your thoughts on an exercise even if you cannot find a full solution. *If you don't know where to get started with some of these exercises, just ask!* A great way to do this is to leave questions on our Piazza forum or to schedule reception hours with the TA in charge of the exercise. Maximal grade in this exercise is **115**, including all possible bonuses.
4. **Collaboration and External Resources.** You are encouraged to discuss all course material with your peers, including the written and coding assignments. You are especially encouraged to seek out new friends from other disciplines (CS, Math, EE, etc.) whose experience might complement your own. However, your final work must be your own, i.e., direct collaboration on assignments is prohibited. You are allowed to refer to any external resources - if you use one, cite such help on your submission. If you are caught cheating, you will get a zero for the entire course.
5. **Late Days.** Due to the large period of time given to this exercise's submission, we will accept no late day requests unless for official Technion reasons - reserve duty, hospitalization, or tragedy. Please be responsible with this
6. **Parenting Concessions.** By Technion regulations, parents are eligible for concessions with the homework. Please contact the Head TA for details if you are eligible.
7. **Warning!** With probability 1, there are typos in this assignment. If anything in this handout does not make sense (or is blatantly wrong), let us know!

0 Preliminary

1. Sign up and download the FAUST[1] dataset. The training registered scans (a subset of the dataset) are watertight meshes holding 6890 vertices each, decomposed into 10 different human subjects, with 10 approximately ***isometric*** poses each (naming convention - tr_reg_0XX.ply), for a total of 100 different triangular meshes. We will be using this dataset in this exercise.

2. Some parts of this exercise rely on Q1-HW1. Specifically, we require some visualization tools and some of your old implementations (such as the vertex adjacency matrix and the barycentric vertex areas). If you would like, you may embed the functions in HW3 as `Mesh` class subroutines – but this is not required. If you have not done Q1-HW1, complete only the minimum needed for this exercise. Instead of writing your own loader, use the `meshio` package to load. off/.ply/etc. as needed.

[1] http://faust.is.tue.mpg.de/overview

1 The Eikonal Equation (Wet)

In this question, we will demonstrate applications of the Eikonal equation solutions in both 2D weighted Euclidean domains and triangular meshes. In both cases, you will be using black-box implementations of the appropriate solvers. For 2D weighed Euclidean domains use the **eikonalfm**[2] package. For triangular meshes use the **tvb-gdist**[3] package. The second is quite slow, so we recommend computing the geodesic distance matrix you need for each mesh only once and saving it on disk for future uses or using the per-vertex distance computation if possible. Install by following the instructions on each attached site in the footnotes.

Let $\mathcal{X}$ be some Riemannian manifold and $\boldsymbol{T}$ a distance map defined on the surface. When the setting is 2D and Euclidean, instead of solving $\left\|\nabla_{\mathcal{X}}\boldsymbol{T}(\boldsymbol{x},\boldsymbol{y})\right\|=1$, we will use various functions $\mathcal{F}:\mathcal{X}\times\mathcal{X}\to\mathbb{R}$ (2D input matrices arrays) as weights corresponding to the ***propagation cost*** or "slowness", inversely proportional to the propagation velocity or ***traversability*** and solve the 2D weighed Euclidean Eikonal equation:

$$\left\|\nabla_{\mathcal{X}}\boldsymbol{T}(\boldsymbol{x},\boldsymbol{y})\right\|=\mathcal{F}(\boldsymbol{x},\boldsymbol{y})$$

with the appropriate boundary conditions.

1.1. Maze Solutions (2D Euclidean Domain)

a. ***Discretized Metric:*** Calculate the distances inside the maze supplied with this exercise (**maze.png)** using the Fast Marching Method (FMM) and a source point at $(\boldsymbol{x},\boldsymbol{y})=(814,383)$. Treat image boundaries as walls (think about what should be the traversability or "speed" in the Eikonal equation definition).

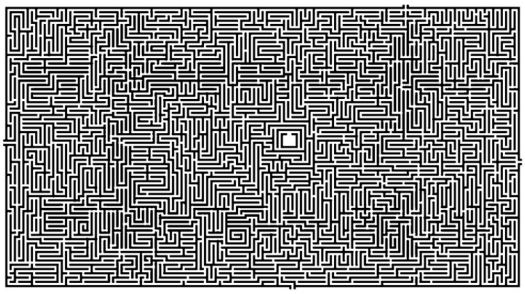

Figure 1: A binary maze

b. Find the shortest path from inside the source, at $(814,383)$, to the target $(8,233)$ outside of it, by following the negative gradient direction of the distance function from **target to source**. Use a simple finite differences approximation for the gradient in the x and y directions. Plot the shortest path on top of the maze image. Make sure you receive approximately the same plot and path length if you flip around source and target (symmetry).

[2] https://github.com/kevinganster/eikonalfm

[3] https://github.com/the-virtual-brain/tvb-gdist

c. ***Discrete Metric:*** View the maze as a graph. Connect nearby pixels that are not walls with an unweighted edge and use the **`networkx`** package's implementation of the shortest path algorithm for graphs (Dijkstra). Plot the received path from both source to target and target to source and discuss the discrepancy between it and the FMM result.

1.2. Optical Path Length (2D Euclidean Domain)

Reread the definition of the Optical Path Length (OPL) between two points and recall its connection to Snell's law (HW1). Use FMM to compute the OPL between two points, one inside a pool with water, with coordinate $(x,y)=(399,499)$ and one outside, with coordinate $(x,y)=(0,0)$, using the indices of refraction given in "**pool.mat**". Use this[4] for extraction of the .mat file.

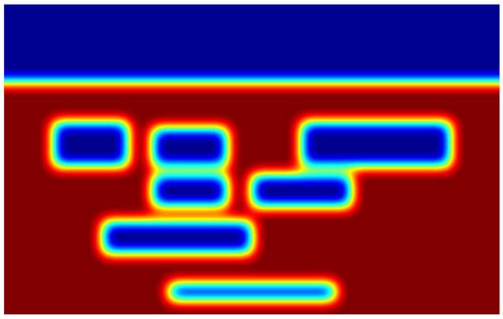

Figure 2: Indices of refraction as given in "pool.mat"

Plot the shortest path on top of the image of indices of refraction. Repeat similarly to the steps in 1.1. Expected results are supplied on Slide 35 of Recitation 7.

1.3. Segmentation of Objects (2D Euclidean Domain)

In this exercise we will use the Geodesic Active Contour (GAC) mathematical formulation as a basis for image segmentation (review the weighted Euclidean domains subject, as seen in the Lecture and Recitations).
Specifically, we aim to find a curve C that segments an image with a single salient object from its background.

Figure 3: Example of an image and its segmentation curve colored in cyan

[4] https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.loadmat.html

In the GAC setting, we aim to find the curve $\boldsymbol{C}(\boldsymbol{s})=\big(\boldsymbol{x}(\boldsymbol{s}),\boldsymbol{y}(\boldsymbol{s})\big)$ (where $\boldsymbol{s}$ is the Euclidean arclength) minimizing the following functional:

$$\mathcal{E}\triangleq\int_{C}\boldsymbol{g}\big(\boldsymbol{x}(\boldsymbol{s}),\boldsymbol{y}(\boldsymbol{s})\big)\boldsymbol{ds}$$

where we choose the function $\boldsymbol{g}$ (i.e., the potential or traversability) to take lower values on structures of interest. For example, for segmentation tasks, we will use a function appropriate to capture information about the hard edges in the image $\boldsymbol{I}$:

$$\boldsymbol{g}(\boldsymbol{x},\boldsymbol{y})\triangleq\frac{1}{1+\left\|\nabla\boldsymbol{K}_{\sigma}(\boldsymbol{x},\boldsymbol{y})*\boldsymbol{I}(\boldsymbol{x},\boldsymbol{y})\right\|_{2}}$$

i.e., we have $\boldsymbol{g}(\boldsymbol{x},\boldsymbol{y})$ as a decreasing function of the gradient magnitude of the image smoothed at some scale σ, obtained by convolution of the image with the gradient of a Gaussian kernel $\boldsymbol{K}_{\sigma}$ with standard deviation σ. The target image structure will be extracted by finding paths of minimal length (geodesics) under the traversability constraint $\boldsymbol{g}(\boldsymbol{x},\boldsymbol{y})$ at each point. The function $\boldsymbol{g}$ chosen above promotes geodesics that pass along edges of the image.

Implement the following segmentation algorithm:

Four-Pronged Geodesic Image Segmentation
Input: Image $\boldsymbol{I}$, hyperparameter σ
Init:

1. Choose a bounding box that contains the object in the image $\boldsymbol{I}$, denoted by four points $\left\{\boldsymbol{p}_{i}^{t=0}\right\}_{i=0}^{3}$.
2. Compute $\boldsymbol{g}(\boldsymbol{x},\boldsymbol{y})$.

While $\left|\mathcal{E}_{t}-\mathcal{E}_{t-1}\right|>\varepsilon$**:**

3. Compute the geodesic between every ***adjacent*** pair of points (pixels) $\boldsymbol{i},\boldsymbol{j}$ (4 geodesics, where $\boldsymbol{j}\equiv(\boldsymbol{i}+1)\bmod 4$) and find the midway points $\left\{\boldsymbol{q}_{k}\right\}_{k=0}^{3}$ along each path where the distance functions are the same, i.e., $\mathcal{D}_{p_{i}^{t}}=\mathcal{D}_{p_{j}^{t}}$.
4. Set $\left\{\boldsymbol{p}_{i}^{t=t+1}\right\}_{i=0}^{3}=\left\{\boldsymbol{q}_{i}\right\}_{i=0}^{3}$.

End While

Test your implementation on a few images from the web, both with cluttered and clear backgrounds. Show images or gifs of your results at each time step. Report a graph of the energy:

$$\mathcal{E}_{t}\triangleq\sum_{i=0}^{3}\int_{p_{i}^{t}}^{p_{j}^{t}}\boldsymbol{g}\big(\boldsymbol{x}(\boldsymbol{s}),\boldsymbol{y}(\boldsymbol{s})\big)\boldsymbol{ds}$$

which at every iteration $\boldsymbol{t}=\{0,1,2,...\}$, integrates over all the weights along each of the four geodesics $\left\{\boldsymbol{C}_{i}^{t}\right\}_{i=0}^{3}$ of time $\boldsymbol{t}$.

Note: We hold:

$$\mathcal{D}_{p_{i}}\left(\boldsymbol{p}_{j}\right)\equiv\int_{p_{i}}^{p_{j}}\boldsymbol{g}\big(\boldsymbol{C}(\boldsymbol{s})\big)\boldsymbol{ds}$$

Explain why in your answers.

1.4. Canonical Shapes via MDS (Triangular Meshes)

Open the two isometric triangular meshes **man1.ply**, **man2.ply**. Review the subjects of MDS and Spherical MDS embedding from the lecture, and embed each triangular mesh's geodesic distance matrix $\mathcal{D}$ using:

1. MDS embedding (into $\mathbb{R}^3$, using your implementation from HW2)
2. Spherical MDS embedding (into $\mathcal{S}^2$).

Recompute the geodesic distance matrix $\hat{\mathcal{D}}$ on the resulting shape and report the embedding error $\mathcal{E} = \left\| \mathcal{D} - \hat{\mathcal{D}} \right\|_F$. Plot the original pair and their embedding next to each other. Explain the results.

1.5. Farthest Point Sampling (Triangular Meshes)

In this subsection, we will consider the problem of discrete shape simplification (compression), i.e., decimating some high dimensional shape in a manner that best preserves its geometry. When our discretization is in the form of triangular meshes, this usually involves two steps:

1. ***Sample*** the vertex set, i.e., choose the most jointly informative vertices out of the original vertex set $\mathcal{V}$
2. ***Remeshing*** - Fix proper triangular connectivity atop the new vertex set.

In this exercise, we address only Step 1. While uniformly removing at random some of the vertices is very simple, it relies on the assumption that the shape itself has been uniformly sampled from the continuous manifold, which does not have to be the case. In this section, we will review and implement the Farthest Point Sampling algorithm – a rather simple method that relies on geodesic distances. This method's resulting sampling can be proven to be a 2-approximation of the ***optimal*** sampling of the shape in the 2D Euclidean setting (for the Euclidean setting, please see the extract attached and video lecture to this exercise).

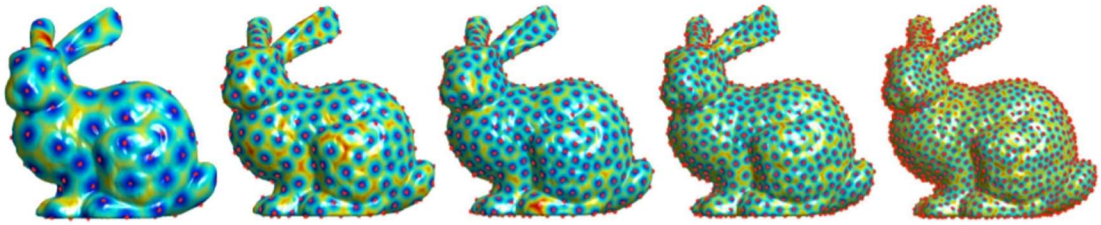

Figure 4: Farthest Point Sampling example, with increasing target number of points from left to right

Implement the following algorithm:

Farthest Point Sampling on Triangular Meshes

Input: A triangular mesh $\mathcal{X} = (\mathcal{V}, \mathcal{F})$ and a target number of points $n \in \mathbb{N} \mid n < |\mathcal{V}|$

Init: Let $s_1 = v_i$ for some arbitrary $v_i \in \mathcal{V}$ and let $\mathcal{S} = \{s_1\}$.

While $|\mathcal{S}| \neq n$:

Find the point $v_f \in \mathcal{V}$ which is farthest away from the selected points in $\mathcal{S}$. That is,

$$v_f \triangleq \arg\max_{v \in \mathcal{V}} \left\{ \min_{s_i \in \mathcal{S}} d_{\mathcal{X}}(v, s_i) \right\}$$

Insert v_f to $\mathcal{S}$.

Showcase the resulting point clouds w.r.t original ones on a few meshes from FAUST, while varying n (similar to Figure 4).

Notes:

- Schemes to insert a new connectivity on to $\mathcal{S}$ exist but are not covered in this exercise. If you are interested, please see the attached article[5] for details.
- The sampling may also be done non-uniformly, by weighing each vertex with some weight $g(v)$ (anisotropic sampling) and altering the Eikonal solver to work on weighted domains.
- We may generalize the method to other metric spaces (such as a Gaussian curvature induced metric) to create compact and highly informative descriptors of the shape with a size of our choosing.

[5] Geodesic Remeshing Using Front Propagation (Gabriel Peyré, Laurent D. Cohen)

2 Spectral Geometry of Shapes (Wet)

Preliminaries:

1. Review the (half) cotangent weight scheme proposed in Meyer et al (2012)[6] . Implement in code the discrete Laplace-Beltrami operator for triangular meshes as detailed in the article. You may also see the needed formulation at the end of Recitation 8. Furthermore, implement in code the discrete graph Laplacian (as defined in HW2, while viewing our triangular meshes as graphs). Specifically, write three functions:

- `def weighted_adjacency(v,f,cls='half_cotangent')`
  Which receives the shared vertex mesh tuple `v,f` and a string `cls` in `{'half_cotangent','uniform'}` and returns:
  The *sparse* cotangent weight matrix if `cls=='half_cotangent'`:

$$(\mathcal{W})_{ij} = \begin{cases} \frac{1}{2}\left(\cot\alpha_{ij} + \cot\beta_{ij}\right) & i \neq j, i \sim j \\ 0 & else \end{cases}$$

  The *sparse* uniform weight matrix (i.e, the vertex adjacency matrix) if `cls=='uniform'`:

$$(\mathcal{W})_{ij} = \begin{cases} 1 & i \neq j, i \sim j \\ 0 & else \end{cases}$$

- `def laplacian(v,f,cls='half_cotangent')`
  Which receives the shared vertex mesh tuple `v,f` and a string `cls` in `{'half_cotangent','uniform'}` and returns the relevant Laplacian via the previous function and:

$$\mathcal{L} = \mathcal{D} - \mathcal{W}$$

  where $\mathcal{W}, \mathcal{D}$ are the weighted adjacency matrix and weighted degree matrix adhering to `cls`. The degree matrix is a *sparse* diagonal matrix received by diagonalizing the row wise summation vector of the weighted adjacency matrix.

- `def barycenter_vertex_mass_matrix(v,f)`
  This function will compute and return the *sparse* diagonal barycenter vertex areas defined by:

$$(\mathcal{M})_{ij} = \begin{cases} m_i & i = j \\ 0 & else \end{cases}$$

  where m_i denotes the vertex area of vertex i. Instead of the "mixed" areas presented in the paper, we will simplify the implementation by using the barycenter vertex areas, as detailed in HW1. This has little to no effect on the spectral decomposition of our Laplacians.

2. Implement an additional function `laplacian_spectrum(v,f,k,cls)` that computes the first k eigenfunctions and eigenvectors of the relevant Laplacian supplied by `cls`.
   Use scipy's eigensolver with the following command:

```
eig_val, eig_vec = scipy.sparse.linalg.eigsh(L, k, M, which='LM', sigma=0, tol=1e-7)
```

[6] http://multires.caltech.edu/pubs/diffGeoOps.pdf

This solves the generalized eigenvalue problem: $\mathcal{L}\boldsymbol{v} = \lambda\mathcal{M}\boldsymbol{v}$
Makes sure to round both the eigenvalues and eigenfunctions to 12 decimal places to safeguard against numerical inaccuracies. Did you receive a first eigenvalue of 0, with a constant eigenvector?

3. **Isometry invariance of the Laplacian**: Show the first 5 eigenfunctions as (color mapped) scalar functions on the vertices of a few pairs of isometric shapes from FAUST for both Laplacians. Are the eigenfunctions approximately isometry invariant? Explain your answer. Showcase any discrepancies.

4. **Connection of the Laplacian to the discrete Mean Curvature**:
When the area normalized Laplacian $\mathcal{M}^{-1}\mathcal{L}$ is applied to the vertex positions $\boldsymbol{V} \in \mathbb{R}^{|\mathcal{V}|\times 3}$, this operator supplies a pointwise (or rather integral average) approximation of the mean curvature normal $\mathcal{H}\boldsymbol{n}$:
$$\mathcal{H}\boldsymbol{n} \equiv \mathcal{M}^{-1}\mathcal{L}\,\boldsymbol{V} \in \mathbb{R}^{|\mathcal{V}|\times 3}$$
Stripping the magnitude off the rows of the resulting matrix would give the per-vertex ***unsigned*** mean curvature $|\mathcal{H}| \in \mathbb{R}^{|\mathcal{V}|\times 1}$. To recover the sign, we can check whether each row in $\mathcal{H}\boldsymbol{n}$ agrees or disagrees with the vertex normals, as defined in HW1.

- Show case the ***signed*** mean curvature on a few meshes from FAUST.
- Explore and explain in a paragraph or two the geometric connection of the mean curvature to the above formula.

5. **Laplacian applications on various scalar functions:**
Pick a mesh from FAUST and a few scalar functions on the vertices as you have seen in the recitations and lectures (for example, the Dirac function for some vertex of your choosing or the geodesic distance function, using Q1 in this exercise). For each of the two Laplacian discretizations
- Approximate each scalar function using the Laplacian eigen decomposition with varying bandwidth values (number of eigenvectors) and plot the result. Seeing we have normalized by the vertex mass matrix in the eigen decomposition definition, the appropriate approximation is given by:
$$\hat{\boldsymbol{f}} = \Phi_k \Phi_k^{\,T} \mathcal{M}\boldsymbol{f} \qquad \Phi_k = \begin{pmatrix} | & | & & | \\ \phi_1 & \phi_2 & \dots & \phi_k \\ | & | & & | \end{pmatrix}$$
- For a given function $\boldsymbol{f}$, plot the function's (area normalized) Laplacian $\mathcal{M}^{-1}\mathcal{L}\boldsymbol{f}$. Explain what you see.
Qualitatively explain the difference between the results with each discretization.

6. **Non-Rigid Shape Classification:** Read the attached extract[7] Recitations 6 and 8 and Lecture 9. Implement in code the following spectral descriptors as described in the extract and in Recitations 6 and 8:
   1. The ShapeDNA.
   2. The Global Point Signature (GPS).
   3. The Heat Kernel Signature (HKS). Use n=10 logarithmically sample time instances via:
      ```
      def expspace(start, stop, n):
      ```

[7] https://silo.tips/download/heat-kernel-signature

```
return np.exp(np.linspace(np.log(start), np.log(stop), n))
```

Experiment with different values of `start` and `stop`, choosing the most informative set (display each time step on isometric pairs of meshes). The heat kernel signature is known to be a ***local*** shape descriptor for low values of time ($t \to 0$), while it is a ***global*** one for large values of $(t >> 0)$. Explain qualitatively why.

Compute the three spectral descriptors and the mean curvature on every shape in the FAUST dataset. Flatten the four descriptors as needed into 1D vectors. Compute the pairwise distance matrix under the $\mathcal{L}_2$ norm between each descriptor to receive $\mathcal{D}_{DNA}, \mathcal{D}_{GPS}, \mathcal{D}_{HKS}, \mathcal{D}_{\mathcal{H}}$. For example, in $\mathcal{D}_{DNA}$ the cell (i, j) would house $\left\| f_i^{DNA} - f_j^{DNA} \right\|_2$, where f_i^{DNA} is the ShapeDNA descriptor for the i-th shape. Embed each matrix once into $\mathbb{R}^2$ via MDS and plot the result, supplying a color for each point once by its relevant subject id (1-10) and once by its relevant pose id (1-10). For ShapeDNA, GPS and HKS also experiment with the result for $k = 50, 200, 1000$. Explain what you see. What is the effect of bandwidth on the embedding? Which descriptor is most useful for shape classification? Which descriptor is most useful for pose classification? Detail any problems, explain their cause if possible.

Note: The FAUST dataset has a 1:1 correspondence between the vertices of each shape, making the direct comparison of vertex-based descriptors possible. One should note that this correspondence is usually not available to us

7. **(Bonus – 5 Points) Scale Invariant Metric:** Similarity between shapes can have a different interpretation depending on the metric one chooses to use. Two given shapes can be said to be isometric from a point of view of one metric and nonisometric from a point of view of a different one. To exemplify this statement, consider two different Riemannian manifolds that differ only by their metric, mainly $\{\mathcal{S}, g\}$ and $\{\mathcal{S}, \tilde{g}\}$ where g is the standard metric and $\tilde{g}$ is the scale invariant one. For example, consider a sphere of radius one, the pairwise distances on the surface overlap in both geometries, as the geodesic arc-length and the scale invariant arc-length coincide. Scaling the sphere by a by uniform factor results in non-isometric surface from the regular perspective and isometric surfaces from the scale invariant one.

   Let us examine another example. Load the attached spherocylinders ("sphero_cyl1.ply", "sphero_cyl2.ply"). Show the first 5 eigenfunctions of the regular LBO and the **Scale Invariant** LBO as (color mapped) scalar functions on the vertices of each spherocylinder. Are the eigenfunctions approximately isometry invariant under the scale invariant metric? Showcase any discrepancies.

   Note: The only change that is needed for the SILBO in contrast to the regular Laplacian is to alter the matrix $\mathcal{M}$ to be defined:

$$(\mathcal{M})_{ij} = \begin{cases} 2\pi - \sum \theta_i & i = j \\ 0 & else \end{cases}$$

   i.e., a diagonal matrix of the angle defects of each vertex, as seen in HW1. Review the lecture notes and show this is indeed the only change needed.