Due date: April 11, 2025

The last homework is here! Hopefully you've liked some of the HW problems this semester.

Problem 1: Let X be a matrix. Describe a relationship between the eigenvalues and singular values of $A = X^T X$.

Problem 2: Let S be a set of n points in $\mathbb{R}^2$. A point $x \in \mathbb{R}^2$ is called a *center point* if any half-plane containing x contains at least $\lfloor n/3 \rfloor$ points of S. It is known that a center point always exists.

(i) Describe an $O(n^2 \log n)$-time algorithm to compute a center point of S. You can assume that the intersection of m halfplanes can be computed in $O(m \log m)$ time. (In fact, the run time can be improved to $O(n^2)$.)

(ii) Describe an $O(n)$ time randomized algorithm that compute an approximate center point $\tilde{x}$ of S with probability at least $1/2$, i.e., any halfplane containing $\tilde{x}$ contains at least $n/4$ points of S.

Problem 3: (**Spectral clustering on image segmentation**) In this problem, you will implement a spectral clustering algorithm to perform image segmentation on images (grayscale OK, or colored if you wish!) **of your choice**. The exact implementation is also flexible, so you are allowed to play around with various setups!

Images are made of pixels, and it's natural that pixels in the same entity (e.g. foreground, or a specific object) should share more resemblance than between pixels from two entities. To this end, one may construct a graph, where the nodes are pixels, and an edge between two nodes indicate a high similarity between the pixels. Spectral clustering, which captures a graph's structural features, will then likely decompose the graph into human-interpretable components.

Here is a bucket list of what you will need to do. Hopefully some of them sound familiar at this point :)

- **Choose any 2-3 images you like**, of any reasonable dimension/resolution, and of either grayscale (easier) or colored (harder).
- A similarity matrix W. If the image has $m \times n$ pixels, then W is $(mn) \times (mn)$, where entry $W_{(x_1,y_1),(x_2,y_2)}$ measures the similarity between pixels (x_1, y_1) and (x_2, y_2), $x_i, y_i \in [m] \times [n]$. Suppose their grayscale intensities are I_1, I_2, respectively. **You are free to use any similarity measure as you wish**, but if you do not have any cool ideas, the **Gaussian RBF kernel** is a standard practice:

$$W_{(x_1,y_1),(x_2,y_2)} = \exp\left(-\left(\frac{\|(x_1,y_1)-(x_2,y_2)\|_2^2}{2\sigma_d^2} + \frac{(I_1-I_2)^2}{2\sigma_I^2}\right)\right)$$

The parameters σ_d, σ_I control the sensitivity to spatial distance and intensity, respectively. Play around to find some nice values to suit your needs. If you use colored images you may need to tweak the formula too.

A good starting point for σ_I would be roughly 10% *the range of intensities: if they are normalized to* $[0, 1]$*, then try* $\sigma_I \approx 0.1$*; if they are in the range* $[0, 255]$*, consider* $\sigma_I \approx 25$*. A good σ_d to start with would be from* 5 *to* 10*, but this greatly depends on your image.*

- Graph Laplacians. Instead of working with a $0-1$ adjacency matrix, we will work with the fractional W. Specifically, we define the following:
  - A "degree" matrix D: diagonal, with $D_{i,i} = \sum_j W_{i,j}$.
  - The unnormalized Laplacian $L = D - W$.
  - The normalized Laplacian $L_{\text{sym}} = D^{-1/2} L D^{-1/2}$.

  You are free to use any existing packages to your aid, **except the functions that directly output Laplacians.**

- Spectral embedding and clustering. Perform eigen-decomposition on both L and L_{sym}. Feel free to use anything here, for example Python's `numpy.linalg.eigh`. Repeat the remaining steps for each Laplacian.
  - (Sanity check) Your eigenvalues should be real and non-negative — if not, check for errors in previous parts. Sort them in *ascending* order $0 \leqslant \lambda_1 \leqslant \lambda_2 \leqslant \ldots$.
  - Select the *smallest* $k = 2$ *nonzero* eigenvalues and their corresponding eigenvectors $v_1, \ldots, v_k$. Also repeat the following for a few larger k's, like 3, 4, and so on.
  - Form the embedding matrix U. This is an $(mn) \times k$ matrix, where column j is the j^{th} eigenvector v_j, which has length mn (number of pixels).
  - For normalized Laplacian L_{sym}, normalize each row in U to have unit length.
  - Now it's time for clustering. Each row in U represents a k-dimensional embedding of a pixel. Run k-means on the rows of U. You are free to use any existing clustering tools.
  - Based on cluster membership, plot the segmentation result (with different colors or grayscale levels for different clusters).

Your expected **deliverables** are mostly just demo of your cool transformed images! For each of the image, describe the parameters you chose, and include a copy of your original image, as well as results for different configurations (varying k and both Laplacian types). Describe your findings – What does $k = 2$ look like? What does higher k capture? How do Laplacian normalization affect the results, if any? And anything else you wish to share?