CS 530 Fall 2025 Homework 3

Calvin Stahoviak – cstahoviak@unm.edu

November 24, 2025

Collaborators: N/A

By turning in this assignment, I agree to the UNM Student Code of Conduct and declare that all of this is my own work.

Code is available at https://github.com/cjstahoviak/CS-530-Geometric-and-Probabilistic-Methods-for-Computer-Scientists.

AI Acknowledgment: Claude Sonnet 4.5 was used to assist in writing code. Gemini 3 was used for in some instances to help with proofs.

Problem 1

(Prove the Associativity of Convolution)

(a) Prove the following:

$$((w * x) * y)(z) = (w * (x * y))(z) \tag{1}$$

Solution: Prior, know that convolution is defined in Equation (2):

$$(a * b)(m) = \sum_{n=0}^{N-1} a(n)b(m-n)_N \tag{2}$$

where $(m-n)_N$ is the modulus of $(m-n)$.

For the left-hand side of Equation (1), expand the outer most convolution using the definition of convolution from Equation (2):

$$\sum_{n=0}^{N-1} (w * x)(n)y(m-n) \tag{3}$$

Expand the inner most convolution using the definition of convolution, Equation (2), again:

$$\sum_{n=0}^{N-1} \left(\sum_{k=1}^{N-1} w(k)x(n-k) \right) y(m-n) \tag{4}$$

Move the $w(k)$ term which does not depend on n outside of the summation:

$$\sum_{k=0}^{N-1} w(k) \left(\sum_{n=0}^{N-1} x(n-k)y(m-n) \right) \tag{5}$$

Undo convolution using Equation (2):

$$\sum_{k=0}^{N-1} w(k)(x * y)(m-k) \tag{6}$$

Undo convolution using Equation (2) again:

$$(w * (x * y))(m) \tag{7}$$

Problem 2

(a) Prove that an N-dimensional column vector $\mathbf{z}$:

$$|\widehat{(R_k z)}(m)| = |\hat{z}(m)| \tag{8}$$

Solution: Note that a translation has the following definition:

$$R_k z(n) = z(n-k) \tag{9}$$

where k is the amount of steps translated.

Starting with $\widehat{(R_k z)}(m)$, perform the Fourier Transform:

$$\widehat{(R_k z)}(m) = \sum_{n=0}^{N-1} R_k z(n) e^{-2\pi j \frac{mn}{N}} \tag{10}$$

Apply the translation $R_k z$ as defined by Equation (9):

$$\sum_{n=0}^{N-1} z(n-k) e^{-2\pi j \frac{m(n-k)}{N}} e^{-2\pi j \frac{mk}{N}} \tag{11}$$

Pull out the term that does not depend on n from the summation:

$$e^{-2\pi j \frac{mk}{N}} \sum_{n=0}^{N-1} z(n-k) e^{-2\pi j \frac{m(n-k)}{N}} \tag{12}$$

Change variables $(n-k)$ back to m by taking advantage of periodicity:

$$e^{-2\pi j \frac{mk}{N}} \hat{z}(m) \tag{13}$$

Taking the absolute value of Equation (13) combined with Equation (8) results in:

$$|\widehat{(R_k z)}(m)| = \left| e^{-2\pi j \frac{mk}{N}} \hat{z}(m) \right| = |\hat{z}(m)| \tag{14}$$

(b) Prove that convolution is also translation invariant:

$$((R_k z) * w)(m) = (R_k(z * w))(m) \tag{15}$$

Solution: For the left-hand side, expand using the definition of convolution, Equation (2):

$$((R_k z) * w)(m) = \sum_{n=0}^{N-1} (R_k z)(n) w(m-n) \tag{16}$$

Apply the translation as defined in Equation (9):

$$\sum_{n=0}^{N-1} z(n-k) w(m-n) \tag{17}$$

Perform a change of variables where $l = n - k$. From such we get $n = l + k$.

$$\sum_{l=0}^{N-1} z(l) w(m-(l+k)) \tag{18}$$

Rearrange index variables:

$$\sum_{l=0}^{N-1} z(l) w((m-k)-l) \tag{19}$$

Use the definition of convolution, Equation (2), again:

$$(z * w)(m-k) \tag{20}$$

Apply the definition of translation, Equation (9):

$$(R_k(z * w))(m) \tag{21}$$

Problem 3

(Compression)

(a) Perform wavelet compression on the 1D signal below and compare your results to DCT.

In previous HW#2, we have used a step function. In this one, you will use the following 1D Gaussian curve as input and practice compression (1:2, 1:4, 1:8, 1:16) using dwt.

Note that the dwt and idwt function in Matlab only performs 1st stage wavelet transform. How do you perform 2-stage or 3-stage wavelet transform to achieve compression ratio of 1:4 or 1:8?

Matlab code for creating a vector of dimension 1024 of a normal distribution with μ = 50 and σ = 10:

```
x = 0:0.1:102.3 ;
f = normpdf(x,50, 10);
figure(1)
plot(f,'LineWidth',4) ;
title ('original signal') ;
```

Solution: This problem was solved using Python, not Matlab. Python's version of the `dwt` and `idwt` functions are in the `pywt` Python package. Unlike the `dwt()` and `idwt()` functions from Matlab, the `pywt` package allows you to set a parameter called `level` to do multi-stage wavelet transforms.

A comparison of Wavelet and DCT compression at compression ratios (1:2, 1:4, 1:8, 1:16) is shown in Figure 1. The MSE is shown for each compression above the plot. The MSEs are also compared against each other in Figure 2. We notice that DCT maintains lower compression error at all times.

Problem 4

(Wavelet Denoising))

(a) Consider the same signal z as in problem 3. Use Matlab functions `normrand` or `wgn` to add Gaussian noise to the input signal. Use wavelet transform and Fourier transform to remove the noise and compare the results.

Solution: The following solution was done in Python. Instead of using the Matlab functions `normrand`, the python method `np.random.normal()` from the `NumPy` library.

The original signal, noisy signal, wavelet transform and Fourier transform are all shown in Figure 3. Visually, wavelet trasform does a much better job of smoothing our the harsh signal created after adding noise. Wavelet transform also has a lower MSE.

Problem 5

(2D FFT, DCT and Wavelet Transform for Image Compression)

(a) In this problem you will practice with 2D transforms for image compression. Please create your own image and perform compression using 2D transforms such as fft2, dct2, and dwt2.

Hints:

- You will need functions `imread` and `imshow` to read in an image and display an image.

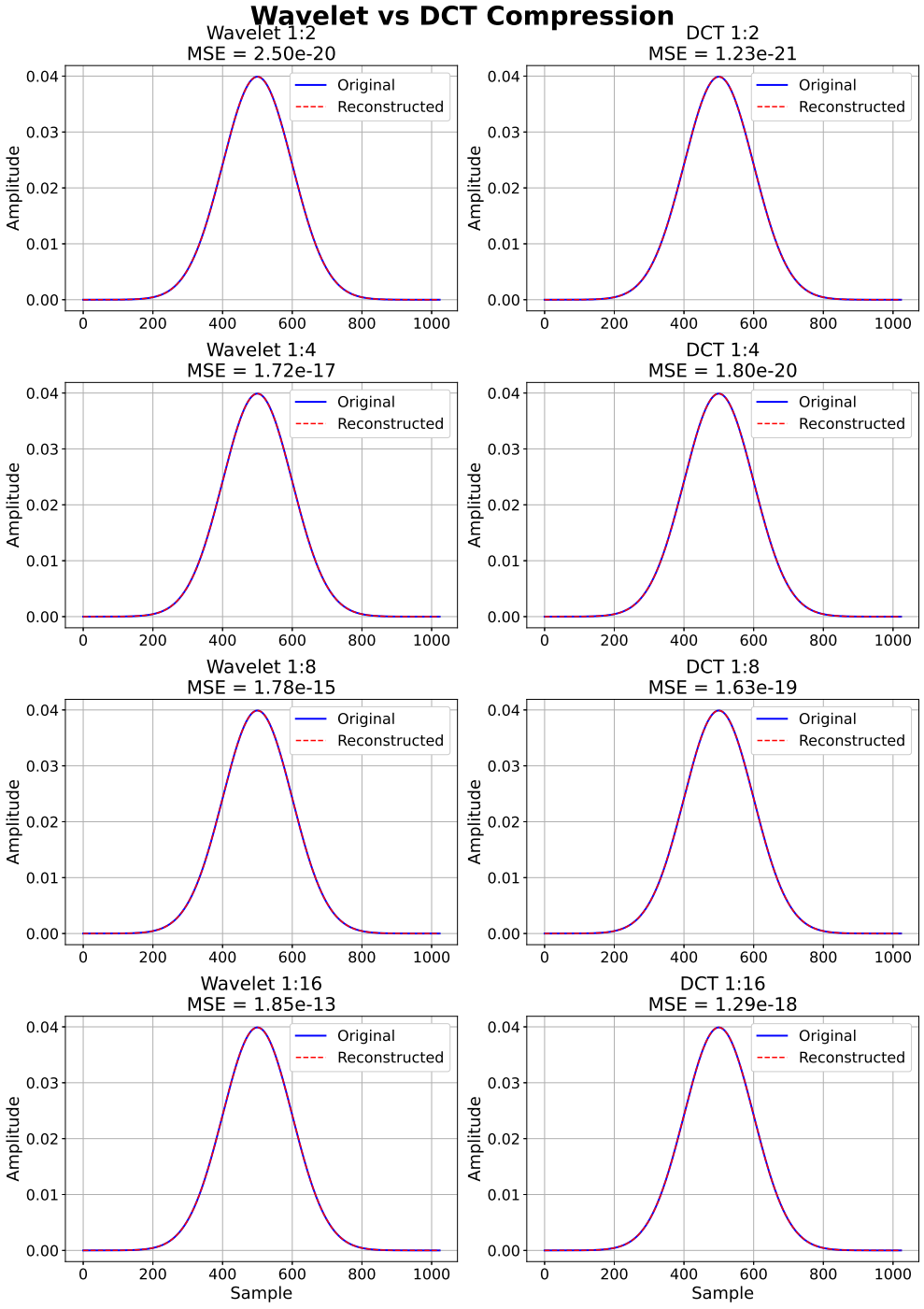

Figure 1: Comparison of Wavelet and DCT compression methods at different compression ratios (1:2, 1:4, 1:8, 1:16). All forms of compressions are visually similar to the original signal with minor errors.

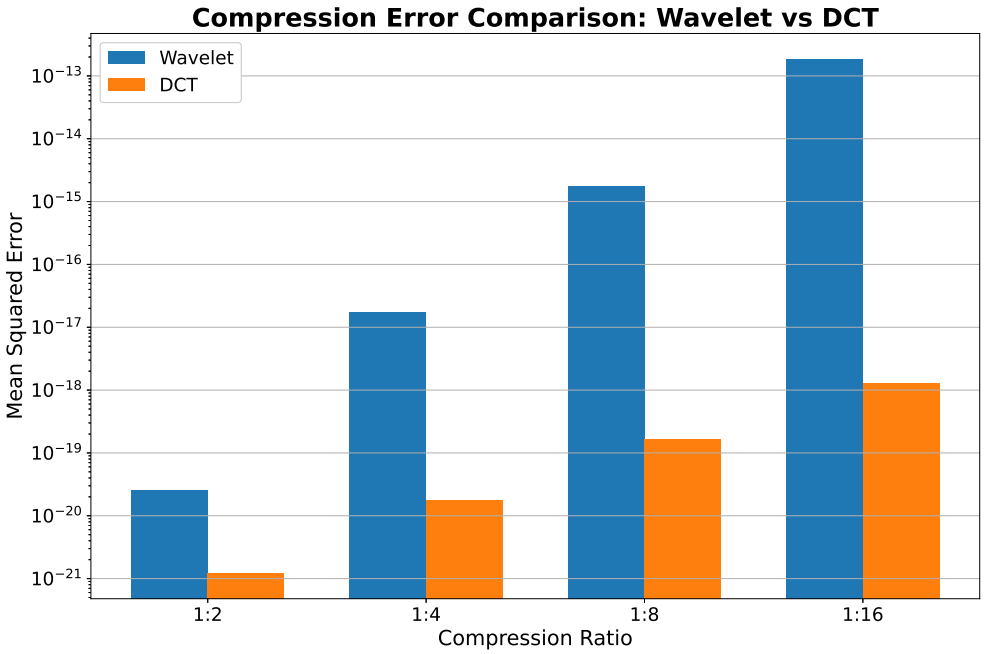

Figure 2: Mean Squared Error comparison between Wavelet and DCT compression methods across different compression ratios. The log scale y-axis shows that both methods exhibit increasing error with higher compression ratios, with the relative performance difference between the two methods visible at each compression level.

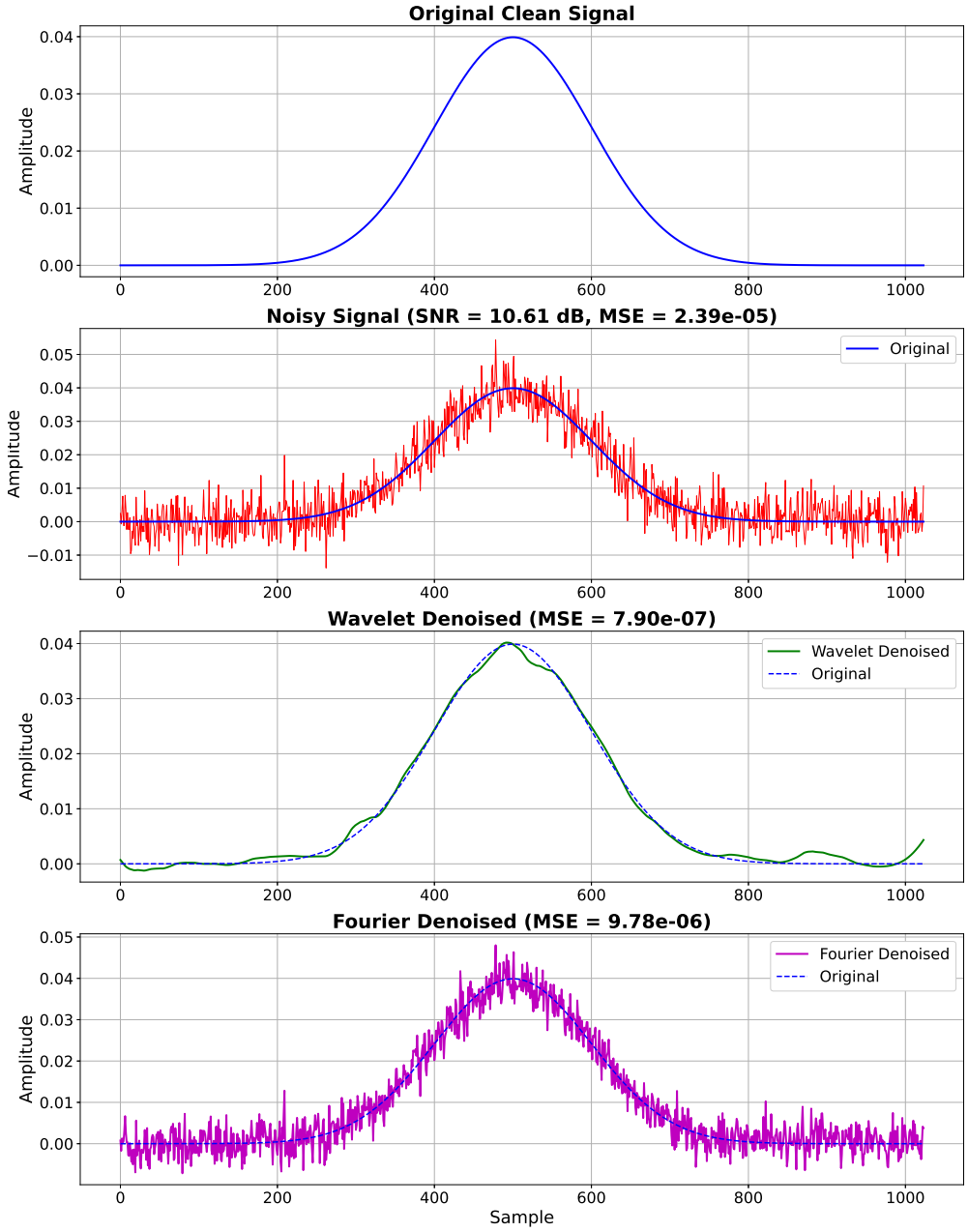

Figure 3: The top two plots show the original and noisy signal. The bottom two plots show the denoising effect of applying the wavelet and Fourier transform to the noisy signal.

- You may also consider concerting your RGB images into a gray level images first using rgb2gray.

Solution: A 2-dimensional fast Fourier transform, discrete cosign transform, and discrete wavelet transform was done for $(1:4, 1:16, 1:64, 1:256)$ compression rations and visualized in Figure 4. The Peak Signal-to-Noise Ratio (PSNR) for each instance is given which is a metric that indicates the quality of a compression.

Note that wavelet has the best compression quality in terms of PSNR up until a $1:256$ compression ratio where it quickly collapses. This is more clearly seen in the image compression quality comparison, Figure 5.

Figure 4: A photo of Professor Luan compressed using FFT, DCT, and Wavelet at $(1:4, 1:16, 1:64, 1:256)$ compression rations.

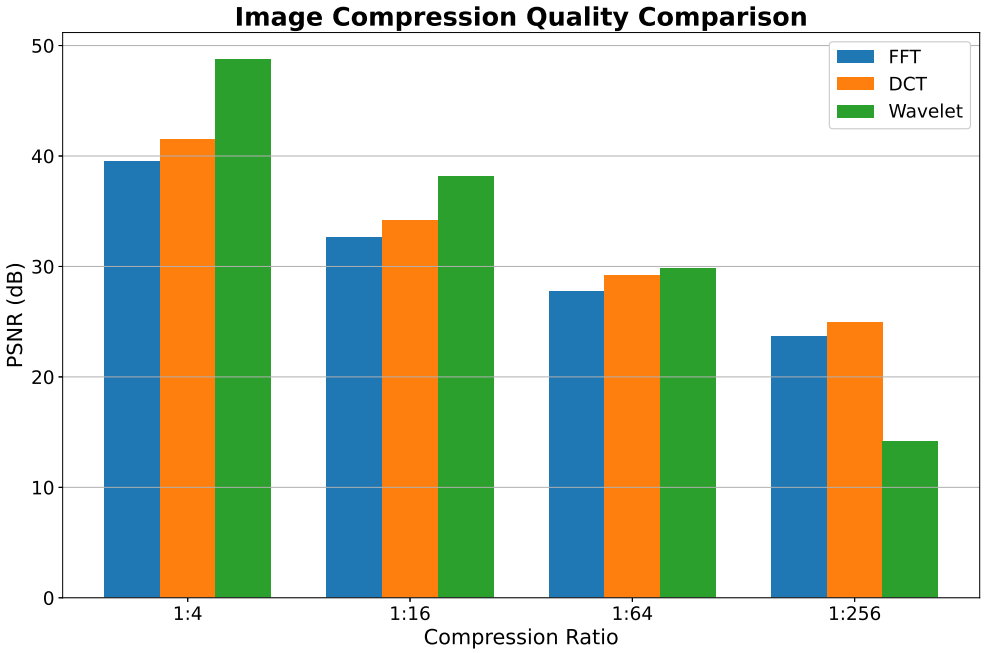

Figure 5: A side-by-side comparison of the PSNR of FFT, DCT, and Wavelet at $(1:4, 1:16, 1:64, 1:256)$ compression rations. Note that Wavelet suddenly collapses at a $1:256$ compression ratio.

Problem 6

(a) Calculate the gradient and Hessian of the following function f.

$$f(\mathbf{x}) = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{22}$$

Start with the analytical form of f:

$$f(\mathbf{x}) = a_{1,1}x_1^2 + a_{2,2}x_2^2 + a_{3,3}x_3^2 + (a_{1,2}+a_{2,1})x_1x_2 + (a_{1,3}+a_{3,1})x_1x_3 + (a_{2,3}+a_{3,2})x_2x_3 \tag{23}$$

and prove that the matrix format of the gradient ∇f and Hessian H_f are correct.

Solution: Compute the partial derivatives of $f(x)$ with respect to x_1, x_2, and x_3:

$$\frac{\partial f}{\partial x_1} = 2a_{1,1}x_1 + (a_{1,2} + a_{2,1})x_2 + (a_{1,3} + a_{3,1})x_3 \tag{24}$$

$$\frac{\partial f}{\partial x_2} = (a_{1,2} + a_{2,1})x_1 + 2a_{2,2}x_2 + (a_{2,3} + a_{3,2})x_3 \tag{25}$$

$$\frac{\partial f}{\partial x_3} = (a_{1,3} + a_{3,1})x_1 + (a_{2,3} + a_{3,2})x_2 + 2a_{3,3}x_3 \tag{26}$$

Next stack Equations (24) to (26) together into a gradient vector:

$$\nabla f = \begin{bmatrix} 2a_{1,1}x_1 + (a_{1,2} + a_{2,1})x_2 + (a_{1,3} + a_{3,1})x_3 \\ (a_{1,2} + a_{2,1})x_1 + 2a_{2,2}x_2 + (a_{2,3} + a_{3,2})x_3 \\ (a_{1,3} + a_{3,1})x_1 + (a_{2,3} + a_{3,2})x_2 + 2a_{3,3}x_3 \end{bmatrix} \tag{27}$$

A gradient matrix is correct if $\nabla f(\mathbf{x}) = (A + A^T)\mathbf{x}$.

The term $(A + A^T)\mathbf{x}$ can be constructed in the following steps:

$$\begin{aligned} (A + A^T)\mathbf{x} &= \left(\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} + \begin{bmatrix} a_{1,1} & a_{2,1} & a_{3,1} \\ a_{1,2} & a_{2,2} & a_{3,2} \\ a_{1,3} & a_{2,3} & a_{3,3} \end{bmatrix} \right) \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \\ &= \begin{bmatrix} 2a_{1,1} & a_{1,2} + a_{2,1} & a_{1,3} + a_{3,1} \\ a_{2,1} + a_{1,2} & 2a_{2,2} & a_{2,3} + a_{3,2} \\ a_{3,1} + a_{1,3} & a_{3,2} + a_{2,3} & 2a_{3,3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \\ &= \begin{bmatrix} 2a_{1,1}x_1 + (a_{1,2} + a_{2,1})x_2 + (a_{1,3} + a_{3,1})x_3 \\ (a_{2,1} + a_{1,2})x_1 + 2a_{2,2}x_2 + (a_{2,3} + a_{3,2})x_3 \\ (a_{3,1} + a_{1,3})x_1 + (a_{3,2} + a_{2,3})x_2 + 2a_{3,3}x_3 \end{bmatrix} \end{aligned} \tag{28}$$

The term $\nabla f(\mathbf{x})$ is also simple to construct:

$$\begin{aligned} \nabla f(\mathbf{x}) &= \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{bmatrix} \\ &= \begin{bmatrix} 2a_{1,1}x_1 + (a_{1,2} + a_{2,1})x_2 + (a_{1,3} + a_{3,1})x_3 \\ (a_{1,2} + a_{2,1})x_1 + 2a_{2,2}x_2 + (a_{2,3} + a_{3,2})x_3 \\ (a_{1,3} + a_{3,1})x_1 + (a_{2,3} + a_{3,2})x_2 + 2a_{3,3}x_3 \end{bmatrix} \end{aligned} \tag{29}$$

Notice that Equation (28) is equivalent to Equation (29), so we have proven $\nabla f(\mathbf{x}) = (A + A^T)\mathbf{x}$.

The Hessian H_f is defined as the matrix of second-order partial derivatives:

$$H_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_1 \partial x_3} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \frac{\partial^2 f}{\partial x_2 \partial x_3} \\ \frac{\partial^2 f}{\partial x_3 \partial x_1} & \frac{\partial^2 f}{\partial x_3 \partial x_2} & \frac{\partial^2 f}{\partial x_3^2} \end{bmatrix} \tag{30}$$

The second order partial derivatives are derived from Equations (24) to (26) substituted into H_f:

$$H_f = \begin{bmatrix} 2a_{1,1} & a_{1,2} + a_{2,1} & a_{1,3} + a_{3,1} \\ a_{1,2} + a_{2,1} & 2a_{2,2} & a_{2,3} + a_{3,2} \\ a_{1,3} + a_{3,1} & a_{2,3} + a_{3,2} & 2a_{3,3} \end{bmatrix} \tag{31}$$

This matrix is also confirms $H_f = A + A^T$.

Problem 7

(Implement Team Rank Algorithm)

(a) The following table shows the head-to-head results of the B1G Conference as of Saturday Oct 25, 2025. Note the lower triangular portion of the table is omitted since a game is between two teams. Also, note that the win-loss matrix is NOT symmetric. You can update your matrix by checking https://www.espn.com/college-football/teams and select "Big Ten" when you start working on this problem.

Please design a ranking algorithm of B1G schools like the Google Page Rank Algorithm and show your computation results. Does your calculation converge? If it doesn't converge, can you explain why?

How would you adjust your algorithm to accommodate the margin of wins and losses? Does it make any difference to your ranking results?

Can you plot the social network between B1G schools, and see if any clusters emerge?

Hint: you should treat each school as a vertex and if two schools played against each other, introduce an undirected edge between them.

Solution: the ranking algorithm used is based off of the Google Page Rank Algorithm. It follows the following steps:

(1) Create a transition matrix M for a graph where each 1 or win represents an arrow from the losing team to the winning team.

(2) Normalize out of of vertex connections (sum of 1) so that it represents a probability distribution.

(3) Add a damping factor $d = 0.85$ such that there is a 15% chance that a random path is chosen.

$$M = 0.85 \cdot M_{\text{wins}} + \frac{0.15}{n} \cdot 1 \tag{32}$$

This prevents the algorithm from getting stuck in cycles and makes the matrix *irreducible*.

(4) Starting with all teams having equal rank of $1/n$, repeatedly multiply by the transition matrix:

$$\mathbf{r}_{t+1} = M \cdot \mathbf{r}_t \tag{33}$$

where $\mathbf{r}$ is the rank vector, a column vector containing the PageRank score for each team.

The algorithm does converge.

To accommodate the margin of wins and losses a margin-weighted variant where a win with a larger score difference implies a higher weight is used. A win is weighted by $\sqrt{|\text{margin}| + 1}$. Then, the transition matrix becomes:

$$M_{\text{weighted}}[j, i] = \frac{W[j, i] \cdot \sqrt{|\Delta[j, i]| + 1}}{\sum_k W[k, i] \cdot \sqrt{|\Delta[k, i]| + 1}} \tag{34}$$

where W is the win matrix and Δ is the margin matrix. Both algorithms are visualized converging in Figure 6. The final network for both algorithms is shown in Figure 7.

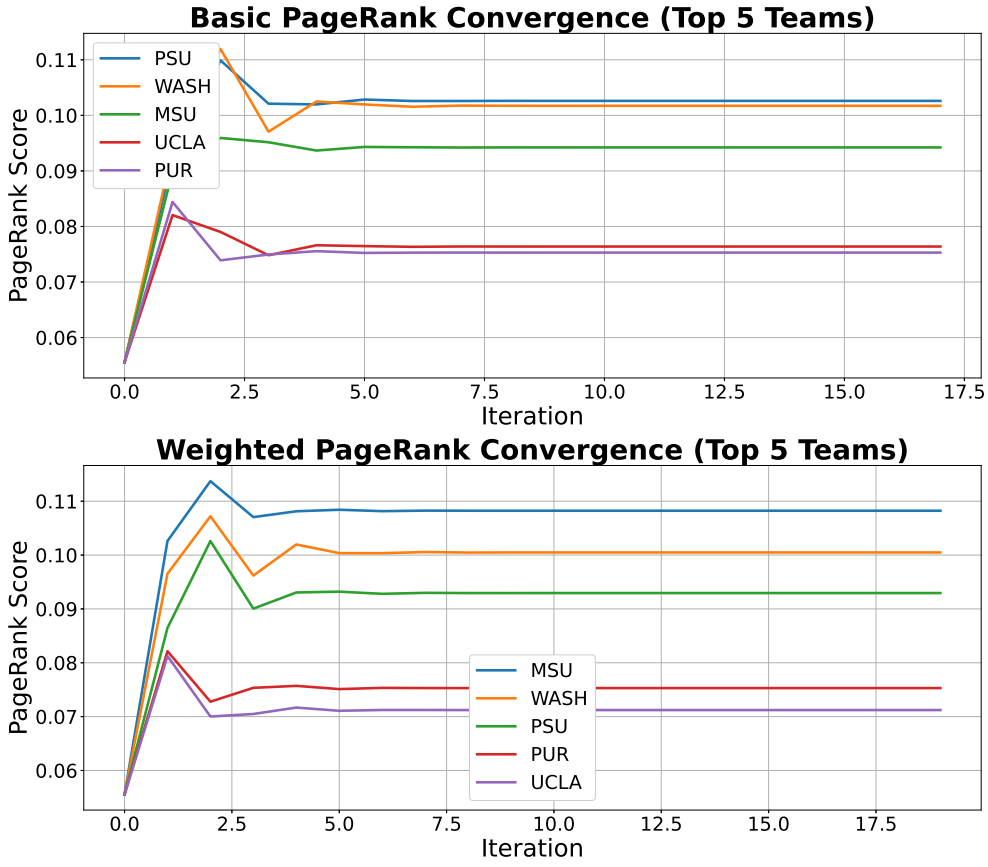

Figure 6: Algorithm convergence of the top 5 teams for the basic and weighted PageRank.

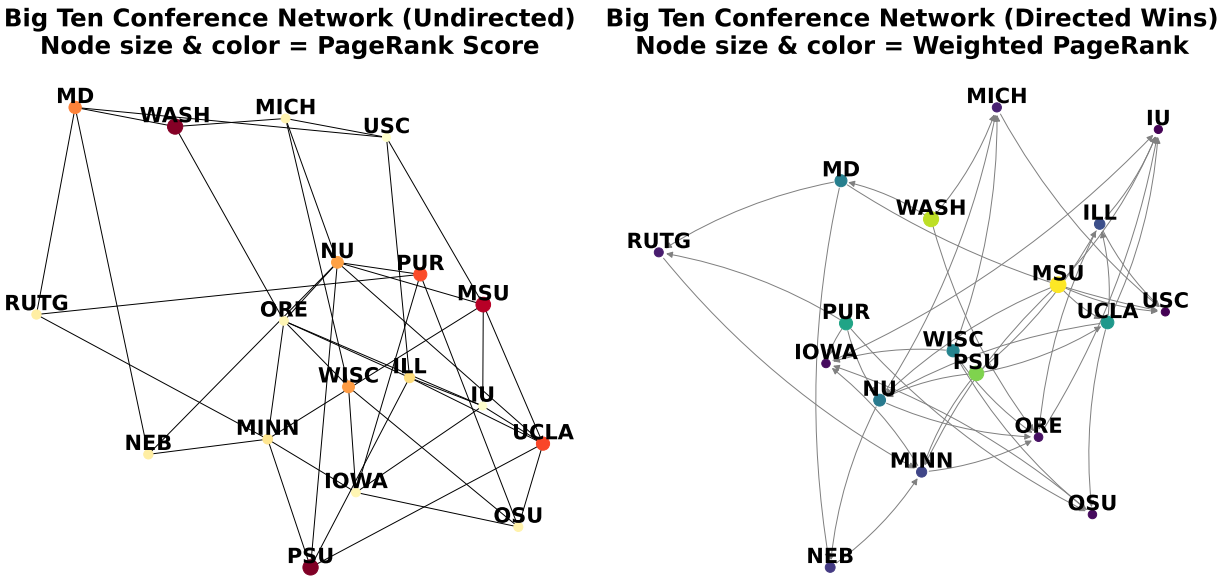

Figure 7: Network of the Big Ten.