APPM 3310 Project - College Football Ranking

Anil Ambrosi, Raaghav Lele, Siraaj Sandhu

Spring 2024

1 Abstract

The College Football Playoff is one of the most-watched sports events of the year. The 4 best teams from across the country play a head-to-head tournament to decide the champion, naming a decisive winner. However, the seeding of the tournament is quite subjective, causing much controversy. A committee of 13 representatives from around the league decides the top 25 teams, the top 4 of which get into the playoff.

We decided to use tournament ranking algorithms from 2 separate papers and compare the results with the official NCAA seeds. We then compared how successful our predictions were with the NCAA seeds.

2 Attribution

Anil worked on research and the mathematical formulation. Raaghav worked on analyzing the data. Siraaj worked on implementing the algorithms in Python. All members wrote and edited the report.

3 Mathematical Formulation

Both methods presented use a result matrix R, where each entry r_{ij} corresponds to the outcome of the games between team i and team j, with 1 representing a win, $\frac{1}{2}$ a tie, and 0 a loss. These results are summed for however many games teams i and j play against each other. A similarly defined preference matrix A may either be identical to R with scores averaged out so that each entry has the same weight, or A may represent game outcomes in some other manner. For example, if Team 1 beats Team 2 twice and ties with Team 3, while Team 2 beats Team 3, we could make a preference matrix as depicted in Figure 1. A *schedule matrix* M, also used throughout these methods, is just a matrix in which each entry (i, j) stores the number of games between the i^{th} team and the j^{th} team.

$$A = \begin{bmatrix} 0 & 1 & \frac{1}{2} \\ 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 \end{bmatrix}$$

Figure 1: Example Preference Matrix

3.1 Method 1: Eigenvector Ranking

The first method comes from "The Perron-Frobenius Theorem and the Ranking of Football Teams" (J. Keener). We will summarize Keener's proposed method below.

Suppose we wish to assign scores to a set of given teams. In effect, we want to rank them. One logical way we can define the score assigned to the i^{th} team is:

$$s_i = \frac{1}{n_i} \sum_{j=1}^{N} a_{ij} r_j \tag{3.1.1}$$

Where the i^{th} team has played n_i total games in a tournament involving N total teams. Let a_{ij} be some nonnegative number that reflects the outcome of games between the i^{th} and j^{th} team (e.g. 1 for a win, 0 for a loss, or $\frac{1}{2}$ for a tie). Let r_j be a strictly positive value that represents the strength of the j^{th} team (such that any win or tie against a stronger team is weighted more heavily in the final score calculation). We can formulate this score calculation as a matrix multiplication: let A be a matrix with entries $\frac{a_{ij}}{n_i}$ and $\mathbf{r}$ be the strength vector. Then the score vector is

$$\mathbf{s} = A\mathbf{r} \tag{3.1.2}$$

Where the score of the i^{th} team is just i^{th} component of the score vector $\mathbf{s}$. However, the score and strength vectors both measure the same thing: they both assign ranks to each team in the competition. It follows, as Keener suggests, that a given team's score is proportional to its strength:

$$\mathbf{s} = \lambda\mathbf{r} \iff A\mathbf{r} = \lambda\mathbf{r} \tag{3.1.3}$$

So, Keener concludes that the strength vector (or *rank* vector) is an eigenvector of the *preference matrix* A.

Given the nonnegative matrix A, we seek a strictly positive eigenvector $\mathbf{r}$. The paper presents the *Perron-Frobenius* theorem as a tool for determining this rank vector $\mathbf{r}$:

Theorem 3.1 (Perron-Frobenius Theorem) *A nonnegative matrix A has a positive eigenvalue λ with corresponding nonnegative eigenvector $\mathbf{r}$. Furthermore, if A is irreducible, then the eigenvalue of A with the largest magnitude (absolute value) has corresponding eigenvector $\mathbf{r}$ which is strictly positive. The matrix A is irreducible if no permutation can transform A into the following block matrix:*

$$\left[\begin{array}{c|c} B & C \\ \hline 0 & D \end{array}\right]$$

Where B and D are square matrices (i.e. the permutation creates a square matrix of zeroes in the bottom left corner of A)

Because the preference matrix is nonnegative and the rank vector we seek is positive, the Perron-Frobenius theorem establishes that a solution does indeed exist if A is irreducible. We can translate the "irreducibility" requirement on A as, equivalently, having no winless teams and not being able to partition the teams into two sets such that no teams play any teams outside of their own set. Since teams play against other teams outside of their own conferences, this problem does not arise in our own data set.

Keener suggests the method of *power iteration* to find the rank vector. Power iteration is surprisingly simple: suppose we know a matrix A has an eigenvalue that is greater in absolute value than all other eigenvalues of A. Then, the recurrence relation

$$\mathbf{r}_{k+1} = \frac{A\mathbf{r}_k}{||A\mathbf{r}_k||}, \tag{3.1.4}$$

with an initial condition $\mathbf{r}_0$, will converge on $\mathbf{r}$, the eigenvector corresponding to the aforementioned greatest eigenvalue of A (assuming $\mathbf{r}_0$ is not orthogonal to $\mathbf{r}$). The Perron-Frobenius theorem assures us that our preference matrix has an eigenvalue that is strictly greater than its other eigenvalues, and thus the rank vector may be obtained via power iteration. Thus,

$$\mathbf{r} = \lim_{n\to\infty} \frac{A^n\mathbf{r}_0}{||A^n\mathbf{r}_0||} \tag{3.1.5}$$

As of now, the matrix methods presented are sufficient to find *a* ranking of NCAA teams. However, with some tweaking, we can make the rankings more reflective of each team in the tournament, and in a sense, more "accurate."

Firstly, Keener points out that the initial definition of the preference matrix A is naive in the sense that assigning each a_{ij} a discrete value of 0, 1, or $\frac{1}{2}$ based on the result of a game does not necessarily reflect the strength of each team in the interaction in the most accurate way possible. Although this scheme is intuitive, assigning 0 to the loser can entirely dismiss the effort they put in, especially if the game was a close victory for the winning team. Moreover, if a there *is* a winless team, assigning them 0 for every one of their games will make the matrix reducible, even if they did win some points in each individual game. A more reasonable definition may be:

$$a_{ij} = \frac{S_i}{S_i + S_j} \tag{3.1.6}$$

Where a_{ij} is the preference value for team i in their game against team j, if the i^{th} team scored S_i points in the encounter and the j^{th} team scored S_j points. With this method, points in the preference matrix are distributed continuously, rather than discretely. Furthermore, we can protect against reducibility with this new scheme and avoid the case where the rank vector is not strictly positive, thereby making our method more robust.

There are additional considerations, however: suppose a pair of evenly-matched teams struggle to score any points for an entire game, with one team winning by a slim margin. If the end result is 3-0, for instance, the losing team gets no credit. We can modify each a_{ij} slightly to account for this:

$$a_{ij} = \frac{S_i + 1}{S_i + S_j + 2} \tag{3.1.7}$$

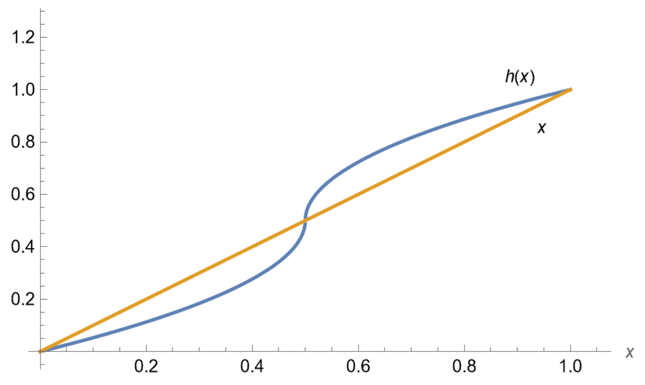


Figure 2: Applying the nonlinear $h(x)$ transformation to a_{ij} values can de-emphasize the importance of scoring points. Observe how the preference value increases more rapidly at $x = \frac{1}{2}$ and then starts to plateau. This creates diminishing returns for good teams that want to improve their ranking by beating poor teams.

The new scheme also prioritizes points rather than outcomes, which can be disadvantageous in the sense that a very good team can simply run up its score in a game against a worse team to improve its ranking. we could pass each preference value through a nonlinear function that slightly de-emphasizes points:

$$\text{Let } h(x) = \frac{1}{2} + \frac{1}{2}\,\text{sign}\left(x - \frac{1}{2}\right)\sqrt{|2x-1|} \tag{3.1.8}$$

$$\text{Then } a_{ij} = h\left(\frac{S_i + 1}{S_i + S_j + 2}\right) \tag{3.1.9}$$

The effect of this transformation is visualized in Figure 2. Keener also proposes an alternative, nonlinear ranking scheme:

$$r_i = \frac{1}{n_i}\sum_{j=1}^{N} f(e_{ij} r_j) \tag{3.1.10}$$

Where e_{ij} is analogous to a_{ij}, as it is reflective of the outcome of games between team i and team j. Once again, r is the rank or strength value. The function f is defined such that $f(0) = 0$ and $\lim_{x\to\infty} f(x) = 1$. The effect is that strong teams with weak schedules, that would be disadvantaged because they face poor teams and thus get ranked lower, can earn more points per game (a maximum of 1) by winning against bad teams.

If we reformulate this as a transformation of $\mathbf{r}$, we get the nonlinear transformation F:

$$F_i(\mathbf{r}) = \frac{1}{n_i}\sum_{j=1}^{N} f(e_{ij} r_j) \tag{3.1.11}$$

Keener goes on to prove that if we further impose the requirement that $f(0) > 0$, the space of strictly positive $\mathbf{r} \in \mathbb{R}^n_+$ is invariant under the transformation F, and consequently, our solution $\mathbf{r}$ is a unique *fixed point* of the transformation F. Therefore, the ranking vector $\mathbf{r}$

can be found via *fixed point iteration*. A fixed point remains unchanged after undergoing a given transformation: $F(\mathbf{r}) = \mathbf{r}$. The method of fixed point iteration, like power iteration, is simple: the recurrence relation

$$\mathbf{r}_{k+1} = F(\mathbf{r}_k), \tag{3.1.12}$$

with an initial condition $\mathbf{r}_0$ in the domain upon which F operates (in this case, strictly positive $\mathbf{r}$), will converge on the fixed point of F. Thus,

$$\mathbf{r} = \lim_{n\to\infty} F^n(\mathbf{r}_0), \tag{3.1.13}$$

We decided to use Keener's suggested definitions of f and e_{ij}, defined below:

$$f(x) = \frac{.05x + x^2}{2 + .05x + x^2}$$
$$e_{ij} = \frac{5 + S_i + S_i^{\frac{2}{3}}}{5 + S_j + S_i^{\frac{2}{3}}}$$

For our purposes, we decided to use every definition of a_{ij}, as well as e_{ij} from the nonlinear scheme, to find our ranking vector $\mathbf{r}$ and see what the precise difference in ranking might be.

3.2 Method 2: Probability Ranking

The second method comes from the paper "The Ranking of Incomplete Tournaments: A Mathematician's Guide to Popular Sports" (T. Jech). This method relies on adding up the results to get a "score" from which we can calculate the probability that a certain team will beat another. For any two teams T_i and T_j, the probability p_{ij} and odds x_{ij} that T_i will win are given by

$$p_{ij} = \frac{r_{ij}}{m_{ij}} \tag{3.2.1}$$

$$x_{ij} = \frac{r_{ij}}{m_{ij} - r_{ij}} \tag{3.2.2}$$

where r is the number of games T_i won against T_j and m is the total number of games played between the two teams. Observe that these r_{ij} and m_{ij} are just elements of the result matrix R and schedule matrix M, respectively.

Doing some algebra, we can relate x_{ij} and p_{ij} with the following equations:

$$x_{ij} = \frac{p_{ij}}{1 - p_{ij}} \tag{3.2.3}$$

$$p_{ij} = \frac{x_{ij}}{1 + x_{ij}} \tag{3.2.4}$$

The *rank matrix* P filled with entries p_{ij} thus gives us an estimate for how each team would fare with another, from which we can calculate our ranking. If $x_{ij} > 1$ or $p_{ij} > 0.5$, then T_i is expected to beat T_j However, this requires some generalization to our case because not every team played each other.

As these are supposed to represent probabilities and odds, we will impose the following conditions on x and p

$$x_{ik} = x_{ij} \cdot x_{jk} \tag{3.2.5}$$

$$p_{ij} \cdot p_{jk} \cdot p_{ki} = p_{kj} \cdot p_{ji} \cdot p_{ik} \tag{3.2.6}$$

In addition, since p signifies the probability that a team wins a game, multiplying it by the number of games each team plays should result in their number of wins (that team's raw score). This results in the equation

$$\sum_{j=1}^{n} m_{ij} \cdot p_{ij} = s_i \tag{3.2.7}$$

$$MP^T = S,\ S_{ii} = s_i \tag{3.2.8}$$

Where n is the number of teams in the tournament and s_i is the score of T_i. To avoid confusion with its usage in the first paper, T_i's "score" in the context of Jech's paper is just the sum of its results in the result matrix: $s_i = \sum_{j=1}^{n} r_{ij}$. So, finding the ranking for the tournament ultimately relies on finding each p_{ij} that satisfies equations (6) and (7). The method for solving this system is outlined in a lemma presented by Jech:

Lemma: *Let R be a result mtarix, M be the corresponding schedule matrix, and $\mathbf{s} = (s_i, \ldots, s_n)^T$ be the score vector. There exists real numbers $v_1, ..., v_n$ which satisfy the following system of equations for $i = 1, ..., n$:*

$$\sum_{j=1}^{n} \frac{m_{ij}}{1 + e^{v_j - v_i}} = s_i \tag{3.2.9}$$

Observe that if we let each $p_{ij} = \frac{e^{v_i}}{e^{v_i}+e^{v_j}} = \frac{1}{1+e^{v_j - v_i}}$, this equation is equivalent to (3.2.7). Thus we are just representing the probabilities in a different form. Observe that $p_{ji} = 1 - p_{ij} = \frac{1}{1+e^{v_i - v_j}}$, making this a suitable representation as it follows the laws for probabilities outlined previously. Thus, this representation can be used to solve (3.2.6) and (3.2.7), and therefore find a ranking for the tournament R. Now, we can prove the Lemma.

Proof of Lemma. Let V be the set of all n-tuples $\mathbf{v} = (v_1, ..., v_n)$ such that $v_1 + \cdots + v_n = 0$. For naming purposes, let us deem this space V as the "space of probability vectors $\mathbf{v}$" (as each $\mathbf{v}$ can be seen as an alternate encoding for p_{ij}).

V is just a closed subspace of $\mathbb{R}^n$: $V \subseteq \mathbb{R}^n$. For each $\mathbf{v} \in V$, let $F(\mathbf{v})$ be the left-hand side of (3.2.9). That is, let $F(\mathbf{v}) = (F_1(\mathbf{v}), ..., F_n(\mathbf{v}))$ where

$$F_i(\mathbf{v}) = \sum_{j=1}^{n} \frac{m_{ij}}{1 + e^{v_j - v_i}} \text{ for } (i = 1, ..., n) \tag{3.2.10}$$

And for each $\mathbf{v} \in V$, we let the nonlinear transformation T be:

$$T(\mathbf{v}) = \mathbf{v} + \mathbf{s} - F(\mathbf{v}) \tag{3.2.11}$$

Where $\mathbf{s}$ is the score vector.

If we let m be the total number of games played in the tournament, $m = \frac{1}{2}\sum_{i,j} m_{ij}$, note that $\sum_{i=1}^n s_i = m$ and that for each $\mathbf{v} \in V$,

$$\sum_{i=1}^n F_i(\mathbf{v}) = \sum_{i,j}^n \frac{m_{ij}}{1+e^{v_j - v_i}} = \frac{1}{2}\sum_{i,j}^n m_{ij}\left(\frac{e^{v_i}}{e^{v_i}+e^{v_j}} + \frac{e^{v_j}}{e^{v_i}+e^{v_j}}\right) = \frac{1}{2}\sum_{i,j}^n m_{ij} = m. \tag{3.2.12}$$

The consequence of this is that summing over the components of $T(\mathbf{v})$ for any $\mathbf{v} \in V$ produces zero, because $\sum_{i=1}^n s_i = m$ and $\sum_{i=1}^n F_i(\mathbf{v}) = m$, so summing the elements of their difference will produce zero. Thus, $\sum_{i=1}^n T(\mathbf{v})_i = \sum_{i=1}^n v_i = 0$. It follows that $T(\mathbf{v}) \in V$. So, V is invariant under the transformation T and, as Jech asserts, T is continuous. To find the solution of (3.2.9), we only need to find a fixed point of T, i.e., some $\mathbf{v} \in V$ such that

$$T(\mathbf{v}) = \mathbf{v}. \tag{3.2.13}$$

If $\mathbf{v}$ is a fixed point, then $\mathbf{s} - F(\mathbf{v}) = 0 \iff F(\mathbf{v}) = \mathbf{s}$ and therefore $\mathbf{v}$ must be a solution of (3.2.9) and by extension (3.2.7). Jech goes on to prove that T does indeed possess a fixed point. As with Keener's nonlinear transformation F, we can find the fixed point $\mathbf{v}$ of T via fixed point iteration. We can then use that $\mathbf{v}$ to find the rank matrix P, which encodes the probabilities that any one team will beat another. Even if a team i didn't play against some other team j, Jech suggests that we can treat their p_{ij} ranking as a sort of *prediction* of the probability that team i will beat team j. While this provides a means to compare two teams directly, we still want to produce a "net" ranking of all teams. Jech proposes a few ways to do this, but we chose his *weighted percentage* method:

$$p_i = \frac{1}{n-1}\sum_{j=1, j\neq i}^n p_{ij}. \tag{3.2.14}$$

In effect, we sum the winning probabilities (actual and predicted) of team i against every other team j. We can use this resulting p_i to directly rank every team in the tournament, and thus our ranking vector $\mathbf{r} = (p_1, \ldots, p_n)^T$.

4 Examples and Numerical Results

Both methods described above rely first and foremost on game data, particularly the specific team match-ups and points scored by each team. For this project, we decided to focus on the 2023 NCAA college football season. Luckily, Sports Reference (`https://www.sports-reference.com/cfb/years/2023-schedule.html`) makes game-by-game data from the latest season available to export as a CSV file. with some modifications to the data, the CSV we use (viewable here: `https://github.com/anilambrosi/APPM-3310-Project/blob/main/data/games.csv`) has the following structure:

Game	i^{th} **team**	i^{th} **pts.**	j^{th} **team**	j^{th} **pts.**
1	Jacksonville State	17	Texas-El Paso	14
2	Louisiana Tech	22	Florida International	17
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
868	Army	17	Navy	11

For our purposes we decided to use Python for its simplicity and because it would give us access to powerful libraries like `numpy` and `matplotlib`. Before going further, it will be helpful to standardize naming conventions for variables and concepts that are used across both papers:

Variable	**Meaning**
`M`	Schedule Matrix, M
`R`	Result Matrix, R
`A`	Preference Matrix, A
`E`	Nonlinear Preference Matrix (e_{ij} from Keener, 1993)
`P`	Rank Matrix, P (from Jech, 1983)
`r`	Rank/Strength Vector, $\mathbf{r}$
`s`	Score Vector, $\mathbf{s}$
`V`	Basis for probability vector space (Jech, 1983), V
`v`	Element of the image of `V`, $\mathbf{v} \in \text{img } V$

Much, if not all, of the code is dedicated to computing these particular matrices and vectors. Construction of the schedule and preference matrices is trivial. Python's built-in `csv` package allows us to load in our `games.csv` file, and iterating over each game entry, we append each team name to a list and remove duplicates. This provides us with a mapping from team names to their index. Looping over games once again, we simply find integers `i` and `j` that correspond to the indices of the i^{th} and j^{th} teams and use these to access the $(i, j)^{\text{th}}$ entry of the `M`, `R`, `A`, and `E` matrices, which are initialized as zero-filled `numpy` `ndarray`'s. Values for each are calculated according to the papers (see Figure 3 for what an example preference matrix might look like):

$$\begin{aligned}
&\texttt{M[i, j]} = \text{total number of games between teams } i \text{ and } j \\
&\texttt{R[i, j]} = 0 \ (i^{\text{th}} \text{ lost}),\ 1 \ (i^{\text{th}} \text{ won}), \text{ or } \frac{1}{2} \ (\text{tie}) \\
&\texttt{A[i, j]} = a_{ij} \\
&\texttt{E[i, j]} = e_{ij}
\end{aligned}$$

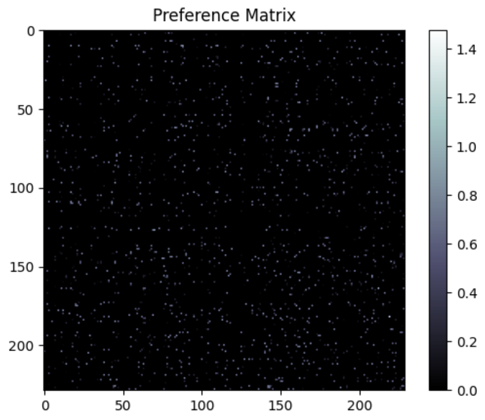


Figure 3: Example preference matrix `A` visualized. The rows have not been normalized (divided by n_i) yet so as to better highlight the sparse data points (otherwise, the points would appear dim). Observe that most of the entries are zero, illustrating how ranking teams can be difficult when the tournament is not round-robin. Observe also how the entries, or "games," seem to be nicely spread across the entire matrix, suggesting, at least visually, that `A` is irreducible. Nevertheless, we pre-screened our data to ensure that our preference matrices would in fact be irreducible.

To carry out power iteration, we generate a random positive seed vector $\mathbf{r}_0$ via `np.random.rand` and repeatedly multiply it by `A` and normalize it for however many iterations we wish to carry out (we chose 100). The result is an (approximate) rank vector which we can then use to directly rank the teams. The results are presented below, in Figure 5.

For the nonlinear scheme, we can look back to the definition of F:

$$F_i(\mathbf{r}) = \frac{1}{n_i} \sum_{j=1}^{N} f(e_{ij} r_j)$$

With our `E` matrix, each $e_{ij} r_j$ multiplication can be encapsulated in the matrix multiplication $E\mathbf{r}_k$, and we can use `numpy` to vectorize $f(x)$ and apply it to the entire product $E\mathbf{r}_k$: `f(E @ r)`. Like with power iteration, we generate a random positive seed vector $\mathbf{r}_0$ and apply the F transformation an arbitrary amount of times to find its (approximate) fixed point. The results are also presented below, in Figure 5.

The second method uses only the result matrix `R` and the schedule matrix `M`. We want to use fixed point iteration to find the fixed point of T:

$$T(\mathbf{v}) = \mathbf{v} + \mathbf{s} - F(\mathbf{v})$$

It follows from Jech's lemma that if we want to find a solution, we need to produce a seed vector $\mathbf{v}_0$ in the set V of all n-tuples $\mathbf{v} = (v_1, \ldots, v_n)$ such that

$$v_1 + \cdots + v_n = 0.$$

In other words, we need a basis for V, which we can obtain by solving the system: $v_1 + v_2 + \cdots + v_n = 0$. So, the basis would be:

$$\mathtt{V} = \begin{bmatrix} -1 & -1 & -1 & \dots & -1 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

Creating this matrix with `numpy` is straightforward, and thus we can obtain an initial $\mathbf{v}_0$ by generating random coordinates in V: `v = V @ np.random.rand(V.shape[1])`. We can then repeatedly apply the T transformation to $\mathbf{v}_0$ to obtain our (approximate) fixed point $\mathbf{v}$. To get our rank matrix P, we use the definition $p_{ij} = \frac{1}{1+e^{v_j - v_i}}$. Figure 4 provides a visualization of what P could look like. With this P in hand, can find $\mathbf{r}$ via the weighted percentage method. The results are presented in Figure 5 below, with a comparison between our methods and the official rankings in Figure 6. Because presenting full rankings (of over 200 teams) would take up too much space, we have chosen to include only the top 15 teams produced by each method. Nevertheless, each algorithm is indeed ranking every team in the dataset and operating on square matrices of dimension well over 200.

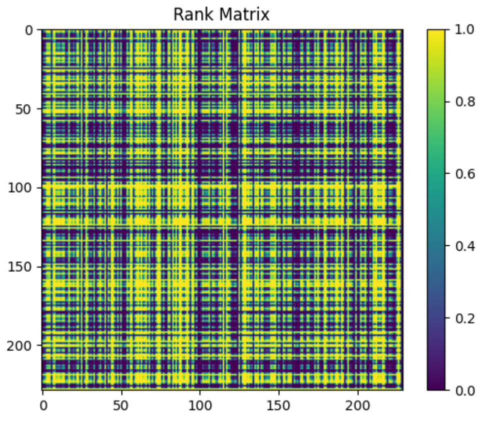
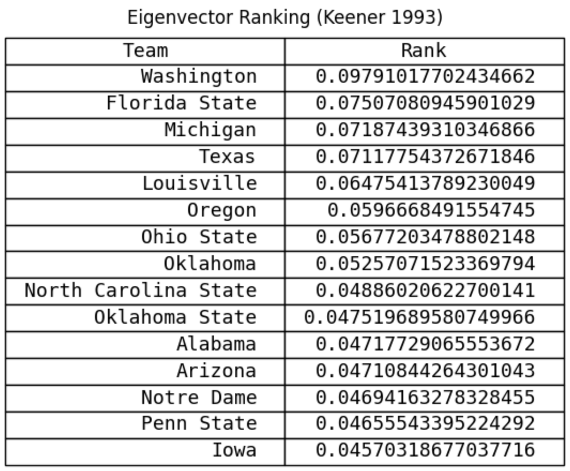


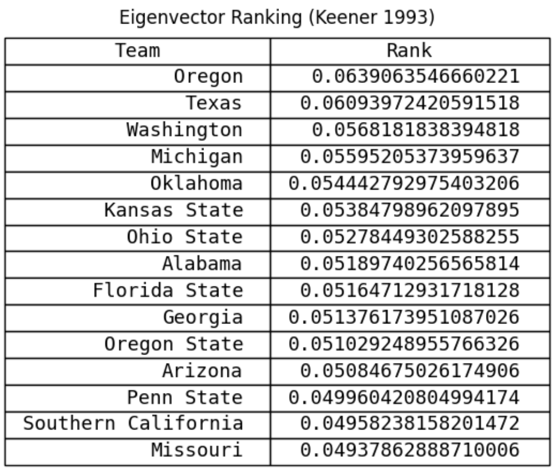
Figure 4: Example rank matrix visualized. Because entries of this matrix are probabilities, the maximum value is 1.



Eigenvector Ranking (Keener 1993)

Team	Rank
Washington	0.09791017702434662
Florida State	0.07507080945901029
Michigan	0.07187439310346866
Texas	0.07117754372671846
Louisville	0.06475413789230049
Oregon	0.0596668491554745
Ohio State	0.05677203478802148
Oklahoma	0.05257071523369794
North Carolina State	0.04886020622700141
Oklahoma State	0.047519689580749966
Alabama	0.04717729065553672
Arizona	0.04710844264301043
Notre Dame	0.04694163278328455
Penn State	0.04655543395224292
Iowa	0.04570318677037716

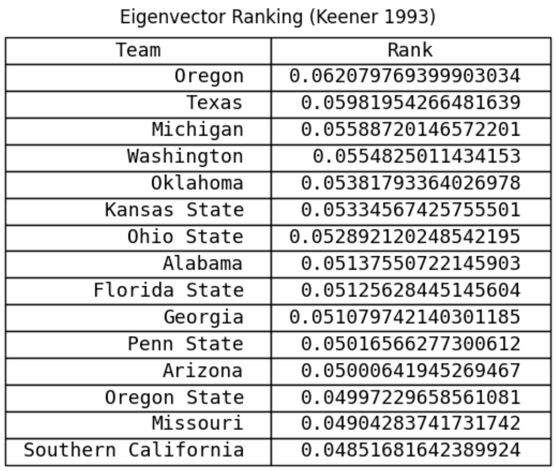
(a) Discrete a_{ij}



Eigenvector Ranking (Keener 1993)

Team	Rank
Oregon	0.0639063546660221
Texas	0.06093972420591518
Washington	0.0568181838394818
Michigan	0.05595205373959637
Oklahoma	0.054442792975403206
Kansas State	0.05384798962097895
Ohio State	0.05278449302588255
Alabama	0.05189740256565814
Florida State	0.05164712931718128
Georgia	0.051376173951087026
Oregon State	0.051029248955766326
Arizona	0.05084675026174906
Penn State	0.049960420804994174
Southern California	0.04958238158201472
Missouri	0.04937862888710006

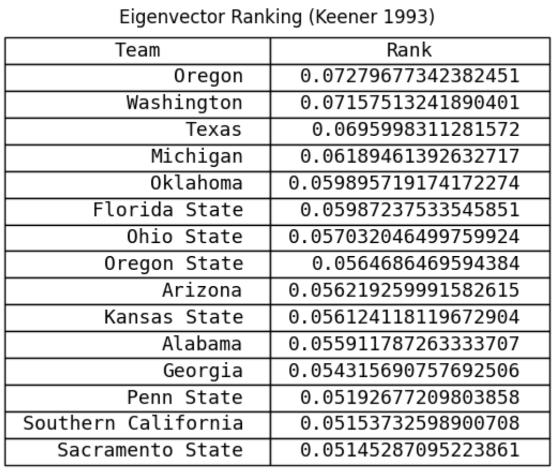
(b) $a_{ij} = \frac{S_i}{S_i+S_j}$



Eigenvector Ranking (Keener 1993)

Team	Rank
Oregon	0.062079769399903034
Texas	0.05981954266481639
Michigan	0.05588720146572201
Washington	0.0554825011434153
Oklahoma	0.05381793364026978
Kansas State	0.05334567425755501
Ohio State	0.052892120248542195
Alabama	0.05137550722145903
Florida State	0.05125628445145604
Georgia	0.051079742140301185
Penn State	0.05016566277300612
Arizona	0.05000641945269467
Oregon State	0.04997229658561081
Missouri	0.04904283741731742
Southern California	0.04851681642389924

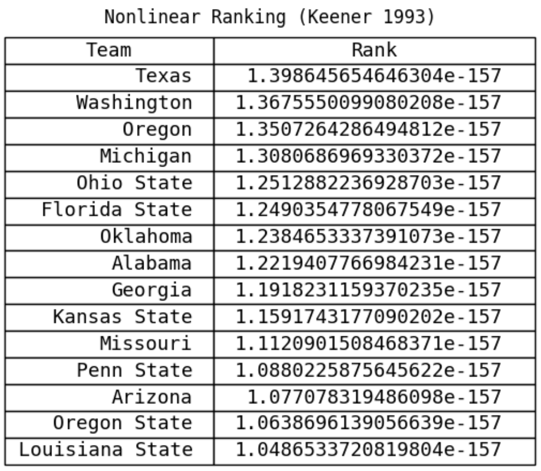
(c) $a_{ij} = \frac{S_i+1}{S_i+S_j+2}$



Eigenvector Ranking (Keener 1993)

Team	Rank
Oregon	0.07279677342382451
Washington	0.07157513241890401
Texas	0.0695998311281572
Michigan	0.06189461392632717
Oklahoma	0.059895719174172274
Florida State	0.05987237533545851
Ohio State	0.057032046499759924
Oregon State	0.0564686469594384
Arizona	0.056219259991582615
Kansas State	0.056124118119672904
Alabama	0.055911787263333707
Georgia	0.054315690757692506
Penn State	0.05192677209803858
Southern California	0.05153732598900708
Sacramento State	0.05145287095223861

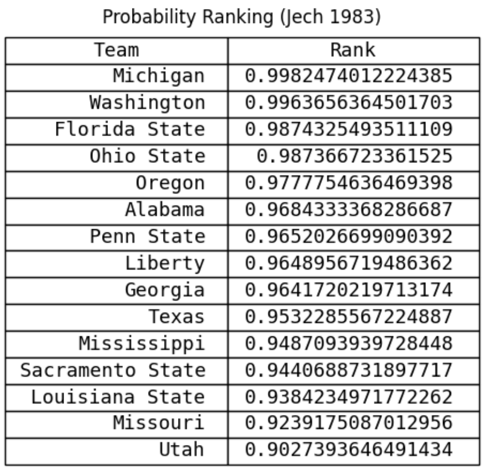
(d) $a_{ij} = h(\frac{S_i+1}{S_i+S_j+2})$



Nonlinear Ranking (Keener 1993)

Team	Rank
Texas	1.398645654646304e-157
Washington	1.3675550099080208e-157
Oregon	1.3507264286494812e-157
Michigan	1.3080686969330372e-157
Ohio State	1.2512882236928703e-157
Florida State	1.2490354778067549e-157
Oklahoma	1.2384653337391073e-157
Alabama	1.2219407766984231e-157
Georgia	1.1918231159370235e-157
Kansas State	1.1591743177090202e-157
Missouri	1.1120901508468371e-157
Penn State	1.0880225875645622e-157
Arizona	1.077078319486098e-157
Oregon State	1.0638696139056639e-157
Louisiana State	1.0486533720819804e-157

(e) Nonlinear Ranking ($F(\mathbf{r})$)

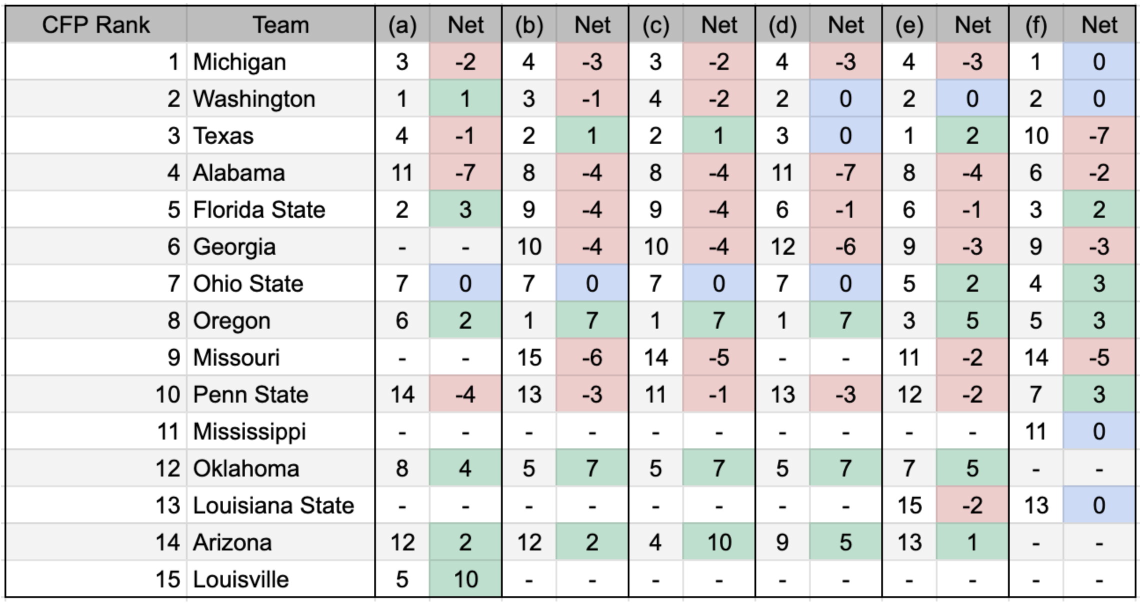


Probability Ranking (Jech 1983)

Team	Rank
Michigan	0.9982474012224385
Washington	0.9963656364501703
Florida State	0.9874325493511109
Ohio State	0.987366723361525
Oregon	0.9777754636469398
Alabama	0.9684333368286687
Penn State	0.9652026699090392
Liberty	0.9648956719486362
Georgia	0.9641720219713174
Texas	0.9532285567224887
Mississippi	0.9487093939728448
Sacramento State	0.9440688731897717
Louisiana State	0.9384234971772262
Missouri	0.9239175087012956
Utah	0.9027393646491434

(f) Probability Ranking

Figure 5: Rankings for the 2023 NCAA season. Ranks are just elements of the rank vector $\mathbf{r}$ after being sorted.



CFP Rank	Team	(a)	Net	(b)	Net	(c)	Net	(d)	Net	(e)	Net	(f)	Net
1	Michigan	3	-2	4	-3	3	-2	4	-3	4	-3	1	0
2	Washington	1	1	3	-1	4	-2	2	0	2	0	2	0
3	Texas	4	-1	2	1	2	1	3	0	1	2	10	-7
4	Alabama	11	-7	8	-4	8	-4	11	-7	8	-4	6	-2
5	Florida State	2	3	9	-4	9	-4	6	-1	6	-1	3	2
6	Georgia	-	-	10	-4	10	-4	12	-6	9	-3	9	-3
7	Ohio State	7	0	7	0	7	0	7	0	5	2	4	3
8	Oregon	6	2	1	7	1	7	1	7	3	5	5	3
9	Missouri	-	-	15	-6	14	-5	-	-	11	-2	14	-5
10	Penn State	14	-4	13	-3	11	-1	13	-3	12	-2	7	3
11	Mississippi	-	-	-	-	-	-	-	-	-	-	11	0
12	Oklahoma	8	4	5	7	5	7	5	7	7	5	-	-
13	Louisiana State	-	-	-	-	-	-	-	-	15	-2	13	0
14	Arizona	12	2	12	2	4	10	9	5	13	1	-	-
15	Louisville	5	10	-	-	-	-	-	-	-	-	-	-

Figure 6: Table comparing the results of the different methods with the official CFP ranking

5 Conclusions

The College Football Rankings have proven to be controversial every year as the ranking of teams is mainly subjective and restricted to the Ranking Committee. Using methodology from two separate papers, we ranked the top four teams of the 2023 season. We modified several functions to account for close games, low-scoring games, and the difference of strength between two teams facing each other. Using Python, we used six different methods across two papers to create six separate rankings. We found Texas, Washington, Oregon, Michigan, Florida State, and Ohio State to be ranked in the top four in at least one iteration, with the first four schools being consistently ranked in all iterations. The actual ranking from the CFP Committee was (1) Michigan, (2) Washington, (3) Texas, and (4) Alabama. The ranking most consistent with the CFP ranking was (d) since it correctly ranked Washington and Texas, and included Michigan.

Two problems that arose from our results were that Alabama was nowhere near the top four and that most of the right teams were picked, but their order was massively inaccurate for each method used. Alabama being left out of the rankings can be attributed to the number of close games they had which would hurt them in all the Keener rankings, and their early season loss to Texas which would hurt them in the Probability Ranking since they had already lost to a future top-four team. In addition, despite being a very solid team who were championship favorites most of the time, Georgia is ranked quite low using our methods, perhaps due to the same reasons as Alabama.

Interestingly enough, some of our methods put Florida State below their official rank of 5, despite this being a great source of controversy upon the rankings' release, with many thinking that they were left out of the playoffs.

Next year, a 12-team playoff will be implemented, which will offer a better proving ground to test our rankings with the CFP committee's, as there will be a decisive method of determining the stronger teams. The 12-team format will also have guaranteed entries for the top 5 conference champions, so our ranking will be a way to measure how strong the champions are without weighting one game over others. In the future, we would re-rank the teams and see how our rankings fared in the playoff compared to the official seed.

6 References

Jech, T. (1983). The Ranking of Incomplete Tournaments: A Mathematician's Guide to Popular Sports. *The American Mathematical Monthly*, 90(4), 246–266. https://doi.org/10.2307/2975756

Keener, J. P. (1993). The Perron-Frobenius Theorem and the Ranking of Football Teams. *SIAM Review*, 35(1), 80–93. http://www.jstor.org/stable/2132526

Sports Reference. (2024, January 8). *2023 college football schedule and results.* sports-reference.com. https://www.sports-reference.com/cfb/years/2023-schedule.html

College Football Playoff. (2023, December 3). *College football playoff selection committee announces final top 25 rankings of 2023.* https://collegefootballplayoff.com/news/2023/12/3/cfp-rankings-2023-1203.aspx

7 Appendix

7.1 Data

The CSV data we used may be accessed at `https://github.com/anilambrosi/APPM-3310-Project/blob/main/data/games.csv`

7.2 Code

```python
# utils for cleaning/loading data

import re
import numpy as np
import matplotlib.pyplot as plt
from csv import reader as csv_reader
from io import StringIO
from functools import cmp_to_key
from math import copysign, sqrt, fabs, pi
from enum import Enum
from collections.abc import Callable

"""
naming conventions:
  M => schedule matrix
  R => result matrix
  A => preference matrix
  E => (nonlinear) preference matrix
  P => rank matrix described in Jech, 1983
  r => rank vector
  s => score vector
  T => nonlinear transformation described in Jech, 1983
  v => probability vector described in Jech, 1983
  V => matrix whose column space spans the probability vector space described in Jech, 1983
"""

# TYPES

type mat_f32 = np.ndarray[np.float32]
type vec_f32 = np.ndarray[np.float32]

# CSV

class Csv:
  def __init__(self, path: str):
    with open(path, mode='r') as file:
      text = file.read() # remove seed from team names
      clean_text = re.sub(r'\([0-9]+\)\s*', '', text)
      csv_file = csv_reader(StringIO(clean_text))
      next(csv_file) # remove header
      self.data = [line for line in csv_file if all(entry != '' for entry in line)]

# PREFERENCE / RESULT MATRICES

calc_pref_interpolate = lambda si, sj: si / (si + sj) # find a_{ij}
calc_pref_interpolate_offset = lambda si, sj: calc_pref_interpolate(si + 1, sj + 1) # find a_{ij}

def calc_pref_discrete(si: int, sj: int) -> float: # find a_{ij}
  if si == sj:
    return 0.5
  elif si > sj:
    return 1
  return 0

def calc_pref_nonlinear(si: int, sj: int) -> float: # find a_{ij}
  x = calc_pref_interpolate_offset(si, sj)
  return 0.5 + 0.5 * copysign(1, x - 0.5) * sqrt(fabs(2 * x - 1))

class PrefMethod(Enum):
  DISCRETE = 'discrete' # DISCRETE -> result matrix (R)
  INTERPOLATE = 'interpoate'
  INTERPOLATE_OFFSET = 'interpolate_offset'
  NONLINEAR = 'nonlinear'

PREF_METHODS = {
  PrefMethod.DISCRETE: calc_pref_discrete,
  PrefMethod.INTERPOLATE: calc_pref_interpolate,
  PrefMethod.INTERPOLATE_OFFSET: calc_pref_interpolate_offset,
  PrefMethod.NONLINEAR: calc_pref_nonlinear
}

def make_pref_matrix(csv: Csv, pref_method: PrefMethod) -> tuple[list[str], mat_f32, mat_f32, mat_f32, mat_f32]:
  teams = list(set(line[0] for line in csv.data) |
               set(line[2] for line in csv.data))
  num_teams = len(teams)

  M = np.zeros((num_teams, num_teams), dtype=np.float32) # schedule matrix
  R = np.zeros((num_teams, num_teams), dtype=np.float32) # result matrix
  E = np.zeros((num_teams, num_teams), dtype=np.float32) # nonlinear
   preference matrix

  calc_a = PREF_METHODS[pref_method] # find each A_ij
  calc_e = lambda si, sj: (5 + si + si**(2/3)) / (5 + sj + si**(2/3)) #
   find each E_ij

  for line in csv.data:
    iname, si, jname, sj = line

    i = teams.index(iname)
    j = teams.index(jname)
    si = int(si)
    sj = int(sj)

    M[i, j] += 1
    M[j, i] += 1
    R[i, j] += calc_a(si, sj)
    R[j, i] += calc_a(sj, si)
    E[i, j] += calc_e(si, sj)
    E[j ,i] += calc_e(sj, si)

  A = np.copy(R) # (normalized) preference matrix
  games_played = M @ np.ones(M.shape[1])
  for i, _ in enumerate(teams):
    A[i, :] /= games_played[i]

  return (teams, R, M, A, E)

# MATRIX METHODS

# approximate eigenvector (rank vector) correlating with largest
   eigenvalue  of A
def power_method(A: mat_f32, num_ierations: int) -> vec_f32:
  r = np.random.rand(A.shape[1])

  for _ in range(num_ierations):
    r = A @ r # multiply r by A
    r /= np.linalg.norm(r) # normalize the result

  return r

def power_method_nonlinear(E: mat_f32, M: mat_f32, f: Callable[[float],
   float], num_iterations: int) -> vec_f32:
  r = np.random.rand(E.shape[1])
  f_vec = np.vectorize(f)
  games_played = M @ np.ones(M.shape[1])

  # repeatedly apply the F transformation to approximate its fixed point
  for _ in range(num_iterations):
    r = np.divide(f_vec(E @ r), games_played)

  return r

def sort_by_rank(teams: list[str], r: vec_f32) -> list[tuple[str, float]]:
  couple = zip(teams, r)
  return sorted(couple, key=cmp_to_key(lambda a, b: b[1] - a[1]))

# find rank matrix P given a probability vector v
def make_rank_matrix(v: vec_f32) -> mat_f32:
  V = np.repeat([v], repeats=[v.shape[0]], axis=0)
  return np.power(1 + np.exp(V - np.transpose(V)), -1)

# create a basis V for the probability vector space
def make_v_basis(M: mat_f32) -> mat_f32:
  n = M.shape[0] - 1
  top = np.full(n, -1, dtype=np.float32)
  return np.vstack([top, np.eye(n)])

# apply the nonlinear T transformation to a probability vector v
def apply_T(v: vec_f32, s: vec_f32, M: mat_f32) -> vec_f32:
  P = make_rank_matrix(v)
  F = np.dot(np.multiply(M, P), np.ones_like(v))
  return v + s - F

# approximate the fixed point of T
def fixed_point_approx(R: mat_f32, M: mat_f32, num_iterations: int) ->
    vec_f32:
  V = make_v_basis(M)
  v = V @ np.random.rand(V.shape[1])
  s = R @ np.ones_like(v)

  for _ in range(num_iterations):
    v = apply_T(v, s, M)

  return v

# display teams and ranks in a table
def disp_ranks(team_ranks: list[tuple[str, float]], title: str):
  fig, ax = plt.subplots()
  plt.rcParams['font.family'] = 'monospace'
  table = ax.table(
    cellText = np.asarray(team_ranks[:15]),
    loc = 'center',
    colLabels = ['Team', 'Rank'])
  table.scale(1, 1.5)
  table.set_fontsize(13)
  table.auto_set_column_width(col=[0,1])
  table.auto_set_font_size(False)
  ax.axis('off')
  ax.set_title(title, pad=20)
  fig.tight_layout()
  fig.set_size_inches(5.25, 4.5)
```

Listing 1: Matrix Method Utilities

```
# FIRST PAPER (Keener, 1993)

from common import (
  Csv, PrefMethod, make_pref_matrix,
  power_method, power_method_nonlinear, sort_by_rank,
  disp_ranks
)

import numpy as np
from matplotlib import colormaps as cmaps
import matplotlib.pyplot as plt

def main():
  csv = Csv('../data/games.csv')

  print(len(csv.data))

  # config
  pref_method = PrefMethod.NONLINEAR
  power_iterations = 100

  # calculate ranking
  teams, R, M, A, E = make_pref_matrix(csv, pref_method)
  r = power_method(A, power_iterations)
  r_nonlinear = power_method_nonlinear(
    E,
    M,
    f = lambda x: (.05*x + x*x) / (2 + .05*x + x*x),
    num_iterations = power_iterations)

  team_ranks_linear = sort_by_rank(teams, A @ r)
  team_ranks_nonlinear = sort_by_rank(teams, A @ r_nonlinear)

  fig, ax = plt.subplots()
  ax.set_title('Preference Matrix')
  im = ax.imshow(R, cmap=cmaps['bone'])
  fig.colorbar(im, ax=ax)
  disp_ranks(team_ranks_linear, title='Eigenvector Ranking (Keener 1993)')
  disp_ranks(team_ranks_nonlinear, title='Nonlinear Ranking (Keener 1993)'
   )
  plt.show()

  for team, rank in team_ranks_linear[:20]:
    print(f'{team}: {rank}')

if __name__ == '__main__':
  main()
```

Listing 2: Driver Code for Keener 1993

```python
# SECOND PAPER (Jech, 1983)

from common import (
  Csv, PrefMethod, make_pref_matrix,
  make_rank_matrix, fixed_point_approx, sort_by_rank,
  disp_ranks
)

import numpy as np
from matplotlib import colormaps as cmaps
import matplotlib.pyplot as plt

def main():
  csv = Csv('../data/games.csv')

  #config
  pref_method = PrefMethod.DISCRETE
  fixed_iterations = 100

  # calculate ranking
  teams, R, M, _, _ = make_pref_matrix(csv, pref_method)

  v = fixed_point_approx(R, M, fixed_iterations)
  P = make_rank_matrix(v)
  mask = np.eye(P.shape[0]) == 0
  P_adj = np.multiply(P, mask)
  r = P_adj @ np.ones(P_adj.shape[1])
  r /= np.full_like(r, P_adj.shape[1] - 1)
  team_ranks = sort_by_rank(teams, r)

  fig, ax = plt.subplots()
  ax.set_title('Rank Matrix')
  im = ax.imshow(P, cmap=cmaps['viridis'])
  fig.colorbar(im, ax=ax)
  disp_ranks(team_ranks, title='Probability Ranking (Jech 1983)')
  plt.show()

  for team, rank in team_ranks[:20]:
    print(f'{team}: {rank}')

if __name__ == '__main__':
  main()
```

Listing 3: Driver Code for Jech 1983