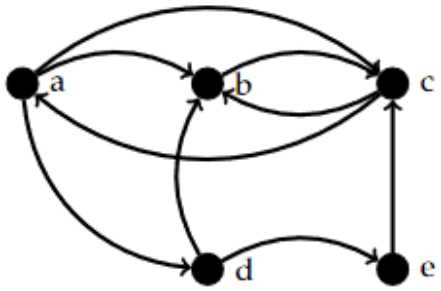
Exercise P22: Consider the graph of five web pages a, b, c, d, e, shown below, where each page links to some of the other pages as show. The goal of this problem, is to compute the Google PageRank of the pages. In fact, this problem, is about the essential idea which created Google, only about 25 years ago.

Figure 1: P22 graph



a. Start by asking Google to document itself. Go to Google Scholar and search for 'pagerank citation ranking'.The first link, the one with highest PageRank, will be a technical preprint by Page, Brin, Motwani, and Winograd; it has more than 15000 citations. Download the 17 page PDF and read it. It describes a simpler time.

The main ideas are about a web as a directed graph, as in the example shown above, that the limiting probability that a certain random web surfer visits any particular page is its PageRank, and finally that a search engine should report results in the Page Rank order. We can regard PageRank as simply an eigenvector of a certain matrix A, a transition probability matrix for a Markov chain. To build A we will start with the adjacency matrix G of the web. Here is how to start with a web as directed graph and construct G,A and the eigenvector of PageRank values:
Let W be a connected directed graph of n webpages, index the pages 1 through n. Let G, be the nxn connectivity matrix W that is,

$$g_{ij} = \left\{ \begin{array}{ll} 1 & \text{There is a hyperlink to } i \text{ from } j \\ 0 & \text{Otherwise} \end{array} \right\}$$

The number of nonzeros in G is the total number of hyperlinks in W. Let c_j be the columns sums of G;

$$c_j = \sum_{i=1}^{n} g_{ij}$$

The value of c_j gives the outdegreee of the jth page. Let p be the fraction of the time that the random walk follows some link, so $1-p$ is the fraction of time that an arbitrary page is chose. Define $\delta = (1-p)/n$.Now let A be the nxn matrix whose entries are,

$$a_{ij} = \frac{pg_{ij}}{c_j} + \delta$$

The matrix A is the transition probability matrix of the random-surfer Markov chain. An old theorem says the largest eigenvalue of A is equal to one and the corresponding eigenvector x, which satisfies

$$Ax = x$$

is unique up to a scaling factor, and has positive entries. Choose the scaling so that,

$$1 = \sum_{i=1}^{n} x_i.$$

The entries of x are the Page Ranks of webpages in W.
For a realistic web, G is huge and sparse because most pairs of webpages are not connected by a single link. Google suggests $p = .85$. Matrix A is not spare because most entries are equal to the small constant $\delta > 0$. The 'old' Perron-Frobenius Theorem says that if all entries in a matrix are positive then all the entries of the eigenvector associated to the largest eigenvalue can we chose to be positive.

b. Folling my description of the process, compute G for the web shown above.

Solution:
From the passage above we know that G is the adjacency matrix for the figure 1. Building the adjacency matrix we get,

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

c. Continuing from my description, build A from G using $p = .85$ as suggested. Finally, use Matlabs eig() to compute the PageRanks. Which page gets the highest PageRank? explain in intuitive terms.

Solution:
The following Matlab script computes A from G, and uses the eig() function to compute the PageRanks.

Code:

```
G =[0 0 1 0 0;
    1 0 1 1 0;
    1 1 0 0 1;
    1 0 0 0 0;
    0 0 0 1 0];

%Compute the columns sums
c = sum(G, 1);
%Repeat column sum so we can divide entrywise to compute A
C = repmat(c,5, 1);
%Set p and delta
p = .85;
delta = (1 - p)/5;
%Compute A
A = (p.*G)./C + delta;

%Compute Eigenvalues and Eigenvectors of A
[V, D] = eig(A);
[M,I] = max(diag(D));
PageRank = V(:,I)/sum(V(:,I));
```

Console:

```
G =

     0     0     1     0     0
     1     0     1     1     0
     1     1     0     0     1
     1     0     0     0     0
     0     0     0     1     0

A =

    0.0300    0.0300    0.4550    0.0300    0.0300
    0.3133    0.0300    0.4550    0.4550    0.0300
    0.3133    0.8800    0.0300    0.0300    0.8800
    0.3133    0.0300    0.0300    0.0300    0.0300
    0.0300    0.0300    0.0300    0.4550    0.0300

PageRank =
```

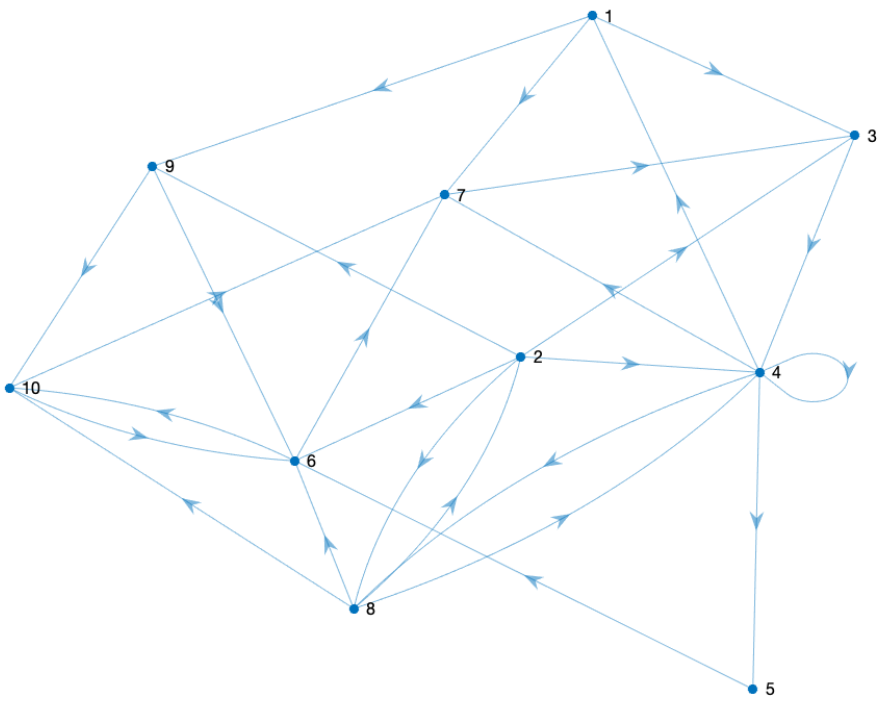
```
0.1909
0.2807
0.3786
0.0841
0.0657
```

From our results we can see that pages sorted by PageRank are c, b, a, d, e. C is likely rated the highest since it shares a loop with the next two highest pages a and b in terms of total degree. It also makes sense that b is second highest as it has the same in-degree as c except they are from pages with lesser total degree.

d. Draw a new web W with 10 pages. Recompute G, A, and the PageRanks.

Solution:
The following Matlab script generates a 10 node directed graph(web) W by generating G and computes A, and the PageRanks.

Figure 2: Generated P22D graph



Code:

```
%Set the number of nodes
n = 10;
%Generate Random G
G = round(.70*rand(n, n));
%Plot W, input is G' because of digraph()
W = digraph(G');
plot(W, 'Layout','force')

%Compute the column sums
c = sum(G, 1);
%Repeat column sum so we can divide entrywise to compute A
C = repmat(c,n, 1);
%Set p and delta
p = .85;
delta = (1 - p)/n;
%Compute A
A = (p.*G)./C + delta;

%Compute Eigenvalues and Eigenvectors of A
[V, D] = eig(A);
[M,I] = max(diag(D));
```

```
PageRank = V(:,I)/sum(V(:,I));

%ExtraCredit Function call
[RandomPageRank, randomWalk, NodeCounts] = RandomWalk(J, 100000);
```

Console:

```
G
     0     0     0     1     0     0     0     0     0     0
     0     0     0     0     0     0     0     1     0     0
     1     1     0     0     0     0     1     0     0     0
     0     1     1     1     0     0     0     1     0     0
     0     0     0     1     0     0     0     0     0     0
     0     1     0     0     1     0     0     1     1     1
     1     0     0     1     0     1     0     0     0     1
     0     1     0     1     0     0     0     0     0     0
     1     1     0     0     0     0     0     0     0     0
     0     0     0     0     0     1     0     1     1     0

A
    0.0150    0.0150    0.0150    0.1850    0.0150    0.0150    0.0150    0.0150    0.0150    0.0150
    0.0150    0.0150    0.0150    0.0150    0.0150    0.0150    0.0150    0.2275    0.0150    0.0150
    0.2983    0.1850    0.0150    0.0150    0.0150    0.0150    0.8650    0.0150    0.0150    0.0150
    0.0150    0.1850    0.8650    0.1850    0.0150    0.0150    0.0150    0.2275    0.0150    0.0150
    0.0150    0.0150    0.0150    0.1850    0.0150    0.0150    0.0150    0.0150    0.0150    0.0150
    0.0150    0.1850    0.0150    0.0150    0.8650    0.0150    0.0150    0.2275    0.4400    0.4400
    0.2983    0.0150    0.0150    0.1850    0.0150    0.4400    0.0150    0.0150    0.0150    0.4400
    0.0150    0.1850    0.0150    0.1850    0.0150    0.0150    0.0150    0.0150    0.0150    0.0150
    0.2983    0.1850    0.0150    0.0150    0.0150    0.0150    0.0150    0.0150    0.0150    0.0150
    0.0150    0.0150    0.0150    0.0150    0.0150    0.4400    0.0150    0.2275    0.4400    0.0150

PageRank
    0.0516
    0.0269
    0.1729
    0.2151
    0.0516
    0.1312
    0.1632
    0.0561
    0.0342
    0.0972
```

ExtraCredit Build a random wlak probram which traverses a web as would a Google 'bot'. Build-up an estimate of PageRank based in the fraction of the time spent at a given page. It is up to you how to represent a web in some data structure; it could be the matrix G, or not. Check that your surfer converges to the results computed in parts c and d.

Solution:
Consider the following code,
Code:

```
function [PageRank, Walk, count] = RandomWalk(G, WalkLength)
n = size(G, 1);
Walk = zeros(1, WalkLength);
count = zeros(1, n);

%Randomly choose starting node
Walk(1) = ceil(n*rand(1));
%Update node count
count(Walk(1)) = count(Walk(1)) + 1 ;
```

```
for i = 2:WalkLength

    %Add current node to possible paths
    PossiblePaths = [];
    CrossRoad = G(:, Walk(i - 1));

    %Iterate through CrossRoad, adding index to PossiblePaths
    for j = 1:n
        if CrossRoad(j) == 1
            %Add all nodes incident to current node
            PossiblePaths = [PossiblePaths j];
        end
    end


    %Compute deltaFactor
    deltaFactor = rand(1);
    if deltaFactor >= .85
            NextNode = ceil(n*rand(1));
        else
        %Sample from possible paths
        NextNode = randsample(PossiblePaths, 1);
    end



    %Update Walk and count
    Walk(i) = NextNode;
    count(Walk(i)) = count(Walk(i)) + 1 ;
end
%Compute Propotions
PageRank = count./WalkLength;
end
```

Console:

```
## Running RandomWalk on the Graph from part C
P22C
>> PageRank

    0.1909
    0.2807
    0.3786
    0.0841
    0.0657

>> [RandomPageRank, randomWalk, NodeCounts] = RandomWalk(G, 1000000);
>> RandomPageRank

    0.2466
    0.3596
    0.2218
    0.0999
    0.0721

>> norm(abs(RandomPageRank - PageRank))

    0.6590
```

```
>> [RandomPageRank, randomWalk, NodeCounts] = RandomWalk(G, 10000000);
>> RandomPageRank

    0.2467
    0.3590
    0.2219
    0.0999
    0.0725

>> norm(abs(RandomPageRank - PageRank))

    0.6584
```

P22D

```
>> PageRank

    0.0516
    0.0269
    0.1729
    0.2151
    0.0516
    0.1312
    0.1632
    0.0561
    0.0342
    0.0972

>> [RandomPageRank, randomWalk, NodeCounts] = RandomWalk(G, 1000000);
>> RandomPageRank

    0.1132
    0.1072
    0.1462
    0.0965
    0.0366
    0.1212
    0.1596
    0.0498
    0.0651
    0.1047

>> norm(abs(RandomPageRank - PageRank))

    0.6531

>> [RandomPageRank, randomWalk, NodeCounts] = RandomWalk(G, 10000000);
>> RandomPageRank

    0.1129
    0.1070
    0.1468
    0.0966
    0.0367
    0.1213
    0.1593
    0.0495
    0.0652
    0.1048

>> norm(abs(RandomPageRank - PageRank))

    0.1621
```

Exercise 20.1: Let $A \in \mathbb{C}^{mxm}$ be nonsingular. Show that A has an LU factorization if and only if for each k with $1 \leqslant k \leqslant m$, the upper-left kxk block of $A_{1:k,1:k}$ is nonsingular.(Hint: The row operations of Gaussian elimination leave the determinants of $A_{1:k,1:k}$ unchanged.)

Prove that this LU factorization is unique.

Proof. Suppose that $A \in \mathbb{C}^{mxm}$ is nonsingular and that has an LU factorization. Recall that term-wise each a_{ij} is produced by the following inner-products,

$$a_{ij} = L_i^* U_j$$

where L_i is the ith row of L and U_j is the jth column of U. Since L and U are triangular for $i, j \leqslant k$, since it only removes the addition of zeros,

$$a_{ij} = L_i^* U_j = L_{i[1:k,1:k]},^* U_{j[1:k,1:k]}.$$

Analogously it follows that,

$$A_{1:k,1:k} = L_{1:k,1:k} U_{1:k,1:k}.$$

Therefore each upper-left kxk block of $A_{1:k,1:k}$ has an LU factorization. Consider taking the determinant of both sides of the previous equation, and let u be terms in U,

$$\begin{aligned} det(A_{1:k,1:k}) &= det(L_{1:k,1:k} U_{1:k,1:k}), \\ det(A_{1:k,1:k}) &= det(L_{1:k,1:k}) det(U_{1:k,1:k}), \\ det(A_{1:k,1:k}) &= 1 \prod_{i=1}^{m} u_{i,i}. \end{aligned}$$

Since A is nonsingular with a full LU factorization we know that none of the terms in the diagonal of U can be zero and therefore $det(A_{1:k,1:k}) \neq 0$.

Suppose that $A \in \mathbb{C}^{mxm}$ be nonsingular and for each k with $1 \leqslant k \leqslant m$, the upper-left kxk block of $A_{1:k,1:k}$ is nonsingular. To prove that A has an LU factorization we will proceed by induction on k. The base case is clearly true,

$$\begin{aligned} A_{1,1} &= 1 * a_1. \\ &= L_{1,1} U_{1,1}, \end{aligned}$$

Which must be valid since $A_{1,1}$ is nonsingular. Suppose that there exist an LU factorization for some $A_{k,k}$. Now consider $A_{k+1,k+1}$ and note it's relation to $A_{k,k}$. As an aside we should note that for all dimensions of A we are bounded above by m. Let $\hat{v}, d$ and $\hat{u}, d$ be the final row and column of $A_{k+1,k+1}$ respectively,

$$A_{k+1,k+1} = \begin{bmatrix} A_{k,k} & \hat{u} \\ \hat{v} & d \end{bmatrix}$$

Recall that by the induction hypothesis, $A_{k,k}$ has an LU factorization so there must exist some factor such that,

$$\begin{bmatrix} L_{k,k} & 0 \\ \hat{v}U_{k,k}^{-1} & 1 \end{bmatrix} \begin{bmatrix} U_{k,k} & L_{k,k}^{-1}\hat{u} \\ 0 & -\hat{v}U_{k,k}^{-1}L_{k,k}^{-1}\hat{u} + d \end{bmatrix} = \begin{bmatrix} L_{k,k}U_{k,k} + 0 & L_{k,k}L_{k,k}^{-1}\hat{u} + 0 \\ \hat{v}U_{k,k}^{-1}U_{k,k} + 0 & \hat{v}U_{k,k}^{-1}L_{k,k}^{-1}\hat{u} - \hat{v}U_{k,k}^{-1}L_{k,k}^{-1}\hat{u} + d \end{bmatrix} = \begin{bmatrix} A_{k,k} & \hat{u} \\ \hat{v} & d \end{bmatrix}$$

Note that by this factorization $det(A_{k+1,k+1}) = det(U_{k,k})(-\hat{v}U_{k,k}^{-1}L_{k,k}^{-1}\hat{u} + d)$ and since we know that $det(A_{k+1,k+1})$ and $det(U_{k,k})$ are not equal to zero it must be the case that $(-\hat{v}U_{k,k}^{-1}L_{k,k}^{-1}\hat{u} + d)$ is also greater than zero. Thus $A_{k+1,k+1}$ has a valid LU factorization.

Suppose that the matrix A has two valid LU factorization L_1U_1 and L_2U_2. Consider the following matrix algebra,

$$L_1U_1 = L_2U_2,$$
$$L_2^{-1}L_1 = U_2U_1^{-1}.$$

Recall that in homework 1 we proved that the inverse of a triangular matrix is also triangular, and clearly the product of two triangular matrices is also triangular. Therefore the last equality shows an upper triangular matrix is equal to a lower triangular matrix, a condition only satisfied by the identity,

$$L_2^{-1}L_1 = I = U_2U_1^{-1}.$$

Thus we get the following equalities $L_2 = L_1$ and $U_2 = U_1$. □

Exercise 20.1(a): Suppose an *mxm* matrix A is written in the block form

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

where A_{11} is an *nxn* matrix and A_{22} is $(m-n)x(m-n)$ Assume A satisfies the condition in Exercise 20.1. Verify the following,

$$\begin{bmatrix} I & 0 \\ -A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{bmatrix}$$

Solution:
Let's consider each block matrix multiplication individually to ensure that the partition

dimension sizes agree for block multiplication. The first multiplication works since the I on the top left must be nxn since $-A_{21}A_{11}^{-1}$ is an $(m-n)xn$ matrix.

$$IA_{11} + 0 = A_{11}$$

Here the dimensions agree since $-A_{21}$ is an $(m-n)xn$ matrix and A_{11}^{-1} is an nxn matrix.

$$-A_{21}A_{11}^{-1}A_{11} + IA_{21} = 0$$

This multiplication works since the bottom right I must be an $(m-n)x(m-n)$ identity since $-A_{21}A_{11}^{-1}$ is an $(m-n)xn$ matrix.

$$IA_{12} + 0 = A_{12}$$

Here note that A_{21} is $(m-n)xn$ and A_{12} is $nx(m-n)$ so the product on the right is (mxm).

$$-A_{21}A_{11}^{-1}A_{12} + A_{22} = A_{22} - A_{21}A_{11}^{-1}A_{12}$$

Exercise 21.1: Let A be the following $4x4$ matrix,

$$A = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix}$$

a Determine the $det(A)$ using the factorization in (20.5),

$$\begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 4 & 3 & 1 & 0 \\ 3 & 4 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 2 \end{bmatrix}.$$

Solution:
Using this factorization we can quickly compute the determinant since, the determinant of a triangular matrix is the product of it's diagonal entries. Doing so we get the following,

$$\begin{aligned} det(A) &= det\left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 4 & 3 & 1 & 0 \\ 3 & 4 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 2 \end{bmatrix}\right), \\ &= det\left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 4 & 3 & 1 & 0 \\ 3 & 4 & 1 & 1 \end{bmatrix}\right) det\left(\begin{bmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 2 \end{bmatrix}\right), \\ &= 1^2 * 2^3 = 8. \end{aligned}$$

b. Determine $det(A)$ using the factorization in (21.3)

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{3}{4} & 1 & 0 & 0 \\ \frac{1}{2} & -\frac{2}{7} & 1 & 0 \\ \frac{1}{4} & -\frac{3}{7} & \frac{1}{3} & 1 \end{bmatrix} \begin{bmatrix} 8 & 7 & 9 & 5 \\ 0 & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ 0 & 0 & -\frac{6}{7} & -\frac{2}{7} \\ 0 & 0 & 0 & \frac{2}{3} \end{bmatrix}$$

Solution:
Recall that the determinant of a permutation matrix is equal to -1^r where r the number of row interchanges. Note that the given permutation matrix has 3 row interchanges, $r_1 \to r_4$, $r_2 \to r_3$, $r_3 \to r_4$. With that in mind we proceed similarly to the last part,

$$-1det(A) = 8 * \frac{7}{4} * -\frac{6}{7} * \frac{2}{3} = -8,$$
$$det(A) = 8.$$

c. Describe how Gaussian Elimination with partial pivoting can be used to find the determinant of a general square matrix.

Solution:
The process would be exactly the same as the previous problem. We would perform LU factorization (Gaussian Elimination) on the matrix, if the matrix is not invertible we would find a zero in the diagonal of U and when we compute the determinant we would get zero. Otherwise we get the following,

$$det(A) = (-1)^r \prod_{i=1}^{m} u_{i,i}$$

Exercise 21.4(a): Gaussian elimniation can be used to compute the inverse A^{-1} of a nonsingular matrix $A \in \mathbb{C}^{mxm}$, though it is rarely really necessary to do so.
Describe an algorithm for computing A^{-1} by solving m system of equations, and show that it's asymptotic operation count is $8m^3/3$ flops.

Solution:
Consider the following equation for an nonsingular matrix $A \in \mathbb{C}^{mxm}$,

$$AA^{-1} = I.$$

Written as m systems of equations we get,

$$Aa_i^{-1} = e_i,$$

where a_i^{-1} is the ith column of A^{-1} and e_i is the ith standard basis vector. Note that before we begin solving the systems we must first compute the LU factors and as discussed in lectures 20 and 21 this process requires operations on the order of size $\frac{2}{3}m^3$. Now note that in order to solve a single system we must perform forward substitution with the L factor and then back substitution with the U factor, both process take $2m^2$ operations (Discussed above equation 17.2). So to solve each of the m systems we need $4m^2$ operations, for a total of $4m^3$ operations. Summing together our total work we get,

$$\text{FLOPs} = \frac{2}{3}m^3 + 4m^2 = \frac{8}{3}m^3$$