(a) Create **pagerank1(G)** by modifying **pagerank** so that it just computes the PageRanks, but does not do any plotting or printing.

Ans:

Unset

```
function x = pagerank1(U,G,p)

   if nargin < 3, p = .85; end

   % Eliminate any self-referential links
   G = G - diag(diag(G));

   % c = out-degree, r = in-degree
   [n,n] = size(G);
   c = sum(G,1);
   r = sum(G,2);

   % Scale column sums to be 1 (or 0 where there are no out links).
   k = find(c~=0);
   D = sparse(k,k,1./c(k),n,n);

   % Solve (I - p*G*D)*x = e
   e = ones(n,1);
   I = speye(n,n);
   x = (I - p*G*D)\e;

   % Normalize so that sum(x) == 1.
   x = x/sum(x);

end
```

(b) Create `pagerank2(G)` by modifying `pagerank1` to use inverse iteration instead of solving the sparse linear system. The key statements are

```
x = (I - A)\e
x = x/sum(x)
```

What should be done in the unlikely event that the backslash operation involves a division by zero?

Ans:

Unset

```
function x = pagerank2(U, G, p, tol, max_iter)
   if nargin < 3, p = 0.85; end
   if nargin < 4, tol = 1e-6; end  % Tolerance for convergence
   if nargin < 5, max_iter = 100; end  % Maximum number of
iterations

   % Eliminate any self-referential links
   G = G - diag(diag(G));

   % Get size and compute out-degrees
   [n, ~] = size(G);
   c = sum(G, 1);

   % Scale columns to sum to 1 (or 0 where there are no out-links)
   k = find(c ~= 0);
   D = sparse(k, k, 1 ./ c(k), n, n);

   % Define matrix A and the vector e
   A = p * G * D;
   e = ones(n, 1);

   % Add a small regularization term to I - A to avoid division by
zero
   reg_param = 1e-9;  % Regularization parameter
   I = speye(n, n);
```

```
   x = (I - A + reg_param * I) \ e;
   x = x / sum(x); % Normalize x to make it a probability
distribution

end
```

To avoid zeros in division, we can add a small regularization term to the numerator I-A, which in this case I have taken as an Identity matrix time small constant(1e-9).

(c) Create **pagerank3(G)** by modifying **pagerank1** to use the power method instead of solving the sparse linear system. The key statements are

```
G = p*G*D
z = ((1-p)*(c~=0) + (c==0))/n;
while termination_test
   x = G*x + e*(z*x)
end
```

What is an appropriate test for terminating the power iteration?

(d) Use your functions to compute the PageRanks of the six-node example discussed in the text. Make sure you get the correct result from each of your three functions.

Ans:

Unset

```
function x = pagerank3(U, G, p, tol, max_iter)

   if nargin < 3, p = 0.85; end
   if nargin < 4, tol = 1e-6; end     % Tolerance for convergence
   if nargin < 5, max_iter = 100; end % Maximum number of iterations

   % Eliminate any self-referential links
   G = G - diag(diag(G));

   % Get size and compute out-degrees
   [n, ~] = size(G);
   c = sum(G, 1);  % Out-degree
   k = find(c ~= 0);
```

```
    D = sparse(k, k, 1 ./ c(k), n, n);

    e = ones(n, 1);

    % Modify G with damping factor
    G = p * G * D;
    z = ((1 - p) * (c ~= 0) + (c == 0)) / n;

    % Initialize the page rank vector x
    x = ones(n, 1) / n;

    % Power method iteration
    for iter = 1:max_iter
        x_old = x;
        x = G * x + e * (z * x);

        % Check for convergence using the termination test
        if norm(x - x_old, 1) < tol
            break;
        end
    end

    % Normalize x to make it a probability distribution
    x = x / sum(x);

end
```

Power Iteration can be stopped when the difference of two matrices generated in consecutive iterations is less than the tolerance level considered.
In this code sample, I have taken the 1 Norm (maximum column sum) of the difference matrix, checking if it's greater than or equal to 1e-6 for convergence.

(d) Use your functions to compute the PageRanks of the six-node example discussed in the text. Make sure you get the correct result from each of your three functions.

Ans:
Variables
U:

U

1x6 cell

	1	2	3	4	5	6
1	http://www.alpha.com	http://www.beta.com	http://www.gamma.com	http://www.delta.com	http://www.rho.com	http://www.sigma.com

G:

G

6x6 sparse double

```
val =


   (2,1)        1
   (6,1)        1
   (3,2)        1
   (4,2)        1
   (4,3)        1
   (5,3)        1
   (6,3)        1
   (1,4)        1
   (1,6)        1
```

P:

p

1x1 double

	1
1	0.8500

Outputs:
pagerank1(U,G,p):

```
ans =

    0.3210
    0.1705
    0.1066
    0.1368
    0.0643
    0.2007
```

pagerank2(U,G,p):

```
ans =

    0.3210
    0.1705
    0.1066
    0.1368
    0.0643
    0.2007
```

pagerank3(U,G,p):

```
ans =

    0.3210
    0.1705
    0.1066
    0.1368
    0.0643
    0.2007
```

(b) Use this function as a template to write a function that computes PageRank in some other programming language.

```
[n,n] = size(G);
for j = 1:n
   L{j} = find(G(:,j));
   c(j) = length(L{j});
end

% Power method

p = .85;
delta = (1-p)/n;
x = ones(n,1)/n;
z = zeros(n,1);
cnt = 0;
while max(abs(x-z)) > .0001
   z = x;
   x = zeros(n,1);
   for j = 1:n
      if c(j) == 0
         x = x + z(j)/n;
      else
         x(L{j}) = x(L{j}) + z(j)/c(j);
      end
   end
   x = p*x + delta;
   cnt = cnt+1;
```

Ans: PageRank in Python:-

Python

```python
from copy import deepcopy

def pagerankpow(G):
    n = len(G)
    L = [[] for i in range(n)]
    c = [0 for i in range(n)]

    for j in range(n):
        for i in range(n):
            if G[i][j] == 1:
                L[j].append(i)
        c[j] = len(L[j])

    p = 0.85
    delta = (1-p)/n
    x = [1/n for i in range(n)]
    z = [0 for i in range(n)]
    cnt = 0

    while max([abs(x[i]-z[i]) for i in range(n)]) > 1e-4:
        z = deepcopy(x)
        x = [0 for i in range(n)]
        for j in range(n):
            if c[j] == 0:
                for i in range(n):
                    x[i] += z[j]/n
            else:
                for i in L[j]:
                    x[i] += z[j]/c[j]
        for i in range(n):
            x[i] = p*x[i] + delta
        cnt += 1
```

```
   return x, cnt

G = [[0., 0., 0., 1., 0., 1.],
     [1., 0., 0., 0., 0., 0.],
     [0., 1., 0., 0., 0., 0.],
     [0., 1., 1., 0., 0., 0.],
     [0., 0., 1., 0., 0., 0.],
     [1., 0., 1., 0., 0., 0.]]

x, cnt = pagerankpow(G)

print("PageRank vector: ", x)
print("Number of iterations: ", cnt)
```