HW1 - Q1: Linear Algebra Basics (30 points)

Notes:

- Questions (a), (b), (c), and (d) need to be typewritten.
- Important:
  - Write all the steps of the solution.
  - Use proper LATEX formatting and notation for all mathematical equations, vectors, and matrices.

(a) Given L_1 and L_2 are two lower triangular matrices of size $n \times n$, prove that $L_1 L_2$ is also a lower triangular matrix. Further, prove by induction that multiplication of $m\ (m > 2)$ lower triangular matrices ($L_1, L_2, \ldots, L_m$) is also a lower triangular matrix. (6 points)

Your answer here:

Given: $\mathbf{L_1}$ and $\mathbf{L_2}$ are lower triangular matrices of size $n \times n$

$$\implies \mathbf{(L_1)_{ij}} = \begin{cases} 0, \ \forall\, i < j \\ (L_1)_{ij}, \ \forall\, i \geq j \end{cases} \quad , \quad \mathbf{(L_2)_{ij}} = \begin{cases} 0, \ \forall\, i < j \\ (L_2)_{ij}, \ \forall\, i \geq j \end{cases}$$

Let $\mathbf{L} = \mathbf{L_1 L_2}$

To prove that $\mathbf{L}$ is a lower triangular matrix, we need to show that $\mathbf{L_{ij}} = 0, \ \forall\, i < j$

Fixing $i < j$, we have $\mathbf{L_{ij}} =$ (i^{th} row of $\mathbf{L_1}$)$\cdot$(j^{th} column of $\mathbf{L_2}$)

$$\implies \mathbf{L_{ij}} = \sum_{k=1}^{n} \mathbf{(L_1)_{ik} (L_2)_{kj}}$$

$$\implies \mathbf{L_{ij}} = \underbrace{\sum_{k=1}^{i} \mathbf{(L_1)_{ik} (L_2)_{kj}}}_{0 \ (\because k \leq i < j \implies \mathbf{(L_2)_{kj}} = 0)} + \underbrace{\sum_{k=i+1}^{n} \mathbf{(L_1)_{ik} (L_2)_{kj}}}_{0 \ (\because i < k \leq j \implies \mathbf{(L_1)_{ik}} = 0)}$$

$$\implies \mathbf{L_{ij}} = 0 + 0 = 0$$

Hence proved

Also, to prove by induction $L_1 L_2 \ldots L_m$ is a lower triangular matrix for m>3, we have,

For m=3, $L_1 L_2 L_3$ = $(L_1 L_2) L_3$ = $L_{12} L_3$ where L_{12} is a lower triangular matrix as proved earlier that product of 2 lower triangular matrices is also a lower trianglular matrix.

Hence, $L_1 L_2 L_3$ = $(L_{12} L_3)$ = (L_{123}) which will also be lower triangular.

Assuming the condition also holds for m=k, we have $(L_{123\ldots k})$ is a lower triangular matrix.

For m = k + 1, $L_1 L_2 \ldots L_k L_{k+1}$ = $(L_1 L_2 \ldots L_k) L_{k+1}$ = $L_{12\ldots k} L_{k+1}$ =$L_{12\ldots k+1}$ which is also lower triangular.

Hence proved.

(b) Use Gauss elimination to solve the following equations: (8 points)

$-4x_1 + 5x_2 - 5x_3 = -29 \backslash -8x_1 - 5x_2 - 3x_3 = -15 \backslash 16x_1 - 5x_2 + 6x_3 = 45$

Your answer here:

$$\left[\begin{array}{ccc|c} -4 & 5 & -5 & -29 \\ -8 & -5 & -3 & -15 \\ 16 & -5 & 6 & 45 \end{array}\right] \begin{array}{l} R_2 \to R_2 - 2R_1 \\ R_3 \to R_3 + 4R_1 \end{array} \left[\begin{array}{ccc|c} -4 & 5 & -5 & -29 \\ 0 & -15 & 7 & -43 \\ 0 & 15 & -14 & -71 \end{array}\right] R_3 \to R_3 + R_2 \left[\begin{array}{ccc|c} -4 & 5 & -5 & -29 \\ 0 & -15 & 7 & -43 \\ 0 & 0 & -7 & -28 \end{array}\right]$$

$$\begin{cases} -4x_1 + 5x_2 - 5x_3 = -29 \\ -15x_2 + 7x_3 = 43 \\ -7x_3 = -28 \end{cases} \to \begin{cases} x_1 = 1 \\ x_2 = -1 \\ x_3 = 4 \end{cases}$$

(c) Do the LU decomposition for the matrix obtained in (b). Using the matrices L and U, do forward and backward substitution and solve for $\mathbf{x}$. Match your answer with the solution obtained in (b). (8 points)

Your answer here:

Let $A = LU$

$$\begin{bmatrix} -4 & 5 & -5 \\ -8 & -5 & -3 \\ 16 & -5 & 6 \end{bmatrix} \begin{array}{c} R_2 \to -2R_1 + R_2 \\ R_3 \to 4R_1 + R_3 \end{array} \begin{bmatrix} -4 & 5 & -5 \\ 0 & -15 & 7 \\ 0 & 15 & -14 \end{bmatrix} R_3 \to R_2 + R_3 \begin{bmatrix} -4 & 5 & -5 \\ 0 & -15 & 7 \\ 0 & 0 & -7 \end{bmatrix}$$

$$\to L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -4 & -1 & 1 \end{bmatrix}, U = \begin{bmatrix} 4 & 5 & -5 \\ 0 & -15 & 7 \\ 0 & 0 & 7 \end{bmatrix}$$

Solve $AX = B \to$ Solve $LUX = B$

Let $UX = Y$

Step 1: First solve $LY = B$ for Y

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -4 & -1 & 1 \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} -29 \\ -15 \\ 45 \end{bmatrix} \to \begin{cases} y_1 = -29 \\ 2y_1 + y_2 = -15 \\ -4y_1 - y_2 + y_3 = 45 \end{cases} \to \begin{cases} y_1 = -29 \\ y_2 = 43 \\ y_3 = -28 \end{cases}$$

Step 2: Then solve $UX = Y$ for X

$$\begin{bmatrix} 4 & 5 & -5 \\ 0 & -15 & 7 \\ 0 & 0 & 7 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -29 \\ 43 \\ -28 \end{bmatrix} \to \begin{cases} -4x_1 + 5x_2 - 5x_3 = -29 \\ -15x_2 + 7x_3 = 43 \\ -7x_3 = -28 \end{cases} \to \begin{cases} x_1 = 1 \\ x_2 = -1 \\ x_3 = 4 \end{cases}$$

(d) Do the QR decomposition for the matrix obtained in (b) using Gram-Schmidt algorithm. Using the decomposition, solve for $\mathbf{x}$. Match your answer with the solution obtained in problem (b). (8 points)

Your answer here:

Given : From part (b), we have $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A}$ = $\begin{bmatrix} -4 & 5 & -5 \\ -8 & -5 & -3 \\ 16 & -5 & 6 \end{bmatrix}$, $\mathbf{x}$ = $\begin{bmatrix} \mathrm{x}_1 \\ \mathrm{x}_2 \\ \mathrm{x}_3 \end{bmatrix}$, $\mathbf{b}$ = = $\begin{bmatrix} -29 \\ -15 \\ 45 \end{bmatrix}$

To Solve: x_1, x_2, x_3 using QR Factorization by the Gram-Schmidt Algorithm.

Solution:

Using the Gram-Schmidt Algorithm, we have : $\mathbf{A}$ = $\begin{bmatrix} | & | & | \\ \mathbf{a_1} & \mathbf{a_2} & \mathbf{a_3} \\ | & | & | \end{bmatrix}$, $\mathbf{Q}$ = $\begin{bmatrix} | & | & | \\ \mathbf{q_1} & \mathbf{q_2} & \mathbf{q_3} \\ | & | & | \end{bmatrix} = \begin{bmatrix} | & | & | \\ \frac{\mathbf{a}_1}{||\mathbf{a}_1||} & \frac{\mathbf{a}_2^{\perp}}{||\mathbf{a}_2^{\perp}||} & \frac{\mathbf{a}_3^{\perp}}{||\mathbf{a}_3^{\perp}||} \\ | & | & | \end{bmatrix}$

, $\mathbf{R}$ = $\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ 0 & r_{22} & r_{23} \\ 0 & 0 & r_{33} \end{bmatrix}$ = $\begin{bmatrix} ||\mathbf{a_1}|| & \mathbf{a}_2^T q_1 & \mathbf{a}_3^T q_1 \\ 0 & ||\mathbf{a_2}^{\perp}|| & \mathbf{a}_3^T q_2 \\ 0 & 0 & ||\mathbf{a_3}^{\perp}|| \end{bmatrix}$

where $\mathbf{a_1}$, $\mathbf{a_2}$, $\mathbf{a_3}$ are the column vectors of $\mathbf{A}$, $\mathbf{q_1}$, $\mathbf{q_2}$, $\mathbf{q_3}$ are the column vectors of orthogonal matrix $\mathbf{Q}$, and $\mathbf{R}$ is an upper triangular matrix.

Step 1: To find $\mathbf{Q}$ and $\mathbf{R}$

To calculate $\mathbf{q_1}$, we have $\mathbf{a_1} = \begin{bmatrix} -4 \\ -8 \\ 16 \end{bmatrix}$,

$$\implies ||\mathbf{a_1}|| = \sqrt{-4^2 + -8^2 + 16^2} = 4\sqrt{21} = r_{11} \tag{1}$$

$$\implies \mathbf{q_1} = \frac{\mathbf{a}_1}{||\mathbf{a}_1||} = \frac{1}{4\sqrt{21}} \begin{bmatrix} -4 \\ -8 \\ 16 \end{bmatrix} = \begin{bmatrix} \frac{-1}{\sqrt{21}} \\ \frac{-2}{\sqrt{21}} \\ \frac{4}{\sqrt{21}} \end{bmatrix} \tag{2}$$

Now to calculate $\mathbf{q_2}$, we have $\mathbf{a}_2{}^{\perp} = \mathbf{a}_2 - (r_{12})\mathbf{q}_1$, where $r_{12} = \mathbf{a}_2{}^T\mathbf{q}_1$

$$\implies r_{12} = \begin{bmatrix} 5 & -5 & -5 \end{bmatrix} \begin{bmatrix} \frac{-1}{\sqrt{21}} \\ \frac{-2}{\sqrt{21}} \\ \frac{4}{\sqrt{21}} \end{bmatrix} = \frac{-15}{\sqrt{21}} \tag{3}$$

Now, $\mathbf{a}_2{}^{\perp} = \mathbf{a}_2 - (r_{12})\mathbf{q}_1 = \begin{bmatrix} 5 \\ -5 \\ -5 \end{bmatrix}$ - $\frac{-15}{\sqrt{21}} \begin{bmatrix} \frac{-1}{\sqrt{21}} \\ \frac{-2}{\sqrt{21}} \\ \frac{4}{\sqrt{21}} \end{bmatrix}$

$$\implies \mathbf{a}_2{}^{\perp} = \frac{5}{7} \begin{bmatrix} 6 \\ -9 \\ -3 \end{bmatrix} \tag{4}$$

$$\implies ||\mathbf{a}_2{}^{\perp}|| = \frac{5}{7}\sqrt{6^2 + -9^2 + -3^2} = \frac{5}{7}\sqrt{126} = r_{22} \tag{5}$$

$$\implies \mathbf{q_2} = \frac{\mathbf{a_2}^{\perp}}{||\mathbf{a}_2{}^{\perp}||} = \frac{1}{\frac{5}{7}\sqrt{126}} \frac{5}{7} \begin{bmatrix} 6 \\ -9 \\ -3 \end{bmatrix} = \begin{bmatrix} \frac{6}{\sqrt{126}} \\ \frac{-9}{\sqrt{126}} \\ \frac{-3}{\sqrt{126}} \end{bmatrix} \tag{6}$$

Now to calculate $\mathbf{q_3}$, we have $\mathbf{a}_3{}^{\perp} = \mathbf{a}_3 - (r_{13})\mathbf{q}_1 - (r_{23})\mathbf{q}_2$, where $r_{13} = \mathbf{a}_3{}^T\mathbf{q}_1$ and $r_{23} = \mathbf{a}_3{}^T\mathbf{q}_2$

$$\implies r_{13} = \begin{bmatrix} -5 & -3 & 6 \end{bmatrix} \begin{bmatrix} \frac{-1}{\sqrt{21}} \\ \frac{-2}{\sqrt{21}} \\ \frac{4}{\sqrt{21}} \end{bmatrix} = \frac{35}{\sqrt{21}} \tag{7}$$

$$\implies r_{23} = \begin{bmatrix} -5 & -3 & 6 \end{bmatrix} \begin{bmatrix} \frac{6}{\sqrt{126}} \\ \frac{-9}{\sqrt{126}} \\ \frac{-3}{\sqrt{126}} \end{bmatrix} = \frac{-21}{\sqrt{126}} \tag{8}$$

Now we have, $\mathbf{a_3}^{\perp} = \mathbf{a}_3 - (r_{13})\mathbf{q}_1 - (r_{23})\mathbf{q}_2 = \begin{bmatrix} -5 \\ -3 \\ 6 \end{bmatrix} - \frac{35}{\sqrt{21}} \begin{bmatrix} \frac{-1}{\sqrt{21}} \\ \frac{-2}{\sqrt{21}} \\ \frac{4}{\sqrt{21}} \end{bmatrix} - \frac{-21}{\sqrt{126}} \begin{bmatrix} \frac{6}{\sqrt{126}} \\ \frac{-9}{\sqrt{126}} \\ \frac{-3}{\sqrt{126}} \end{bmatrix}$

$$\implies \mathbf{a_3}^{\perp} = \begin{bmatrix} \frac{-7}{3} \\ \frac{-7}{6} \\ \frac{-7}{6} \end{bmatrix} \tag{9}$$

$$\implies ||\mathbf{a_3}^{\perp}|| = \frac{7}{3}\sqrt{-1^2 + (\frac{-1}{2})^2 + (\frac{-1}{2})^2} = \frac{7}{\sqrt{6}} = r_{33} \tag{10}$$

$$\implies \mathbf{q_3} = \frac{\mathbf{a_3}^{\perp}}{||\mathbf{a_3}^{\perp}||} = \frac{1}{\frac{7}{\sqrt{6}}}\frac{5}{7} \begin{bmatrix} \frac{-7}{3} \\ \frac{-7}{6} \\ \frac{-7}{6} \end{bmatrix} = \begin{bmatrix} \frac{-\sqrt{2}}{\sqrt{3}} \\ \frac{-1}{\sqrt{6}} \\ \frac{-1}{\sqrt{6}} \end{bmatrix} \tag{11}$$

Finally, plugging the values obtained in equations (1) to (11), we get:

$$\mathbf{Q} = \begin{bmatrix} \frac{-1}{\sqrt{21}} & \frac{6}{\sqrt{126}} & \frac{\sqrt{-2}}{\sqrt{3}} \\ \frac{-2}{\sqrt{21}} & \frac{-9}{\sqrt{126}} & \frac{-1}{\sqrt{6}} \\ \frac{4}{\sqrt{21}} & \frac{-3}{\sqrt{126}} & \frac{-1}{\sqrt{6}} \end{bmatrix} \approx \begin{bmatrix} -0.218 & 0.534 & -0.817 \\ -0.436 & -0.801 & -0.408 \\ 0.873 & -0.267 & -0.408 \end{bmatrix}$$

$$\mathbf{R} = \begin{bmatrix} 4\sqrt{21} & \frac{-15}{\sqrt{21}} & \frac{35}{\sqrt{21}} \\ 0 & \frac{5\sqrt{126}}{7} & \frac{-21}{\sqrt{126}} \\ 0 & 0 & \frac{7}{\sqrt{6}} \end{bmatrix} \approx \begin{bmatrix} 18.33 & -3.27 & 7.63 \\ 0 & 8.01 & -1.87 \\ 0 & 0 & 2.85 \end{bmatrix}$$

Step 2: Solving $\mathbf{Ax} = \mathbf{b}$ by susbstituting $\mathbf{A} = \mathbf{QR}$

This is equivalent to solving

$$\mathbf{Rx} = \mathbf{Q^T b}$$

$$\underbrace{\begin{bmatrix} 4\sqrt{21} & \frac{-15}{\sqrt{21}} & \frac{35}{\sqrt{21}} \\ 0 & \frac{5\sqrt{126}}{7} & \frac{-21}{\sqrt{126}} \\ 0 & 0 & \frac{7}{\sqrt{6}} \end{bmatrix}}_{\mathbf{R}} \underbrace{\begin{bmatrix} \mathrm{x}_1 \\ \mathrm{x}_2 \\ \mathrm{x}_3 \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} \frac{-1}{\sqrt{21}} & \frac{-2}{\sqrt{21}} & \frac{4}{\sqrt{21}} \\ \frac{6}{\sqrt{126}} & \frac{-9}{\sqrt{126}} & \frac{-3}{\sqrt{126}} \\ \frac{-\sqrt{2}}{\sqrt{3}} & \frac{-1}{\sqrt{6}} & \frac{-1}{\sqrt{6}} \end{bmatrix}}_{\mathbf{Q^T}} \underbrace{\begin{bmatrix} -29 \\ -15 \\ 45 \end{bmatrix}}_{\mathbf{b}}$$

$$\begin{bmatrix} 4\sqrt{21} & \frac{-15}{\sqrt{21}} & \frac{35}{\sqrt{21}} \\ 0 & \frac{5\sqrt{126}}{7} & \frac{-21}{\sqrt{126}} \\ 0 & 0 & \frac{7}{\sqrt{6}} \end{bmatrix} \begin{bmatrix} \mathrm{x}_1 \\ \mathrm{x}_2 \\ \mathrm{x}_3 \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{239}{\sqrt{21}} \\ \frac{-174}{\sqrt{126}} \\ \frac{28}{\sqrt{6}} \end{bmatrix}}_{\mathbf{Q^T b}}$$

Using Backwards Substitution, we have,

$$\begin{cases} \frac{7}{\sqrt{6}}x_3 = \frac{28}{\sqrt{6}} \\ \frac{5\sqrt{126}}{7}x_2 - \frac{21}{\sqrt{126}}x_3 = \frac{-174}{\sqrt{126}} \\ 4\sqrt{21}x_1 + \frac{-15}{\sqrt{21}}x_2 + \frac{35}{\sqrt{21}}x_3 = \frac{239}{\sqrt{21}} \end{cases} \rightarrow \begin{cases} x_3 = 4 \\ x_2 = -1 \\ x_1 = 1 \end{cases}$$

HW1 - Q2: Eigenvalues and Eigenvectors (30 points)

Notes:

- Questions (a), (b) need to be typewritten.
- Questions (c), (d) need to be programmed.
- For typewritten solution:
  - Write all the steps of the solution .
  - Use proper LATEX formatting and notation for all mathematical equations, vectors, and matrices.
- For programming solution:
  - Properly add comments to your code.

(a) Write down the characteristic equation for matrix

$$A = \begin{bmatrix} 2 & 2 \\ 5 & -1 \end{bmatrix}.$$

Use the above characteristic equation to solve for eigenvalues and normalized eigenvectors of matrix A. (7 points)

Your answer here:

$$A = \begin{bmatrix} 2 & 2 \\ 5 & -1 \end{bmatrix}$$

Characteristic equation: $P_A(\lambda) = det(A - \lambda I)$

$$A = \begin{bmatrix} 2 & 2 \\ 5 & -1 \end{bmatrix} \rightarrow A - \lambda I = \begin{bmatrix} 2-\lambda & 2 \\ 5 & -1-\lambda \end{bmatrix}$$

$$\rightarrow det(A - \lambda I) = (2-\lambda)(-1-\lambda) - (2 \times 5) = -2 - 2\lambda + \lambda + \lambda^2 - 10 = \lambda^2 - \lambda - 12 = (\lambda+3)(\lambda-4)$$

$$\text{Eigenvalue } \lambda_1 = 4 \rightarrow (A - 4I)x = 0 \rightarrow \begin{bmatrix} -2 & 2 \\ 5 & -5 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0 \rightarrow= \begin{cases} -2x_1 + 2x_2 = 0 \\ 5x_1 - 5x_2 = 0 \end{cases} \rightarrow \begin{cases} x_2 = x_1 \\ 5x_1 - 5 \end{cases}$$

$$\rightarrow Length = \sqrt{1^2 + 1^2} = \sqrt{2} \rightarrow \text{Normalized eigenvector } \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\text{Eigenvalue } \lambda_2 = -3 \rightarrow (A + 3I)x = 0 \rightarrow \begin{bmatrix} 5 & 2 \\ 5 & 2 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0 \rightarrow= \begin{cases} 5x_1 + 2x_2 = 0 \\ 5x_1 + 2x_2 = 0 \end{cases} \rightarrow \begin{cases} x_2 = \frac{-5}{2}x_1 \\ 5x_1 + 2x_2 \end{cases}$$

$$\rightarrow Length = \sqrt{1^2 + (\frac{-5}{2})^2} = \frac{\sqrt{29}}{2} \rightarrow \text{Normalized eigenvector } \begin{bmatrix} 1/\frac{\sqrt{29}}{2} \\ \frac{-5}{2}/\frac{\sqrt{29}}{2} \end{bmatrix} = \begin{bmatrix} \frac{2}{\sqrt{29}} \\ \frac{-5}{\sqrt{29}} \end{bmatrix}$$

(b) Prove that if a real matrix $A_{n \times n}$ has unique eigenvalues (i.e. $\lambda_i \neq \lambda_j \, \forall \, i \neq j$), then the eigenvectors x_i are linearly independent. (7 points)

Hint: Prove by contradiction, start with $n = 2$ case.

Your answer here:

Given : Real matrix $A_{n \times n}$ has eigenvalues $\lambda_i \neq \lambda_j \; \forall \, i \neq j$

To Prove: The set of eigenvectors $X = \{x_1, x_2, \ldots, x_n\}$ is linearly independent

Proof by contradiction:

Assume set $\{x_1, x_2, \ldots, x_n\}$ is linearly dependent

$\implies$ we can express some eigenvector $x_k \in X$ as a linear combination of other eigenvectors in X with scalars c_i,

$$\implies x_k = \sum_{i=1}^{k-1} c_i x_i, \quad (c_i \neq 0, x_k \neq 0)$$

$$\implies x_k = c_1 x_1 + c_2 x_2 + \ldots + c_{k-1} x_{k-1} \quad (c_i \neq 0, x_k \neq 0) \tag{1}$$

Since x_i are the eigenvectors of A, we have

$$A x_i = \lambda_i x_i \tag{2}$$

By multiplying equation (1) by A, we get

$$A x_k = c_1 A x_1 + c_2 A x_2 + \ldots + c_{k-1} A x_{k-1} \tag{3}$$

Substituting values of $A x_i$ from equation (2) in equation (3), we get

$$\lambda_k x_k = c_1 \lambda_1 x_1 + c_2 \lambda_2 x_2 + \ldots + c_{k-1} \lambda_{k-1} x_{k-1} \tag{4}$$

Also, by multiplying equation (1) by λ_k, we get

$$\lambda_k x_k = c_1 \lambda_k x_1 + c_2 \lambda_k x_2 + \ldots + c_{k-1} \lambda_k x_{k-1} \tag{5}$$

Now, by subtracting equation (4) by equation (5), we get

$$\lambda_k x_k - \lambda_k x_k = c_1(\lambda_1 - \lambda_k) x_1 + c_2(\lambda_2 - \lambda_k x_2) + \ldots + c_{k-1}(\lambda_{k-1} - \lambda_k) x_{k-1}$$

$$\implies 0 = c_1(\lambda_1 - \lambda_k) x_1 + c_2(\lambda_2 - \lambda_k x_2) + \ldots + c_{k-1}(\lambda_{k-1} - \lambda_k) x_{k-1} \tag{6}$$

Now, since $\lambda_i \neq \lambda_j$, $\forall \, i \neq j$ and $x_i \neq 0$ because x_i are eigenvectors, this implies $c_1 = c_2 = \ldots = c_{k-1} = 0$

Then, using equation (1) we have $x_k = c_1 x_1 + c_2 x_2 + \ldots + c_{k-1} x_{k-1} = 0$ which is a contradiction to our initial assumption. This implies that x_k cannot be expressed as a linear combination of other eigenvectors in X. Finally, this implies that set $\{x_1, x_2, \ldots, x_n\}$ is linearly independent.

Hence proved.

(c) Write function `power_method(A,x)`, which takes as input matrix A and a vector `x`, and uses power method to calculate eigenvalue and eigenvector. Get the largest eigenvalue and eigenvector for matrix

$$A = \begin{bmatrix} 2 & 1 & 2 \\ 1 & 3 & 2 \\ 2 & 4 & 1 \end{bmatrix}$$

using above function. Start with intial eigenvector guesses: `[-1, 0.5, 3]` and `[2,-6,0.2]`. For each of the vectors, iterate until convergence. Plot how the eigenvalue changes w.r.t. iterations. Report the number of steps it took to converge, eigenvalue and eigenvector. Match your output with the results generated by the numpy API: `numpy.linalg.eig`

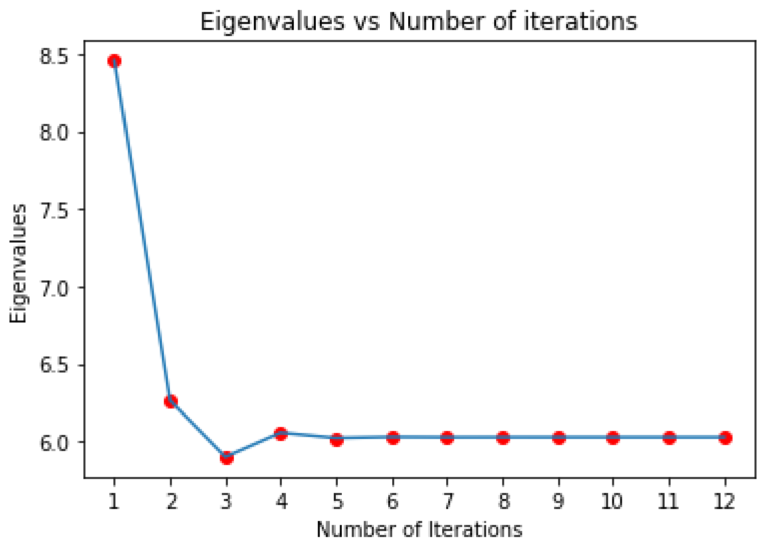
Note that you only need to look at magnitudes of eigenvalues. Use an absolute tolerance of 10^{-6} between eigenvalue output of previous and current iteration as stopping criteria. You may also need to normalize the final eigenvector to match with output of numpy API `numpy.linalg.eig`. (8 points)

In [7]:

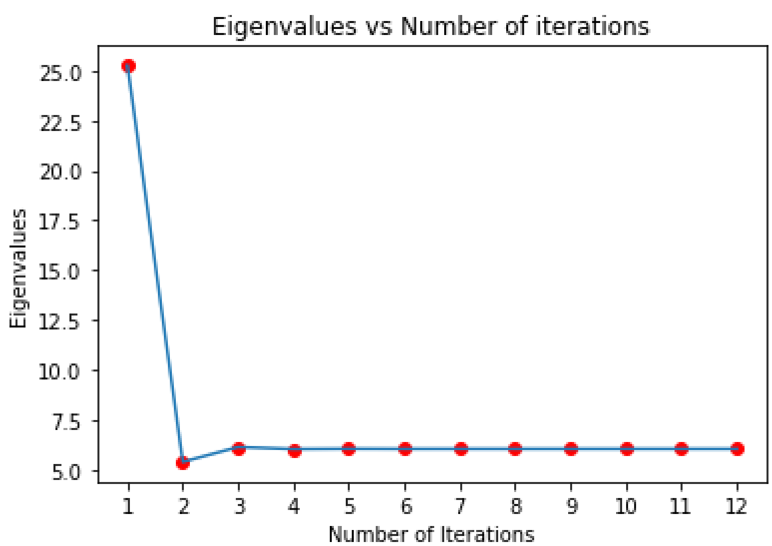
```
# !!!! YOUR CODE HERE !!!!

# importing libraries
import numpy as np
import matplotlib.pyplot as plt

def power_method(A, x):

    absTolerance = 10**(-6)
    oldEigenValue = 0
    curError = float("inf") # setting initial current error to positive
 infinity
    noOfSteps = 0  # to count number of steps
    eigenValueList = []  # to store iterative eigen values

    while (curError >= absTolerance):
        x = np.dot(A, x)  # matrix multiplication
        # newEigenValue = max(abs(x))  # new Eigenvalue
        newEigenValue =np.linalg.norm(x)  # new Eigenvalue
        x = x/newEigenValue  # normalizing x
        curError = abs(newEigenValue - oldEigenValue)  # calculating err
or
        eigenValueList.append(newEigenValue)
        oldEigenValue = newEigenValue
        noOfSteps += 1

    # Normalising eigenvector by the norm
    xNorm = np.linalg.norm(x)
    x = x/xNorm

    # printing eigenvalues and eigenvector after rounding the values to
 the first 6 decimal places
    print("Eigenvalue = ", round(newEigenValue, 6), ", Eigenvector = ",
[round(num, 6) for num in x],
          ", Number of iterations = ", noOfSteps)

    # Plotting Eigenvalues vs Number of iterations
    plt.title("Eigenvalues vs Number of iterations")
    plt.xlabel("Number of Iterations")
    plt.ylabel("Eigenvalues")
    plt.plot([i+1 for i in range(noOfSteps)], eigenValueList, 'ro')
    plt.plot([i+1 for i in range(noOfSteps)], eigenValueList)
    plt.xticks([i+1 for i in range(noOfSteps)])
    plt.show()


A = np.array([[2, 1, 2],
              [1, 3, 2],
              [2, 4, 1]])

x1 = np.array([-1, 0.5, 3])
x2 = np.array([2, -6, 0.2])

power_method(A, x1)
power_method(A, x2)
```

```
# using np.linalg.eig
w, v = np.linalg.eig(A)

#Sorting w in decreasing order to get the maximum eigenvalue (at index
 0) and corresponding eigenvector
w_abs = abs(w)
idx = w_abs.argsort()[: : -1]
w = w[idx]
v = v[:,idx]

print("Using np.linalg.eig, we have")
print("Eigenvalue = ", round(w[0], 6), ", Eigenvector = ", [round(num, 6
) for num in v[:, 0]])
print("Hence our values match in both cases")
```

```
Eigenvalue =  6.029112 , Eigenvector =  [0.471857, 0.58897, 0.656099] ,
Number of iterations =  12
```




```
Eigenvalue =  6.029112 , Eigenvector =  [-0.471858, -0.58897, -0.65609
9] , Number of iterations =  12
```




```
Using np.linalg.eig, we have
Eigenvalue =  6.029112 , Eigenvector =  [-0.471858, -0.58897, -0.65609
9]
Hence our values match in both cases
```

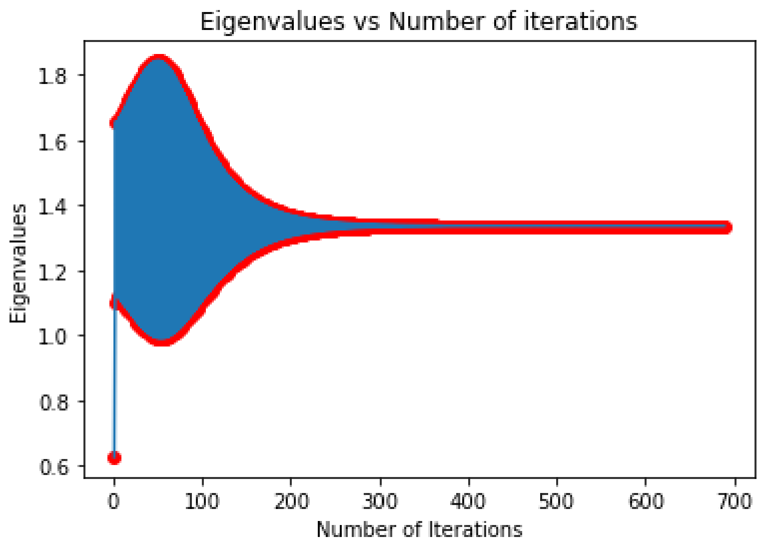
(d) Write function `inverse_power_method(A,x)`, which takes as input matrix A and a vector $\mathbf{x}$, and uses inverse power method to calculate the smallest eigenvalue and corresponding eigenvector. Solve for the smallest eigenvalue and corresponding eigenvector for the matrix from (c). Use the same intial eigenvector guesses as (c). Report how many iterations do you need for it to converge to the smallest eigenvalue. Plot the computed/estimated eigenvalue w.r.t iterations (keep in mind to plot $1/\lambda$ vs. number of iterations). Report the final eigenvalue and eigenvector you get. Match your answer with the results generated by the numpy API `numpy.linalg.eig`.

Note that you only need to look at magnitudes of eigenvalues. Use an absolute tolerance of 10^{-6} between eigenvalue output of previous and current iteration as stopping criteria. You may also need to normalize the final eigenvector to match with output of numpy API `numpy.linalg.eig`. (8 points)

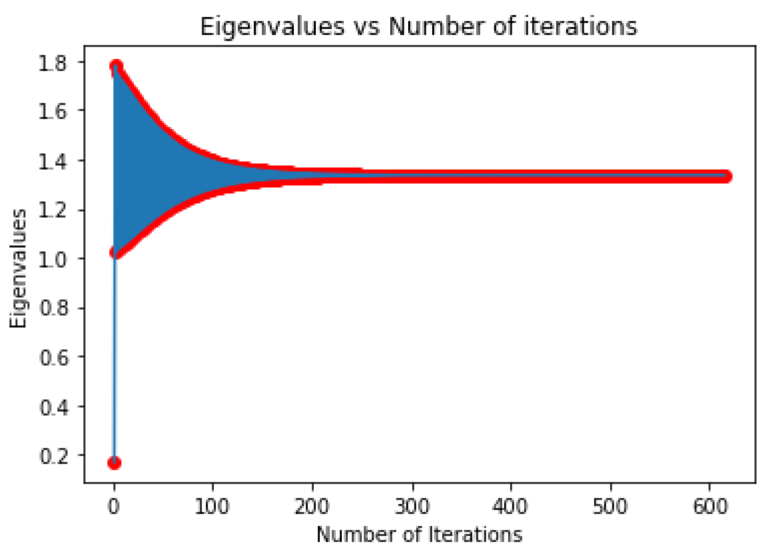
In [8]:
```python
# !!!! YOUR CODE HERE !!!!

# importing libraries
import numpy as np
import matplotlib.pyplot as plt

def inverse_power_method(A, x):

    absTolerance = 10**(-6)
    oldEigenValue = 0
    curError = float("inf") # setting initial current error to positive
 infinity
    noOfSteps = 0  # to count number of steps
    eigenValueList = []  # to store iterative eigen values

    while (curError >= absTolerance):

        Q, R = np.linalg.qr(A) # QR decomposition with qr function
        y = np.dot(Q.T, x) # Let y=Q'.x using matrix multiplication
        x = np.linalg.solve(R, y) # Solve Rx=y

        # newEigenValue = max(abs(x))  # new Eigenvalue
        newEigenValue = np.linalg.norm(x)  # new Eigenvalue
        x = x/newEigenValue  # normalizing x

        curError = abs(newEigenValue - oldEigenValue)  # calculating err
or

        eigenValueList.append(1/newEigenValue)
        oldEigenValue = newEigenValue
        noOfSteps += 1


    # Normalising eigenvector by the norm
    xNorm = np.linalg.norm(x)
    x = x/xNorm



    # printing eigenvalues and eigenvector after rounding the values to
 the first 6 decimal places
    print("Eigenvalue = ", round(1/newEigenValue, 6), ", Eigenvector = "
, [round(num, 6) for num in x],
          ", Number of iterations = ", noOfSteps)

    # Plotting Eigenvalues vs Number of iterations
    plt.title("Eigenvalues vs Number of iterations")
    plt.xlabel("Number of Iterations")
    plt.ylabel("Eigenvalues")
    plt.plot([i+1 for i in range(noOfSteps)], eigenValueList, 'ro')
    plt.plot([i+1 for i in range(noOfSteps)], eigenValueList)
    # plt.xticks([i+1 for i in range(noOfSteps)])
    plt.show()
```

```python
A = np.array([[2, 1, 2],
              [1, 3, 2],
              [2, 4, 1]])

x1 = np.array([-1, 0.5, 3])
x2 = np.array([2, -6, 0.2])

inverse_power_method(A, x1)
inverse_power_method(A, x2)

# using np.linalg.eig
w, v = np.linalg.eig(A)

#Sorting w in decreasing order to get the minimum eigenvalue (at index
 0) and corresponding eigenvector
w_abs = abs(w)
idx = w_abs.argsort()
w = w[idx]
v = v[:,idx]

print("Using np.linalg.eig, we have")
print("Eigenvalue = ", round(w[0], 6), ", Eigenvector = ", [round(num, 6
) for num in v[:, 0]])
print("Hence our values match in both cases")
```

```
Eigenvalue =  1.336257 , Eigenvector =  [-0.889875, 0.450815, 0.069916]
, Number of iterations =  689
```




```
Eigenvalue =  1.336255 , Eigenvector =  [0.889875, -0.450815, -0.06991
8] , Number of iterations =  615
```




```
Using np.linalg.eig, we have
Eigenvalue =  1.336256 , Eigenvector =  [-0.889875, 0.450815, 0.069917]
Hence our values match in both cases
```

HW1 - Q3: Face Recognition with Eigenfaces (40 points)

Keywords: Principal Component Analysis (PCA), Eigenvalues and Eigenvectors

About the dataset: \ [*Labeled Faces in the Wild* (http://vis-www.cs.umass.edu/lfw/)](http://vis-www.cs.umass.edu/lfw/) dataset consists of face photographs designed for studying the problem of unconstrained face recognition. The original dataset contains more than 13,000 images of faces collected from the web.

Agenda:

- In this programming challenge, you will be performing face recognition on the *Labeled Faces in the Wild* dataset using PyTorch.
- First, you will do Principal Component Analysis (PCA) on the image dataset. PCA is used for dimentionality reduction which is a type of unsupervised learning.
- You will be applying PCA on the dataset to extract the principal components (Top k *eigenvalues*).
- As you will see eventually, the reconstruction of faces from these *eigenvalues* will give us the *eigen-faces* which are the most representative features of most of the images in the dataset.
- Finally, you will train a simple PyTorch Neural Network model on the modified image dataset.
- This trained model will be used prediction and evaluation on a test set.

Note:

- Run all the cells in order.
- **Do not edit** the cells marked with !!DO NOT EDIT!!
- Only **add your code** to cells marked with !!!! YOUR CODE HERE !!!!
- Do not change variable names, and use the names which are suggested.

In [71]:

```
# !!DO NOT EDIT!!
# loading the dataset directly from the scikit-learn library (can take a
bout 3-5 mins)
import numpy as np
from sklearn.datasets import fetch_lfw_people
dataset = fetch_lfw_people(min_faces_per_person=80)

# each 2D image is of size 62 x 47 pixels, represented by a 2D array.
# the value of each pixel is a real value from 0 to 255.
count, height, width = dataset.images.shape
print('The dataset type is:',type(dataset.images))
print('The number of images in the dataset:',count)
print('The height of each image:',height)
print('The width of each image:',width)

# sklearn also gives us a flattened version of the images which is a vec
tor of size 62 x 47 = 2914.
# we can directly use that for our exercise
print('The shape of data is:',dataset.data.shape)
```

```
The dataset type is: <class 'numpy.ndarray'>
The number of images in the dataset: 1140
The height of each image: 62
The width of each image: 47
The shape of data is: (1140, 2914)
```

For optimum performance, we have only considered people who have more than 80 images. This restriction notably reduces the size of the dataset.\ Now let us look at the labels of the people present in the dataset

In [72]:

```
# !!DO NOT EDIT!!
# create target label - target name pairs
targets = [(x,y) for x,y in zip(range(len(np.unique(dataset.target))), d
ataset.target_names)]
print('The target labels and names are:\n', targets)
```

```
The target labels and names are:
 [(0, 'Colin Powell'), (1, 'Donald Rumsfeld'), (2, 'George W Bush'),
(3, 'Gerhard Schroeder'), (4, 'Tony Blair')]
```

(a) Preprocessing: Using the `train_test_split` API from sklearn, split the data into train and test dataset in the ratio 3:1. Use `random_state=42`.

For better performance, it is recommended to normalize the features which can have different ranges with huge values. As all our features here are in the range [0,255], it is not explicitly needed here. However, it is a good exercise. Use the `StandardScaler` class from sklearn and use that to normalize X_train and X_test. Validate and show your result by printing the first 5 columns of 5 images of X_train (This result can vary from pc to pc). (5 points)

In [73]:

```
# !!DO NOT EDIT!!
X = dataset.data
y = dataset.target
```

In [74]:

```
#######
# !!!! YOUR CODE HERE !!!!
#Importing required utilities
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

#Splitting data into training and testing sets in the ratio 3:1
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25
, random_state=42)

#Normalizing X_train and X_test
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Printing first 5 columns of 5 images of X_train
print(X_train[:5, :5])

# output variable names -  X_train, X_test, y_train, y_test
#######
```

```
[[-0.9581398  -1.0216656  -1.0032063  -0.7196299  -0.5934948 ]
 [ 2.532961    2.3217816   1.5588257   0.757766    0.2888807 ]
 [-1.07198    -1.0914823  -1.1301181  -0.93524987  1.2193854 ]
 [ 0.08919033 -0.02871611 -0.53521895 -1.1588557  -1.0747904 ]
 [ 0.02847559 -0.02871611 -0.08309564 -0.21651669 -0.07209121]]
```

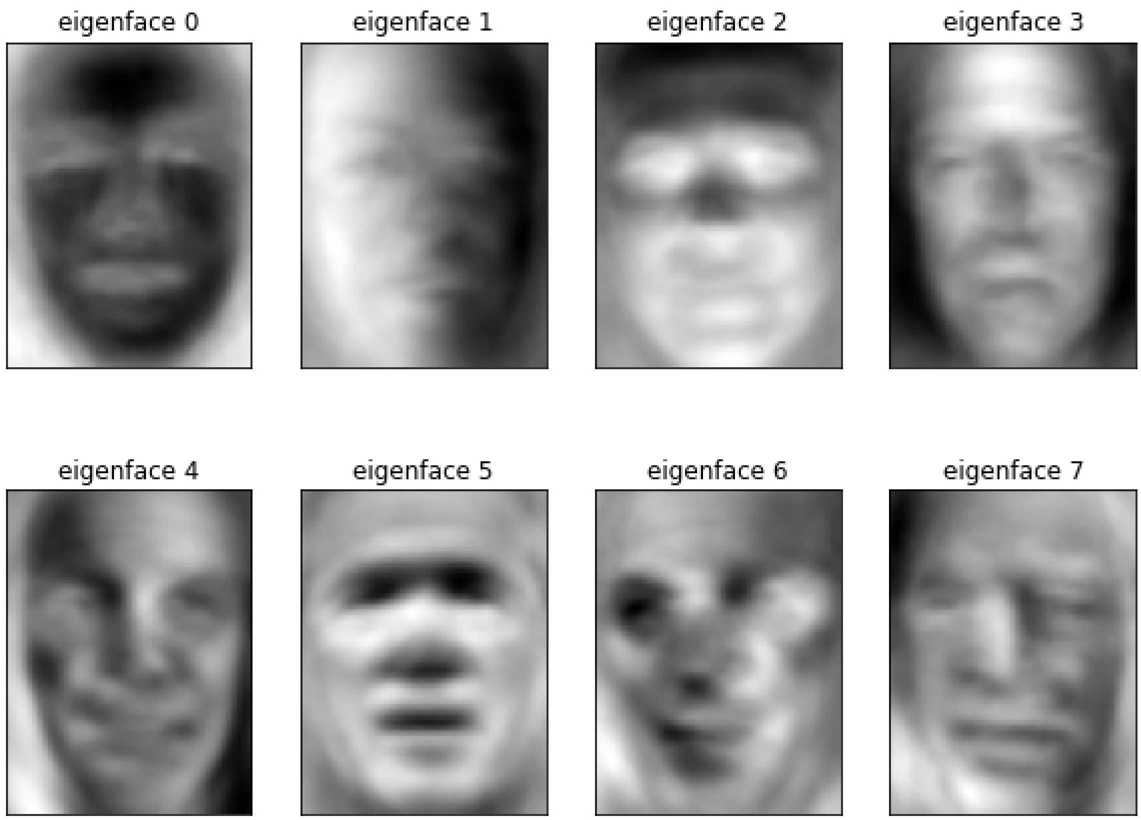
(b) Dimentionality reduction : In this section, use the `PCA` API from sklearn to extract the top 100 principal components of the image matrix and fit it on the training dataset. We can then visualize some of the top few components as an image (eigenfaces). (5 points)

In [75]:
```
#######
# !!!! YOUR CODE HERE !!!!
# initialize PCA API from sklearn with n_components. Also set svd_solver
="randomized" and whiten=True in the initialization parameters.

#importing PCA
from sklearn.decomposition import PCA

#fitting PCA
n_components = 100
pca = PCA(n_components=n_components, svd_solver="randomized", whiten=Tru
e).fit(X_train)

# output variable name -  pca
#######
```

Now we will plot the most representative eigenfaces:

In [76]:
```
# !!DO NOT EDIT!!
# Helper function to plot
import matplotlib.pyplot as plt
def plot_gallery(images, titles, height, width, n_row=2, n_col=4):
    plt.figure(figsize=(2* n_col, 3 * n_row))
    plt.subplots_adjust(bottom=0, left=0.01, right=0.99, top=0.90, hspac
e=0.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((height, width)), cmap=plt.cm.gray)
        plt.title(titles[i], size=12)
        plt.xticks(())
        plt.yticks(())
```

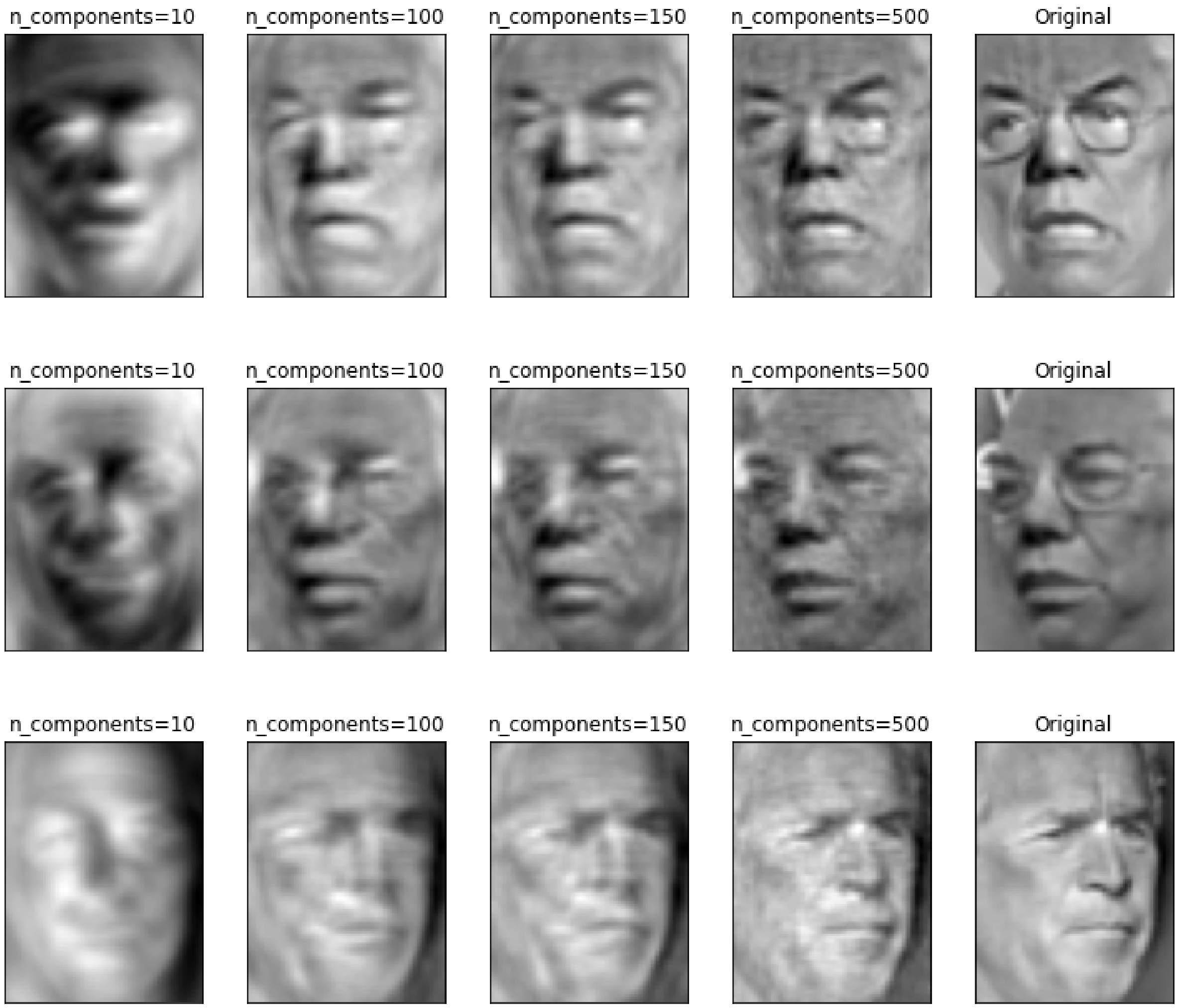
```
In [77]: # !!DO NOT EDIT!!
         # get the 100 eigen faces and reshape them to original image size which
          is 62 x 47 pixels
         eigenfaces = pca.components_.reshape((n_components, height, width))

         # plot the top 8 eigenfaces
         eigenface_titles = ["eigenface %d" % i for i in range(eigenfaces.shape[0
         ])]
         plot_gallery(eigenfaces, eigenface_titles, height, width)

         plt.show()
```



(c) Face reconstruction: In this section, we will reconstruct an image from its point projected on the principal component basis. Project the first three faces on the eigenvector basis using PCA models trained with varying number of principal components. Using the projected points, reconstruct the faces, and visualize the images. Your final output should be a 3×5 image matrix, where the rows are the data points, and the columns correspond to original image and reconstructed image for n_components= $[10, 100, 150, 500]$. (15 points)

In [78]:

```
#######
# !!!! YOUR CODE HERE !!!!

n_components_list = [10, 100, 150, 500]
no_of_faces = 3

# function to plot faces
def plot_eigenface(x):
    plt.imshow(x.reshape((height, width)), cmap=plt.cm.gray)
    plt.xticks([])
    plt.yticks([])

# fitting PCA for 500 (max) components
max_n_component = max(n_components_list)
pca2 = PCA(n_components = max_n_component, svd_solver="randomized", whit
en=True).fit(X_train)

# Applying PCA to X_test
Z = pca2.transform(X_test)

# Setting figure size
n_cols = len(n_components_list)+1
n_rows = no_of_faces
plt.figure(figsize=(2 * n_cols, 3 * n_rows))
plt.subplots_adjust(bottom=0, left=0.01, right=0.99, top=0.90, hspace=0.
35)

#Looping over figures
count = 0
for i in range(no_of_faces):
    for n in n_components_list:
        plt.subplot(no_of_faces,n_cols, count+1)

        # Creating a copy of the matrix and setting all coefficients aft
er n to 0
        Z_copy = np.copy(Z[i,:])
        Z_copy[n:] = 0

        # Reconstructing the face
        Z_inverse = pca2.inverse_transform(Z_copy)

        # Plotting the reconstructed face
        plot_eigenface(Z_inverse)
        plt.title('n_components={}'.format(n))
        count += 1

    # Plotting the original face
    plt.subplot(no_of_faces, n_cols, count + 1)
    plot_eigenface(X_test[i,:])
    plt.title('Original')
    count += 1

#######
```



(d) Prediction: In this section, we will train a neural network classifier in PyTorch on the transformed dataset. This classifier will help us with the face recognition task. Complete each of the steps below.

For PyTorch reference see [documentation (https://pytorch.org/docs/stable/index.html)](https://pytorch.org/docs/stable/index.html). (15 points)

In [79]:

```
# !!DO NOT EDIT!!
# define imports here
import torch
import torch.nn as nn
```

Before we start training, we need to transform the training and test dataset to reduced forms (100 dimensions) using the pca function defined in (b).

we will also need to move the train and test dataset to torch tensors in order to work with pytorch.

In [80]:
```
#######
# !!!! YOUR CODE HERE !!!!
# 1. project X_train and X_test on orthonormal basis using the PCA API i
nitialized in part (b).
n_components = 100

# pca initialized in part (b)
# pca = PCA(n_components = n_components, svd_solver="randomized", whiten
=True).fit(X_train)

X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

# 2. now convert X_train_pca, X_test_pca, y_train and y_test to torch.te
nsor. For y_train and y_test, set dtype=torch.long
X_train_pca_torch = torch.tensor(X_train_pca)
X_test_pca_torch = torch.tensor(X_test_pca)
y_train_torch = torch.tensor(y_train, dtype=torch.long)
y_test_torch = torch.tensor(y_test, dtype=torch.long)

# output variable names -  X_train_pca_torch, X_test_pca_torch, y_train_
torch, y_test_torch
#######
```

In [81]:
```
from torch.nn.modules.activation import LogSoftmax
#######
# !!!! YOUR CODE HERE !!!!
# 3. We will implement a simple multilayer perceptron (MLP) in pytorch w
ith one hidden layer.
# Using this neural network model, we will train on the transformed data
set.
class MLP(torch.nn.Module):
  def __init__(self):
    super(MLP, self).__init__()
    # Initalize various layers of MLP as instructed below
    # DO: initialze two linear layers: 100 -> 1024  and 1024-> 5
    # DO: initialize relu activation function
    # DO: initialize LogSoftmax
    self.layers = nn.Sequential(
      nn.Linear(100, 1024),
      nn.Linear(1024,5),
      nn.ReLU(),
      nn.LogSoftmax()
    )

  def forward(self, x):
    # DO: define the feedforward algorithm of the model and return the f
inal output
    return self.layers(x)

#######
```

In [82]:
```
#######
# !!!! YOUR CODE HERE !!!!
# 4. create an instance of the MLP class here
model = MLP()

# 5. define loss (use negative log likelihood loss: torch.nn.NLLLoss)
criterion = torch.nn.NLLLoss()

# 6. define optimizer (use torch.optim.SGD (Stochastic Gradient Descen
t)).
# Set learning rate to 1e-1 and also set model parameters
optimizer = torch.optim.SGD(model.parameters(), lr = 1e-1)

#######

# !!DO NOT EDIT!!
# 7. train the classifier on the PCA-transformed training data for 500 e
pochs
# This part is already implemented.
# Go through each step carefully and understand what it does.
for epoch in range(501):
  # reset gradients
  optimizer.zero_grad()

  # predict
  output=model(X_train_pca_torch)

  # calculate loss
  loss=criterion(output, y_train_torch)

  # backpropagate loss
  loss.backward()

  # performs a single gradient update step
  optimizer.step()

  if epoch%50==0:
    print('Epoch: {}, Loss: {:.3f}'.format(epoch, loss.item()))
```

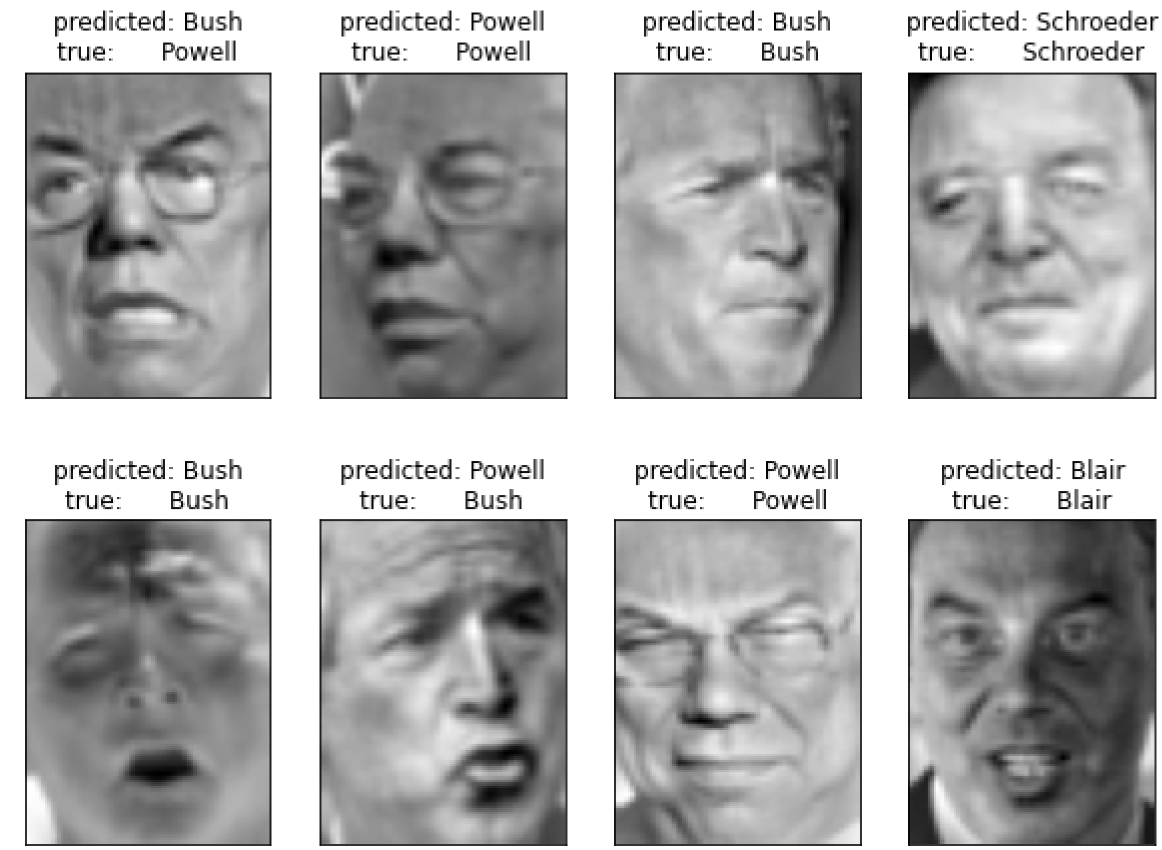
```
c:\Users\shrey\Anaconda3\lib\site-packages\torch\nn\modules\container.p
y:141: UserWarning: Implicit dimension choice for log_softmax has been
deprecated. Change the call to include dim=X as an argument.
  input = module(input)

Epoch: 0, Loss: 1.623
Epoch: 50, Loss: 0.305
Epoch: 100, Loss: 0.207
Epoch: 150, Loss: 0.164
Epoch: 200, Loss: 0.138
Epoch: 250, Loss: 0.120
Epoch: 300, Loss: 0.107
Epoch: 350, Loss: 0.096
Epoch: 400, Loss: 0.088
Epoch: 450, Loss: 0.081
Epoch: 500, Loss: 0.076
```

In [83]:

```
# !!DO NOT EDIT!!
# predict on test data
predictions = model(X_test_pca_torch) # gives softmax logits
y_pred = torch.argmax(predictions, dim=1).numpy() # get the labels from
 prdictions: nx5 -> nx1
```

In [84]:
```
# !!DO NOT EDIT!!
# here, we will print the multi-label classification report: precision,
 recall, f1-score etc.
from sklearn.metrics import classification_report
target_names=[y for x,y in targets]
print(classification_report(y_test, y_pred, target_names=target_names))

# let us validate some of the predictions by plotting images
# display some of the results
def title(y_pred, y_test, target_names, i):
    pred_name = target_names[y_pred[i]].rsplit(" ", 1)[-1]
    true_name = target_names[y_test[i]].rsplit(" ", 1)[-1]
    return "predicted: %s\ntrue:      %s" % (pred_name, true_name)

prediction_titles = [
    title(y_pred, y_test, target_names, i) for i in range(y_pred.shape[0
])
]

plot_gallery(X_test, prediction_titles, height, width)
```

```
                   precision    recall  f1-score   support

     Colin Powell       0.85      0.86      0.85        64
  Donald Rumsfeld       0.82      0.84      0.83        32
    George W Bush       0.89      0.92      0.91       127
Gerhard Schroeder       0.88      0.76      0.81        29
       Tony Blair       0.90      0.85      0.88        33

         accuracy                           0.87       285
        macro avg       0.87      0.85      0.86       285
     weighted avg       0.87      0.87      0.87       285
```



4. Google Pagerank Algorithm (10 points)

Keywords: Pagerank, Power Method

About the dataset: \ *DBpedia* (from "DB" for "database") is a project aiming to extract structured content from the information created in the Wikipedia project. This structured information is made available on the World Wide Web. DBpedia allows users to semantically query relationships and properties of Wikipedia resources, including links to other related datasets. for more info, see: [https://en.wikipedia.org/wiki/DBpedia (https://en.wikipedia.org/wiki/DBpedia)](https://en.wikipedia.org/wiki/DBpedia). \ We will download two files from the data respository:

- The first file -- **redirects_en.nt.bz2** -- contains redirects link for a page. Let A redirect to B and B redirect to C. Then we will replace article A by article C wherever needed.
- The second file -- **page_links_en.nt.bz2** -- contains pagelinks which are links within an article to other wiki article.

Note that the data is both files is a list of lines which can be split into 4 parts:

- The link to first article.
- Whether it is a redirect, or a pagelink.
- The link to second article.
- A fullstop.

Note: Any line which cannot be split into 4 parts is skipped from consideration.

Agenda:

- In this programming challenge, you will be implementing the [*google pagerank algorithm* (https://towardsdatascience.com/pagerank-algorithm-fully-explained-dc794184b4af)](https://towardsdatascience.com/pagerank-algorithm-fully-explained-dc794184b4af) to determine the most important articles.
- This will be done by computing the principal eigenvector of the article-article graph adjacency matrix.
- In this challenge, you will be applying the *power method* to extract the principal eigenvector from the adjacency matrix.
- Using the computed eigenvector, we can assign each article a eigenvector-centrality score. Then we can determine the most important articles.

Environment: Ensure following libraries are installed

- sklearn
- numpy

Also ensure that you have around **4 GB** of memory.

Note:

- Run all the cells in order.
- **Do not edit** the cells marked with !!DO NOT EDIT!!
- Only **add your code** to cells marked with !!!! YOUR CODE HERE !!!!
- Do not change variable names, and use the names which are suggested.

In [3]:
```
# !! DO NOT EDIT !!
# imports
import pickle
from bz2 import BZ2File
import bz2
import os
from datetime import datetime
import pprint
from time import time
import numpy as np
from urllib.request import urlopen
import scipy.sparse as sparse
pp = pprint.PrettyPrinter(indent=4)
```

In [4]:
```
# !! DO NOT EDIT !!
# download the dataset and store files in local

# dbpedia download urls
redirects_url = "http://downloads.dbpedia.org/3.5.1/en/redirects_en.nt.b
z2"
page_links_url = "http://downloads.dbpedia.org/3.5.1/en/page_links_en.n
t.bz2"

# extarct the file-names from the urls. Needed to load the files later
redirects_filename = redirects_url.rsplit("/", 1)[1] # redirects_en.nt.b
z2 ~ 59MB
page_links_filename = page_links_url.rsplit("/", 1)[1] # page_links_en.n
t.bz2 ~ 850MB

resources = [
    (redirects_url, redirects_filename),
    (page_links_url, page_links_filename),
]

# download the files
# this will take some time
for url, filename in resources:
    if not os.path.exists(filename):
        print("Downloading data from '%s', please wait..." % url)
        opener = urlopen(url)
        open(filename, "wb").write(opener.read())
        print()
```

In [5]:
```
# !! DO NOT EDIT !!
# load the data from the downloaded files
# this may take some time

#read redirects file
redirects_file = bz2.open(redirects_filename, mode='rt')
redirects_data = redirects_file.readlines()
redirects_file.close()

# pagelinks data has 119M entries
# We will only consider the first 5M for this question
pagelinks_file = bz2.open(page_links_filename, mode='rt')
pagelinks_data = [next(pagelinks_file) for x in range(5000000)]
pagelinks_file.close()
```

In [6]:
```
# !! DO NOT EDIT !!
# look at the size of the data and some examples
print ('The number of entries in redirects:', len(redirects_data))
print ('A couple of examples from redirects:')
print (redirects_data[10000:10002])
print ('\n')

print ('The number of entries in pagelinks:', len(pagelinks_data))
print ('A couple of examples from pagelinks:')
print (pagelinks_data[100000:100002])
```

```
The number of entries in redirects: 4082533
A couple of examples from redirects:
['<http://dbpedia.org/resource/Proper_superset> <http://dbpedia.org/pro
perty/redirect> <http://dbpedia.org/resource/Subset> .\n', '<http://dbp
edia.org/resource/Jean_Paul_Sartre> <http://dbpedia.org/property/redire
ct> <http://dbpedia.org/resource/Jean-Paul_Sartre> .\n']


The number of entries in pagelinks: 5000000
A couple of examples from pagelinks:
['<http://dbpedia.org/resource/Antipope> <http://dbpedia.org/property/w
ikilink> <http://dbpedia.org/resource/Council_of_Constance> .\n', '<htt
p://dbpedia.org/resource/Antipope> <http://dbpedia.org/property/wikilin
k> <http://dbpedia.org/resource/Pope_Alexander_V> .\n']
```

Note: It is worth noting here that each article is uniquely represented by its URL, or rather, the last segment of its URL

(a) Define a function `get_article_name` which takes as input the URL string, and extracts the last segment of the URL, which we can call as article name. (1 point)

In [7]:
```
#######
# !!!! YOUR CODE HERE !!!!
len_of_prefix = len("http://dbpedia.org/resource/")
last_segment_slice = slice(len_of_prefix + 1, -1)

def get_article_name(url):
  return url[last_segment_slice]

#######
```

In [8]:
```
# !! DO NOT EDIT !!
# some unit tests to validate your solution
assert get_article_name('<http://dbpedia.org/resource/Pope_Alexander_V>'
) == 'Pope_Alexander_V'
assert get_article_name('<http://dbpedia.org/resource/Jean-Paul_Sartre>'
) == 'Jean-Paul_Sartre'
```

(b) Define a function `resolve_redirects` which takes as input a list of redirect lines, and returns a map between the initial and the resolved redirect page. (2 points)

e.g.: input = \ ['\http://dbpedia.org/resource/A (http://dbpedia.org/resource/A) \http://dbpedia.org/property/redirect (http://dbpedia.org/property/redirect) \http://dbpedia.org/resource/B (http://dbpedia.org/resource/B) .\n',\ '\http://dbpedia.org/resource/B (http://dbpedia.org/resource/B) \http://dbpedia.org/property/redirect (http://dbpedia.org/property/redirect) \http://dbpedia.org/resource/C (http://dbpedia.org/resource/C) .\n',\ '\http://dbpedia.org/resource/C (http://dbpedia.org/resource/C) \http://dbpedia.org/property/redirect (http://dbpedia.org/property/redirect) \http://dbpedia.org/resource/D (http://dbpedia.org/resource/D) .\n',\ '\http://dbpedia.org/resource/X (http://dbpedia.org/resource/X) \http://dbpedia.org/property/redirect (http://dbpedia.org/property/redirect) \http://dbpedia.org/resource/Z (http://dbpedia.org/resource/Z) .\n']

output = {'A': 'D', 'B': 'D', 'C': 'D', 'X': 'Z'}

Note: Remember to ignore malformed lines which are those which do not split in 4 parts.

In [9]:
```
#######
# !!!! YOUR CODE HERE !!!!
def resolve_redirects(redirects_url):
  output = {}

  for url in redirects_url:
    # split the redirect url into 3 parts
    components = url.split(' ')

    # ignoring malformed lines
    if len(components) != 4:
      continue

    # the starting redirect is the first element of components list
    start = get_article_name(components[0])

    #the ending redirect is the second to last element of components lis
t
    end = get_article_name(components[-2])

    # if a line redirects to itself, we ignore it
    if start == end:
      continue

    # add start end to dictionary
    if start not in output:
      output[start] = end

  print("Updating immediate redirects to final redirects")
  for i, start in enumerate(output.keys()):
    final_redirect = None
    immediate_redirect = output[start]
    alreadySeen = {start}

    while True:
        final_redirect = immediate_redirect
        immediate_redirect = output.get(immediate_redirect)
        if immediate_redirect is None or immediate_redirect in alreadySe
en:
            break
        alreadySeen.add(immediate_redirect)

    output[start] = final_redirect

    # printing checkpoint
    if i % 1000000 == 0:
        print("Completed Line: ", i)

  return output

#######
```

In [10]:
```
# !! DO NOT EDIT !!
# some unit tests to validate your solution
test_input = ['<http://dbpedia.org/resource/A> <http://dbpedia.org/prope
rty/redirect> <http://dbpedia.org/resource/B> .\n',
              '<http://dbpedia.org/resource/B> <http://dbpedia.org/prope
rty/redirect> <http://dbpedia.org/resource/C> .\n',
              '<http://dbpedia.org/resource/C> <http://dbpedia.org/prope
rty/redirect> <http://dbpedia.org/resource/D> .\n',
              '<http://dbpedia.org/resource/X> <http://dbpedia.org/prope
rty/redirect> <http://dbpedia.org/resource/Z> .\n']

test_output = {'A': 'D', 'B': 'D', 'C': 'D', 'X': 'Z'}

assert resolve_redirects(test_input) == test_output
```

```
Updating immediate redirects to final redirects
Completed Line:  0
```

(c) Create article-article adjacency matrix.

Let the number of articles n. The adjacency matrix should have a value $A[i][j] = 1$ if there is a link from i to j. Note that the matrix may not be symmetric. This matrix would have rows as source, and columns as destinations. However, for further sections, we need it the other way round. Therefore, return $A^\top$ matrix.

Create a function `make_adjacency_matrix` that takes as input the resolved redirect map from part (b), and the list from `pagelinks_data`.

Return a tuple of `(index_map, A')`, where `index_map` is a map of each article to a unique number between 0 and $n-1$, also its unique numerical id. `A` is the adjacency matrix in [scipy.sparse.csr_matrix (https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.htm](https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html) format. `A'` is the transpose of matrix `A`. (2 points)

Note: Take care that if A redirects to D and X redirects to Y, and there is a pagelink entry from A to X, then the resolved pagelink entry should be D to Y.

In [11]:

```
#######
# !!!! YOUR CODE HERE !!!!

def get_index(resolved_redirects, index_map, n):
    # Find and return unique integer index of article names after resolv
ing redirects
    n = resolved_redirects.get(n, n)
    return index_map.setdefault(n, len(index_map))


def make_adjacency_matrix(resolved_redirects, page_links_data):

    # Computing the Index Map
    print("Computing the Index Map")
    index_map = dict()
    list_of_links = list()

    for k, line in enumerate(page_links_data):
        segment = line.split()
        if len(segment) != 4:
            continue
        i = get_index(resolved_redirects, index_map, get_article_name(se
gment[0]))
        j = get_index(resolved_redirects, index_map, get_article_name(se
gment[2]))
        list_of_links.append((i, j))

        # checkpoint
        if k % 1000000 == 0:
            print("Completed Line:", k)

    # Computing the Adjacency Matrix
    print("Computing the Adjacency Matrix")
    X = sparse.lil_matrix((len(index_map), len(index_map)), dtype=np.flo
at32)

    for i, j in list_of_links:
        X[i, j] = 1.0
    del list_of_links

    # Converting to CSR representation
    X = X.tocsr()
    # Taking transpose of Adjacency Matrix
    X = X.transpose()

    # returning Index Map and Adjacency Matrix
    return index_map, X

#######
```

In [12]:
```python
# !! DO NOT EDIT !!
# some unit tests to validate your solution

test_redirects = {'A': 'D', 'B': 'D', 'C': 'D', 'X': 'Z', 'K':'L', 'M':
'N'}
test_pagelinks_data = ['<http://dbpedia.org/resource/A> <http://dbpedia.
org/property/wikilink> <http://dbpedia.org/resource/X> .\n', '<http://db
pedia.org/resource/L> <http://dbpedia.org/property/wikilink> <http://dbp
edia.org/resource/N> .\n', '<http://dbpedia.org/resource/P> <http://dbpe
dia.org/property/wikilink> <http://dbpedia.org/resource/Q> .\n']

test_output_index_map = {'D': 0, 'Z': 1, 'L': 2, 'N': 3, 'P': 4, 'Q': 5}
test_output_adjacency_matrix = np.array([[0., 1., 0., 0., 0., 0.],
                                         [0., 0., 0., 0., 0., 0.],
                                         [0., 0., 0., 1., 0., 0.],
                                         [0., 0., 0., 0., 0., 0.],
                                         [0., 0., 0., 0., 0., 1.],
                                         [0., 0., 0., 0., 0., 0.]])

output_index_map, output_A = make_adjacency_matrix(test_redirects, test_
pagelinks_data)

assert output_index_map == test_output_index_map
np.testing.assert_array_equal(output_A.toarray(), test_output_adjacency_
matrix.T)
```

```
Computing the Index Map
Completed Line: 0
Computing the Adjacency Matrix
```

(d) Apply the above functions on the dataset to create adjacency matrix A and other relevant maps as directed below. Then apply `SVD` from sklearn on the adjacency matrix to determine principal singular vectors. (2 points)

In [13]:
```
#######
# !!!! YOUR CODE HERE !!!!
# 1. with redirects_data as input, use the resolve_redirects function to
generate the redirects_map
# redirects_map = _________
redirects_map = resolve_redirects(redirects_data)

# 2. with redirects map from previous step pagelinks_data as inputs, use
the make_adjacency_matrix to generate index_map and adjacency_matrix
# index_map, X = _________
index_map, X = make_adjacency_matrix(redirects_map, pagelinks_data)

# 3. using index_map, create a reverse_index_map, which has the article
 name as key, and its index as value
# reverse_index_map = _________
reverse_index_map = {i: article_name for article_name, i in index_map.it
ems()}
#######
```

```
Updating immediate redirects to final redirects
Completed Line:  0
Completed Line:  1000000
Completed Line:  2000000
Completed Line:  3000000
Completed Line:  4000000
Computing the Index Map
Completed Line: 0
Completed Line: 1000000
Completed Line: 2000000
Completed Line: 3000000
Completed Line: 4000000
Computing the Adjacency Matrix
```

In [14]:
```
# !! DO NOT EDIT !!
# Now we will save the csr matrix, index_map and reverse_index_map in pi
ckle files
# so that we do not have to recompute steps (a)-(d) next time we load th
e program
# (Note: beneficial only when working on local machine, as colab session
times out)
PATH='./'
pickle.dump(X, open(PATH+'X.pkl', 'wb'))
pickle.dump(index_map, open(PATH+'index_map.pkl', 'wb'))
pickle.dump(reverse_index_map, open(PATH+'reverse_index_map.pkl', 'wb'))

# free up RAM
del(redirects_data, pagelinks_data)
```

! ---------- Checkpoint ----------- !

In [15]:
```
# !! DO NOT EDIT !!
# Load the data from here
PATH='./'
X = pickle.load(open(PATH+'X.pkl', 'rb'))
index_map = pickle.load(open(PATH+'index_map.pkl', 'rb'))
reverse_index_map =  pickle.load(open(PATH+'reverse_index_map.pkl', 'rb'
))
```

Apply `randomized_svd` from sklearn on the adjacency matrix. Extract top 5 components and run for 3 iterations.

In [16]:
```
#######
# !!!! YOUR CODE HERE !!!!
# U, s, V = __________
from sklearn.decomposition import randomized_svd
U, s, V = randomized_svd(X, 5, n_iter=3)

#######
```

In [17]:
```
# !! DO NOT EDIT !!
# now, we print the names of the wikipedia related strongest components
 of the
# principal singular vector which should be similar to the highest eigen
vector
print("Top wikipedia pages according to principal singular vectors")
pp.pprint([reverse_index_map[i] for i in np.abs(U.T[0]).argsort()[-10
:]])
pp.pprint([reverse_index_map[i] for i in np.abs(V[0]).argsort()[-10:]])
```

```
Top wikipedia pages according to principal singular vectors
[   'England',
    'Spain',
    'Italy',
    'Canada',
    'Japan',
    'Germany',
    'World_War_II',
    'France',
    'United_Kingdom',
    'United_States']
['1989', '1971', '1975', '1970', '2006', '1972', '1996', '1966', '196
7', '2007']
```

(e) The pagerank algorithm

In this final section we will implementing the google pagerank algorithm by computing principal eigenvector using power iteration method. (3 points)

To start with the power iteration method, we first need to make the matrix X obtained in (d) *column stochastic*. A column stochastic matrix is a matrix in which each element represents a probability and the sum each column adds up to 1. Recall that X is a matrix where the rows represent the destination and columns represents the source. The probability of visiting any destination from the source s is $1/k$, where k is the total number of outgoing links from s. Use this information to make the matrix column stochastic.

In [28]:

```
#######
# !!!! YOUR CODE HERE !!!!
# Make a copy of X
Y = X.copy()

# get 1D flattened array total outgoing links corresponding to each inde
x
outgoing_links = np.asarray(Y.sum(axis=1)).ravel()

print("Making the matrix column stochastic")

# Looping over nonzero indices
for i in outgoing_links.nonzero()[0]:
    # update value of probability by dividing it by corresponding total
 outgoing links
    Y.data[Y.indptr[i] : Y.indptr[i + 1]] *= 1.0 / outgoing_links[i]

#######
```

```
Making the matrix column stochastic
```

***Dangling Nodes*: There may exisit some pages which have no outgoing links. These are called as dangling nodes. If a random surfer just follows outgoing page links, then such a person can never leave a dangling node. We cannot just skip such a node, as there may be many pages pointing to this page, and could therefore be important.**

To solve this problem, we introduce teleportation which says that a random surfer will visit an outgoing link with β probability and can randomly jump to some other page with a $(1-\beta)/n$ probability (like through bookmarks, directly going through URL, etc.). Here n is the total number of pages under consideration, and β is called the damping factor. So now, the modified transition matrix is:

$$R = \beta X + \frac{(1-\beta)}{n} I_{n \times n}$$

where X is the column stochastic matrix from previous step, and $I_{n \times n}$ is a $n \times n$ identity matrix.

Using the transition matrix R, use the power iteration method to solve for the principal eigenvector $\mathbf{p}_{n \times 1}$. Start with an initial guess of $\mathbf{p}_{n \times 1} = [\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}]$, which intuitively represents that a random surfer can start at any page with a $\frac{1}{n}$ probability. Use a damping factor of 0.85, and perform a maximum of 100 iterations.

Report the top 10 page names which correspond to the top 10 scores (magnitudes) in the principal eigenvector.

In [29]:

```
#######
# !!!! YOUR CODE HERE !!!!
def power_method(X, beta=0.85, max_iterations=100):

    n = X.shape[0]
    tolerance = 1e-10

    #initial guess, every value = 1/n
    scores = np.array([(1.0 / n) for i in range(n)], dtype=np.float32)

    oldEigenValue = 0

    for i in range(max_iterations):
        prev_scores = scores
        # modifying scores using new transition matrix
        scores = beta*prev_scores*X + (1.0 - beta) * prev_scores.sum() /
n

        # eigenvalue calculation
        newEigenValue = np.abs(scores).max()

        if newEigenValue == 0.0:
            newEigenValue = 1.0

        # normalizing scores
        scores = scores/newEigenValue

        # error calculation
        error = np.abs(newEigenValue - oldEigenValue)
        if error < tolerance:
            return scores

        oldEigenValue = newEigenValue

    return scores


# Calculating principal eigenvector scores using power method
scores = power_method(Y, max_iterations=100)

# Reporting the top 10 page names which correspond to the top 10 scores
 (magnitudes) in the principal eigenvector.
print("Top 10 page names")
pp.pprint([reverse_index_map[i] for i in np.abs(scores).argsort()[-10
:]])

#######
```

```
Top 10 page names
[   'Telecommunications_in_Brazil',
    'Politics_of_Romania',
    'List_of_Star_Trek:_The_Next_Generation_episodes',
    'Foreign_relations_of_Afghanistan',
    'Demographics_of_Poland',
    'Foreign_relations_of_Syria',
    'Foreign_relations_of_South_Africa',
    'List_of_fictional_robots_and_androids',
    'Foreign_relations_of_Uruguay',
    'Foreign_relations_of_Turkey']
```