Eigenvector of a matrix

Eigenvalues are a special set of scalars associated with a linear system of equations (i.e., a matrix equation) that are sometimes also known as characteristic roots or characteristic values. The determination of the eigenvalues and eigenvectors of a system is extremely important in physics and engineering, where it is equivalent to matrix diagonalization and arises in such common applications as stability analysis, the physics of rotating bodies, and small oscillations of vibrating systems. Each eigenvalue is paired with a corresponding so-called eigenvector. The decomposition of a square matrix **A** into eigenvalues and eigenvectors is known in this work as eigen decomposition. The power iteration or von-Mises iteration is a simple algorithm to compute the greatest absolute eigenvalue and its eigenvector of a matrix. It will find only one eigenvalue and it may converge only slowly. For further information see Wikipedia.

Class design

Design a new class `EigenvalueProblem`. The properties of an eigenvalue problem are the associated matrix, the calculated eigenvector and a tolerance level. The class `EigenvalueProblem` has a constructor with a matrix object as parameter. Specify a method to compute and store the eigenvector using the power iterator algorithm. Furthermore define another method to retrieve a pre-calculated eigenvector. Create the UML diagram of the `EigenvalueProblem` class. In the next step develop the Nassi-Schneiderman diagram for the power iteration algorithm. Use an appropriate loop statement with a termination condition using the defined tolerance level.

Implementation

Implement the class `EigenvalueProblem` in Java. Use the `multiply` method of the class `Matrix` to compute the next vector $\mathbf{b}_{k+1} = \mathbf{A}\mathbf{b}_k / \|\mathbf{A}\mathbf{b}_k\|$ during every iteration (see algorithm at Wikipedia). Develop some test cases (test programs, test units) to verify the power iteration method. Think about problems which might arise when using special matrices. As an example, note that an eigenvector of the matrix $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ is $[0.4574, 1.0]^T$.

Power iteration (from Wikipedia):

In mathematics, **power iteration** (also known as the **power method**) is an eigenvalue algorithm: given a diagonalizable matrix A, the algorithm will produce a number λ, which is the greatest (in absolute value) eigenvalue of A, and a nonzero vector v, which is a corresponding eigenvector of λ, that is, $Av = \lambda v$. The algorithm is also known as the **Von Mises iteration**.[1]

The power iteration algorithm starts with a vector b_0, which may be an approximation to the dominant eigenvector or a random vector. The method is described by the recurrence relation

$$b_{k+1} = \frac{Ab_k}{\|Ab_k\|}$$

So, at every iteration, the vector b_k is multiplied by the matrix A and normalized.

If we assume A has an eigenvalue that is strictly greater in magnitude than its other eigenvalues and the starting vector b_0 has a nonzero component in the direction of an eigenvector associated with the dominant eigenvalue, then a subsequence (b_k) converges to an eigenvector associated with the dominant eigenvalue.

Without the two assumptions above, the sequence (b_k) does not necessarily converge. In this sequence,

$$b_k = e^{i\phi_k} v_1 + r_k,$$

where v_1 is an eigenvector associated with the dominant eigenvalue, and $\|r_k\| \to 0$. The presence of the term $e^{i\phi_k}$ implies that (b_k) does not converge unless $e^{i\phi_k} = 1$. Under the two assumptions listed above, the sequence (μ_k) defined by

$$\mu_k = \frac{b_k^* A b_k}{b_k^* b_k}$$

converges to the dominant eigenvalue (with Rayleigh quotient).[clarification needed]

```python
#!/usr/bin/env python3

import numpy as np

def power_iteration(A, num_iterations: int):
    # Ideally choose a random vector
    # To decrease the chance that our vector
    # Is orthogonal to the eigenvector
    b_k = np.random.rand(A.shape[1])

    for _ in range(num_iterations):
        # calculate the matrix-by-vector product Ab
        b_k1 = np.dot(A, b_k)

        # calculate the norm
        b_k1_norm = np.linalg.norm(b_k1)

        # re normalize the vector
        b_k = b_k1 / b_k1_norm

    return b_k

power_iteration(np.array([[0.5, 0.5], [0.2, 0.8]]), 10)
```

The vector b_k to an associated eigenvector. Ideally, one should use the Rayleigh quotient in order to get the associated eigenvalue.

This algorithm is used to calculate the *Google PageRank*.

The method can also be used to calculate the spectral radius (the eigenvalue with the largest magnitude, for a square matrix) by computing the Rayleigh quotient

$$\rho(A) = \max\{|\lambda_1|, \ldots, |\lambda_n|\} = \frac{b_k^\top A b_k}{b_k^\top b_k} = \frac{b_{k+1}^\top b_k}{b_k^\top b_k}.$$

```
1. Start

2. Read Order of Matrix (n) and Tolerable Error (e)

3. Read Matrix A of Size n x n

4. Read Initial Guess Vector X of Size n x 1

5. Initialize: Lambda_Old = 1

6. Multiply: X_NEW = A * X

7. Replace X by X_NEW

8. Find Largest Element (Lamda_New) by Magnitude from X_NEW

9. Normalize or Divide X by Lamda_New

10. Display Lamda_New and X

11. If |Lambda_Old - Lamda_New| > e then
      set Lambda_Old = Lamda_New and goto
      step (6) otherwise goto step (12)

12. Stop
```

My code

```
package chapter1;

public class EigenvalueProblem {

    private Matrix M;
    private static final double EPS=1e-10;
    private Vector V;
    private int MAX_ITER = 40;

    public EigenvalueProblem (Matrix M) { //We define the matrix we are dealing with
        this.M=M;
    }
```

```
    public Vector getEigenvector () {
        if (this.V!=null) {
            return this.V;}
        int i= M.getM();
        double [] x = new double [i];
        Vector v= new Vector (x);
        v.fill(1);
        Vector vnew = new Vector (i);
        double lambdaOld = 1;
        int steps=0;
        double lambdaNew;

            do {

                vnew = M.multiply(v);
                v=vnew;
                lambdaNew=vnew.maxNorm();
                double alpha = 1/lambdaNew;
                v = v.multiply(alpha);
                steps++;
    } while (Math.abs(lambdaOld-lambdaNew) > EPS && steps < MAX_ITER);
            this.V=v;
            return v;}
```

```
        public double getEigenvalue () {
                int i= M.getM();
                double [] x = new double [i];
                Vector v= new Vector (x);
                v.fill(1);
                Vector vnew = new Vector (i);
                double lambdaOld = 1;
                int steps=0;
                double lambdaNew;
            do {
                        vnew = M.multiply(v);
                        v=vnew;
                        lambdaNew=vnew.maxNorm();
                        double alpha = 1/v.maxNorm();
                        v = v.multiply(alpha);
                        steps++;
            } while (Math.abs(lambdaOld-lambdaNew) > EPS && steps < MAX_ITER);
        this.V=v;
        return (v.scalar(M.multiply(v)))/v.scalar(v);
}
```

```java
    public Vector preEigenvector () {
        if (this.V!=null) {
            System.out.print("Eigenvector has been retrieved and his value is:  " );
            this.V.print("Eigenvector");
            return this.V;
        } else {
            System.out.println("Eigen vector has not been calculated yet");
            return null;
        }
    }

    public static void main (String [] args) {

        double [] [] x = {{1,2,7},{4,12,5}, {67,5,9}};
        Matrix M = new Matrix (x);
        EigenvalueProblem M1 = new EigenvalueProblem (M);
        Vector v = M1.getEigenvector();
        v.print("Eigenvector is");
        double lambda=M1.getEigenvalue();
        System.out.println("\n"+"Eigen value is:"+lambda);
        Vector v1 = M1.preEigenvector();
    }
}
```

Results via calculator:

Matrix A:

$$\begin{pmatrix} 1 & 2 & 7 \\ 4 & 12 & 5 \\ 67 & 5 & 9 \end{pmatrix}$$

Cells | + | −

Find

1. $\lambda_1 \approx -16.900$
2. $\lambda_2 \approx 9.761$
3. $\lambda_3 \approx 29.139$

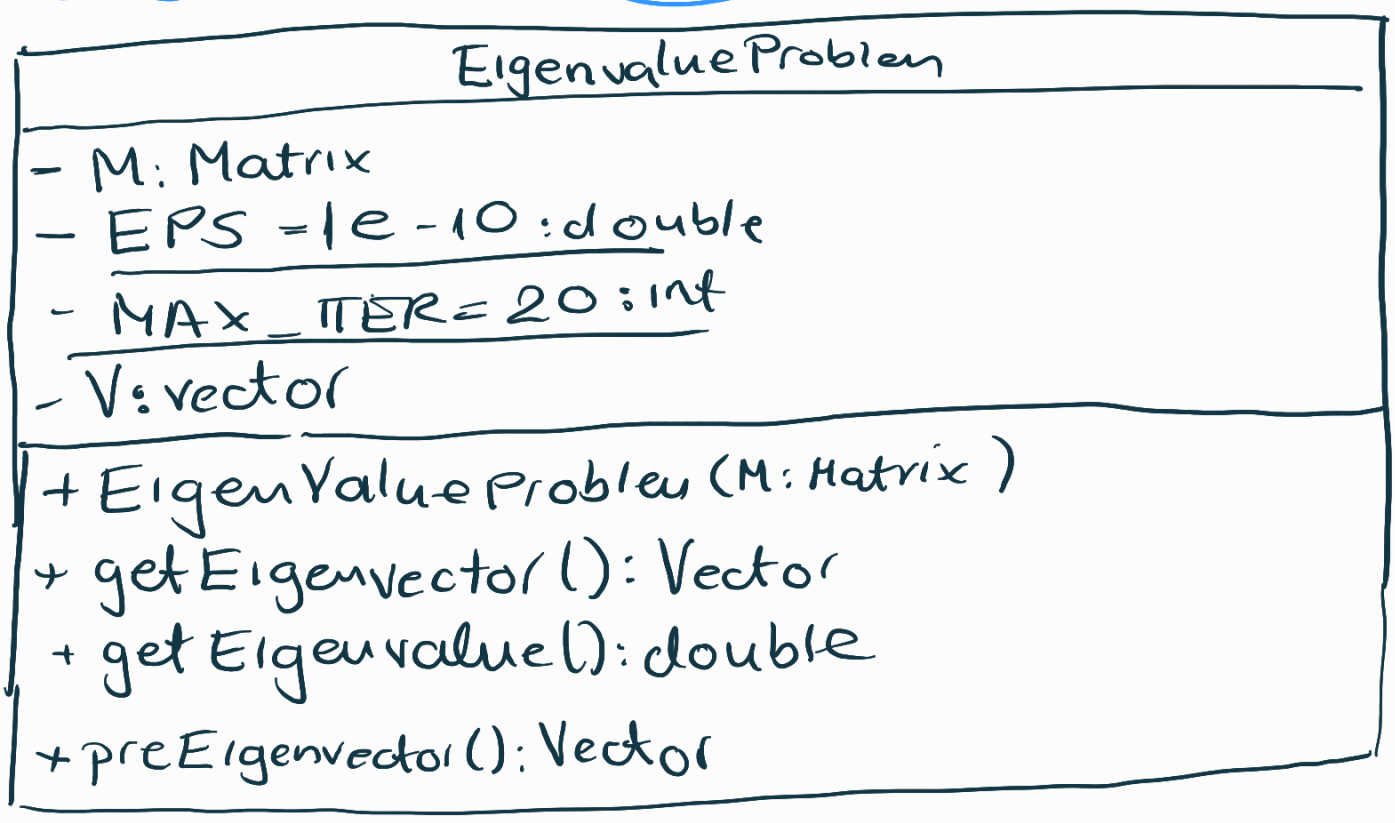
Let $x_3 = 1$, $v_3 \approx \begin{pmatrix} 0.274 \\ 0.356 \\ 1 \end{pmatrix}$

My solution:

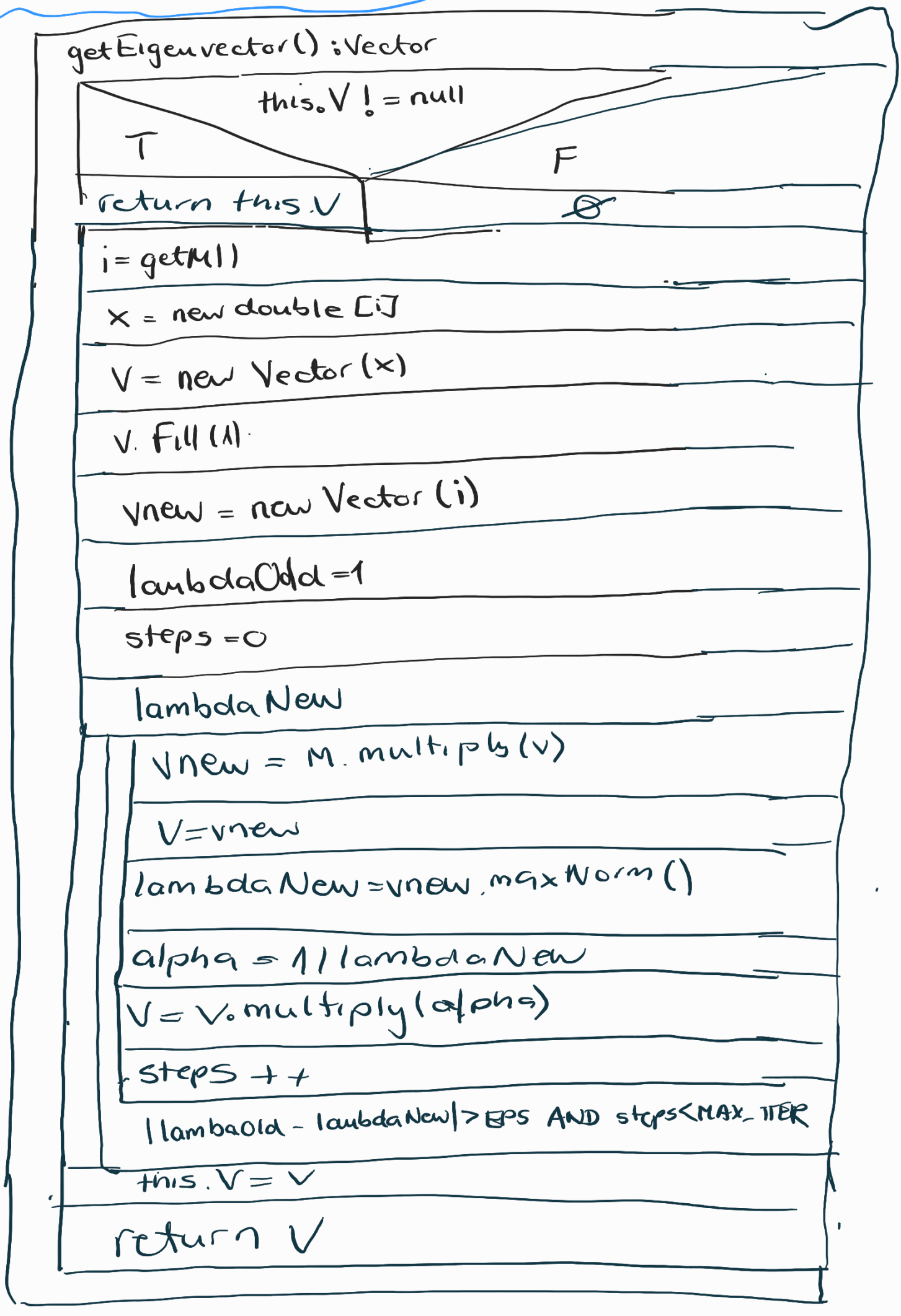
Problems | Javadoc | Declaration | Console | Call Hierarchy

```
<terminated> EigenvalueProblem [Java Application] C:\Users\User\Desktop\Programming\Eclipse\jre1.8.0_201\jre1.8.0_201\jre1.8.0_201\bin\javaw.exe (7.
Eigenvector is= (0.27404277602137705,0.3556840691783756,1.0)^T
Eigen value is:29.139286336954072
Eigenvector has been retrieved and his value is:        Eigenvector= (0.27404277602137705,0.3556840691783756,1.0)^T
```

UML Diagram of the class:



Nassi - Shneiderman



Additions to the program, addressing potential problems and errors will be done in the future

- We should check if the Matrix is a $\underline{0}$ matrix
- we should check if the eigenvalue was found or did the iterations reach the max number