UNIVERSIDADE ESTADUAL DE CAMPINAS

Faculdade de Engenharia Civil, Arquitetura e Urbanismo

IC639: Métodos Numéricos para Engenharia Civil

List 5

Numerical Solutions of Nonlinear Equations

Student:

Carlos Henrique Chama Puga - 195416

Advisors:

Porf. Dr. Philippe Devloo

Dr. Giovane Avancini

Campinas

2024

Contents

1 Introduction

It is possible to think of a matrix as a linear transformation applied to a vector. When the matrix is applied to a vector and the resulting vector is parallel to the original vector, this vector is said to be an eigenvector of the matrix so that

$$A\mathbf{v} = \lambda\mathbf{v}, \tag{1.1}$$

in which A is the matrix, $\mathbf{v}$ is the eigenvector, and λ is the eigenvalue.

Equation (1.1) can be rewritten as

$$(A - \lambda I)\mathbf{v} = 0, \tag{1.2}$$

which means that, for any nonzero vector $\mathbf{v}$, Eq. (1.2) holds only if the determinant of the matrix $A - \lambda I$ is zero. In other words, matrix $A - \lambda I$ is singular. This is the characteristic equation of the matrix A and can be used to find the eigenvalues of the matrix.

Naturally, the eigenvalues of a matrix are the roots of the characteristic equation. However, a generalization of the characteristic equation is not always possible, and therefore, its analytical solution is not always feasible.

The Power Method is an iterative method that can be used to find the most prominent eigenvalue of a matrix. The method is simple to implement and can be used to find both the eigenvalue and the eigenvector of the matrix.

For a given vector $\mathbf{y}$ linearly independent of $\mathbf{v}$, it is possible to write

$$\mathbf{y} = \sum_{i=1}^{n} \beta_i \mathbf{v}_i, \tag{1.3}$$

in which $\mathbf{v}_i$ are the eigenvectors of the matrix A.

Multiplying Eq. (1.3) by A^k yields

$$A^k\mathbf{y} = \sum_{i=1}^{n} \beta_i \lambda_i^k \mathbf{v}_i, \tag{1.4}$$

and factoring Eq. (1.4) by λ_1^k gives

$$A^k\mathbf{y} = \lambda_1^k \sum_{i=1}^{n} \beta_i \left(\frac{\lambda_i}{\lambda_1}\right)^k \mathbf{v}_i. \tag{1.5}$$

Since λ_1 is the most prominent eigenvalue, the term $\left(\frac{\lambda_i}{\lambda_1}\right)^k$ tends to zero for $i \neq 1$. Therefore, the term $\sum_{i=1}^{n} \beta_i \left(\frac{\lambda_i}{\lambda_1}\right)^k \mathbf{v}_i$ tends to $\beta_1 \lambda_1^k \mathbf{v}_1$ as k tends to infinity.

This is the main idea behind the Power Method. The method starts with an initial guess $\mathbf{v}_0$. The matrix A is multiplied by $\mathbf{v}_0$ to find the next eigenvector

$$\mathbf{y} = A\mathbf{v}_{i-1}, \tag{1.6}$$

the norm of the result is calculated and considered the new eigenvalue

$$\lambda_i = \|\mathbf{y}\|, \tag{1.7}$$

the new eigenvector is the previous result divided by the new eigenvalue

$$\mathbf{v}_i = \frac{\mathbf{y}}{\lambda_i}, \tag{1.8}$$

This process is repeated until the error between the current and previous eigenvalues is smaller than a given precision. Since Eq. (1.6) is performed at each iteration, the limit when k tends to infinity is approached.

In the next sections, the power method is implemented and the results are discussed. The following bibliography is referred to in this work: (1, 2).

2 The PowerMethod Class

The power method for finding the most prominent eigenvalue of the matrix A is implemented in the PowerMethod class. The class initializer is shown in Code 1.

```
class PowerMethod:
    A: np.array
    fixed: bool

    number_of_iterations: int = 20
    precision: float = 1e-8

    size: int = field(init=False)
    B: np.array = field(init=False)

    eigenval: list[float] = field(init=False, default_factory=list)
    eigenvec: list[float] = field(init=False, default_factory=list)
    error: list[float] = field(init=False, default_factory=list)

    LAMBDA: np.array = field(init=False, default_factory=list)
    Q: np.array = field(init=False, default_factory=list)

    def __post_init__(self):
        self.A = ARRAY(self.A)
        self.B = self.A.copy()
        self.size = len(self.A)

        self.LAMBDA = np.zeros_like(self.A)
        self.Q = np.zeros_like(self.A)
```

Code Listing 1: PowerMethod class initializer

The attribute A is the matrix for which the eigenvalues and eigenvectors are to be found. The field fixed is a bool that determines whether the power method's initial guess is random or not (in the case of fixed=True, a vector with ones is created). Size is the matrix dimension, and B is a copy of A, as shown

in the post_init method.

The maximum number of iterations is set to 20, but a function to change this value is available. The same happens to the precision, which is set to 10^{-8}, but can be changed.

The eigenvalues and eigenvectors are stored in the lists eigenval and eigenvec, respectively, as well as the error in the list error. Last but not least, the decomposition matrices Λ and Q are stored in the attributes LAMBDA and Q, respectively.

Several methods to set and get the class attributes are defined. They are SetNumberOfIterations, SetPrecision, GetEigenvector, GetEigenvalue, GetEigensystem (to get both eigenvalues and eigenvectors) and GetError. Those methods are not shown here due to their simplicity.

The power method starts when the Run method is called. The method is shown in Code 2.

```
1  def Run(self)->None:
2      for _ in range(self.size):
3          lambda_, v, e = self.FindEigenSystem()
4
5          self.eigenval.append(lambda_)
6          self.eigenvec.append(v)
7          self.error.append(e)
8
9          self.UpdateMatrix(lambda_[-1], v[-1])
10
11     self.Decompostion()
```

Code Listing 2: PowerMethod Run method

The for-loop runs the FindEigenSystem method for each eigenvalue of the matrix (line 3). Here the matrix is assumed to have a full set of eigenvalues. The eigenvalues, eigenvectors, and errors of each iteration of the power method are stored in the lists eigenval, eigenvec, and error, respectively (lines 5 to 7).

Finally, the matrix B is updated to remove the found eigenvalue and respective eigenvector (line 9). The matrix A is then decomposed into the matrices Λ and Q (line 11). More details about these methods are given next.

The FindEigenSystem method is shown in Code 3.

```
1  def FindEigenSystem(self)->tuple:
2      v0 = np.ones(self.size) if self.fixed else np.random.uniform(-1, 1,
       self.size)
3
4      lambda_i = [NORM(v0)]
5      v_i = [v0/lambda_i[0]]
6
7      errors = []
8
9      for i in range(1, self.number_of_iterations):
10         previous = self.B @ v_i[i-1]
11
12         lambda_i.append(NORM(previous))
13         v_i.append(previous/lambda_i[i])
14
15         sign = previous @ v_i[i-1]
16         if sign < 0:
17             lambda_i[i] *= -1
18
19         error = abs(lambda_i[i] - lambda_i[i-1])/abs(lambda_i[i])
20         errors.append(error)
21
22         if error < self.precision:
23             break
24
25     return lambda_i, v_i, errors
```

Code Listing 3: PowerMethod FindEigenSystem method

The initial guess v_0 is set depending on the fixed attribute. The lists v_i and lambda_i store the eigenvectors and eigenvalues of each iteration of the method.

From line 9, the power method is properly implemented. The matrix B is multiplied by the eigenvector of the previous iteration. The norm of the result

is calculated and considered the new eigenvalue. The new eigenvector is the previous result divided by the new eigenvalue.

If the dot product between the new eigenvector and the previous one is negative, the eigenvalue is multiplied by -1. The error is calculated as the difference between the current and previous eigenvalues divided by the absolute value of the current eigenvalue. If the error is smaller than the precision, the loop is broken.

The UpdateMatrix method is shown in Code 4.

```
def UpdateMatrix(self, eigenval, eigenvec)->None:
    self.B -= eigenval * OUTER(eigenvec, eigenvec)
```

Code Listing 4: PowerMethod UpdateMatrix method

The method subtracts the outer product of the current eigenvector by itself multiplied by the current eigenvalue from the matrix B. This operation removes the just-found eigenvalue and eigenvector from the matrix. Code 5 shows the Decomposition method.

```
def Decomposition(self)->None:
    for i, eigenvalues in enumerate(self.eigenval):
        self.LAMBDA[i, i] = eigenvalues[-1]

    for i, eigenvectors in enumerate(self.eigenvec):
            self.Q[:, i] = eigenvectors[-1]
```

Code Listing 5: PowerMethod Decomposition method

The final method called by the Run function is the Decomposition method. It fills the matrices Λ and Q with the found eigenvalues and eigenvectors, respectively. There is yet another method in the PowerMethod class, the WriteResults function, which writes the results to a file.

An example of the main function using the PowerMethod class is shown in Code 6.

```
1  def main()->None:
2      fixed = True
3      file_name = "List6/results_v0Fixed" if fixed else "List6/
       results_v0Variable"
4
5      A = [ -- MATRIX --]
6
7      pm = PowerMethod(A, fixed)
8
9      pm.SetPrecision(1e-13)
10     pm.SetNumberOfIterations(50)
11
12     pm.Run()
13
14     error = pm.GetError()
15     PlotConvergence(error, f"{file_name}.pdf")
16
17     pm.WriteResults(f"{file_name}.txt")
18
19     return
```

Code Listing 6: PowerMethod main function

On line 5 the matrix A is set. Due to its size, it is not shown here. On line 7 the object pm of the PowerMethod class is created and its precision is set to 10^{-13} and the number of iterations to 50. The Run method is called on line 12, and the error is plotted on line 15 (plot function defined separately). The results are written in a file on line 15.

3 Results

Although a fixed initial guess can be set, the results presented here are for a random initial guess. The max number of iterations is set to 50 and the precision to 10^{-14}. Results are shown in Table 3.1.

Table 3.1: Eigenvalues results for the power method.

Index	Eigenvalue	Error	Iterations
λ_1	49999999999.9982	3.280e-14	15
λ_2	16666600000.00010	0.0	4
λ_3	-244480.84437	5.952e-16	7
λ_4	10136.22809	2.332e-14	17
λ_5	4201.35058	2.359-14	20
λ_6	1882.41227	2.427e-14	16
λ_7	593.93884	2.067e-14	16
λ_8	193.27852	5.543e-14	5
λ_9	-1.80918	6.382e-14	37
λ_{10}	1.20523	3.505e-9	49
λ_{11}	-1.00002	1.036e-11	49
λ_{12}	1.00556	9.958e-14	14
λ_{13}	-0.35385	4.156e-11	49
λ_{14}	-0.27373	4.015e-14	6
λ_{15}	-0.00558	4.470e-14	16

The convergence of the power method is shown in Figure 3.1.

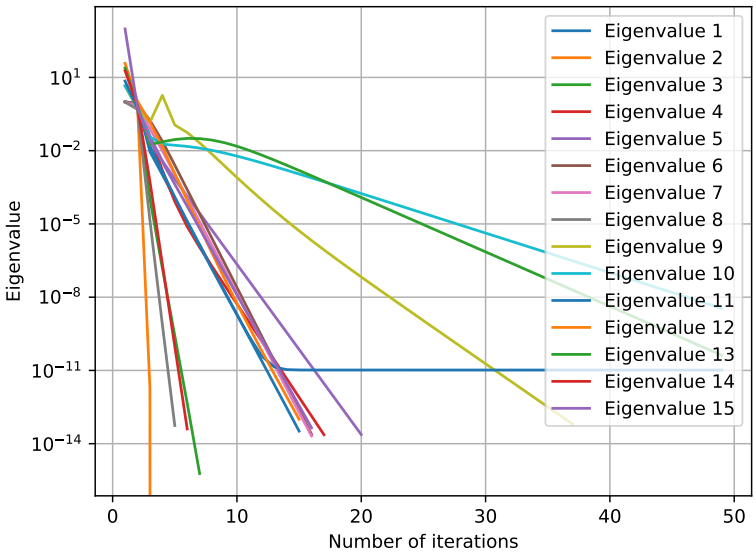

Figure 3.1: Convergence of the power method.

It is observed that almost every eigenvalue converges in less than 50 itera-

tions. Only the 10th, 11th, and 13th do not converge. Figure 3.1 shows that for the 11th eigenvalue, the error finds itself in a plateau, with no significant decrease between iterations.

Results for eigenvectors as well as the decomposition of the matrix can be found in the attached files *results_v0Fixed.txt* and *results_v0Variable.txt*. Both files are in the git repository mentioned in Section A. The eigenvectors are not shown here due to their size.

4 Conclusions

In this list, the power method for finding eigenvalues and eigenvectors was implemented. The method was tested with a random initial guess and a fixed initial guess. The results showed that the method converges, for this specific matrix, in less than 50 iterations for almost every eigenvalue. The code implementation could recover the eigenvalues with a precision of 10^{-14} for the majority of the eigenvalues.

One advantage of the power method is that it is simple to implement and computes both eigenvalue and eigenvector at the same time. However, it is necessary to have a full set of eigenvalues, which is not always the case.

The power method is a good starting point for finding the most prominent eigenvalue of a matrix. A generalization can be made so that the method is extended to find any eigenvalue of the matrix, not only the most prominent one.

Finally, the matrix decomposition was implemented, recovering the matrix A from the matrices Λ and Q.

References

1 CUNHA, M. C. de C. *Métodos numéricos*. [S.l.]: Editora da UNICAMP, 2000.

2 BURDEN, R. L.; FAIRES, J. D. *Numerical analysis, brooks*. [S.l.]: Cole publishing company Pacific Grove, CA:, 1997.

A GitHub Repository

The source code for this report and every code mentioned can be found in the following GitHub repository: CarlosPuga14/MetodosNumericos_2024S1.