Solving the eigenvalue problem over the Levi-Civita field

Ragon Ebker

Fernuninversität Hagen

26th ISEM, 18.07.2023

Overview

Requirements

- The **von Mises iteration** or **power iteration** presents an algorithm to approximate the eigenvector corresponding to the largest eigenvalue of a Matrix $A \in \mathbb{K}^{n \times n}$. $\mathbb{K} = \mathbb{R}, \mathbb{C}, \mathcal{R}$ or $\mathcal{C}$.

Requirements

Requirements

Definition

Let $A \in \mathbb{K}^{n \times n}$. May the eigenvalues of A be ordered such that $|\lambda_i| \geq |\lambda_j|, i, j \in \{1, ..., n\}, i \geq j$.

We define λ_1 to be the **dominant eigenvalue** of A, if $|\lambda_1| > |\lambda_j|, j \in \{r+1, ..., n\}$. $r \in \mathbb{N}$ being the multiplicity of λ_1. The corresponding eigenspace $\mathcal{N}(\lambda_1 I_m - A)$ is called the **dominant eigenspace**.

Algorithm

Algorithm

Now in every step of our algorithm, we multiply our matrix A with our current vector $b_k, k \in \mathbb{N}$. As an initial vector $b_0 \in \mathbb{K}^n$ we choose a random vector. We have the following recurrent definition:

$$c_k = A \cdot b_k$$

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

$|| \cdot ||$ being a vector norm on $\mathbb{K}^n$

Non-vanishing component

Non-vanishing component

- We require our initial vector b_0 to have a non-vanishing component in the dominant eigenspace.
- Numerically we can neglect this, because even if this is not the case for our initial vector, it will start happening in the course of iteration, through rounding errors.

The Reason why the von Mises algorithm is important

The Reason why the von Mises algorithm is important

- In the numerical investigation of (Complex and Real valued)-Matrices there are three methods to solve the eigenvalue problem

The Reason why the von Mises algorithm is important

- In the numerical investigation of (Complex and Real valued)-Matrices there are three methods to solve the eigenvalue problem
  - Characteristic Polynomial
  - Reduction to Hessenberg or Tridiagonal form
  - Iterative methods

Three different methods

- It is possible to calculate the characteristic polynomial and its roots for a Levi-Civita matrix, but its a topic for another day
- Since we are not only looking at symmetric or hermitian matrices, the methods of reducing our matrix to a convenient form are not useful to calculate its eigenvalues.
- What's left are iterative methods.

Iterative Methods

Iterative methods are all based on the von Mises algorithm

Representation of Levi-Civita numbers in Python

- We can represent Levi-Civita numbers symbolically in Sympy, as a rational function:
  `eps = symbols('eps')`
  `x = 1 + 2*eps**1 + 5*eps**2`
- We can also represent them with the help of the Levi-Civita library [Andrew Barnertt] [1]
- We choose the Sympy library because it has more numerical stability while calculating the expansion of a Levi-Civita number
- **Disclaimer:** For testing the algorithm we only used integers ≥ 0 as exponents. Theoretically, we work with rational exponents ≥ 0.

[1] Andrew Barnert, https://github.com/abarnert/levicivita

An example for numerical instability

```
from levicivita import *
from levicivita import lcmath
a = (1+d)/(2+2*d + d**2)

print(a)
print((a**(1/2))*(a**(1/2)))
```

This outputs:

```
0.5-0.25*d**2+0.25*d**3-0.125*d**4+0.0625*d**6
0.5-0.25*d**2+0.25*d**3+0.09375*d**4-0.4375*d**5
```

Expansion of an LC number

What does that even mean expanding an LC number?

We need to rewrite every LC term after every calculation, otherwise, we accumulate huge algebraic expressions:

$$\begin{bmatrix} -\frac{1.0\epsilon\left(\frac{1.0\epsilon}{2.0\epsilon+1.0}+1\right)}{1.0\epsilon+2.0} - \frac{1.0\epsilon}{1.0\epsilon+2.0} - \frac{1.0\left(-1-\frac{1.0}{1.0\epsilon+2.0}\right)}{1.0\epsilon+2.0} + \frac{1.0}{1.0\epsilon+2.0} \\ -\frac{1.0\epsilon\left(-1-\frac{1.0}{1.0\epsilon+2.0}\right)}{2.0\epsilon+1.0} - \frac{1.0\epsilon\left(-\frac{1.0\epsilon}{1.0\epsilon+2.0}+\frac{1.0}{1.0\epsilon+2.0}\right)}{2.0\epsilon+1.0} + \frac{1.0\epsilon}{2.0\epsilon+1.0} + 1 \\ -1 - \frac{1.0\left(-\frac{1.0\epsilon}{1.0\epsilon+2.0}+\frac{1.0}{1.0\epsilon+2.0}\right)}{1.0\epsilon+2.0} - \frac{1.0\cdot\left(\frac{1.0}{2.0\epsilon+1.0}+1\right)}{1.0\epsilon+2.0} - \frac{1.0}{1.0\epsilon+2.0} \end{bmatrix}$$

Expansion of an LC number

- How do we create this expansion?
  - A simple way to get an expansion is to write its Taylor expansion
  - The Levi-Civita library creates Taylor expansions recursively, which is very fast, but also numerically unstable
  - The Sympy library creates Taylor expansions in a very slow and unreliable way

Expansion of an LC number

- How do we create this expansion?
  - For this reason, we wrote a custom Taylor expansion
  - After researching various implementations we encountered an experimental library from sympy that executes these operations efficiently and exactly.

Expansion of an LC number

- How can we create this expansion efficiently?
- We basically only have two Operations which brings us headaches
  - Inverses $\rightarrow$ Long Division
  - Square roots $\rightarrow$ Hard coded formula
  - We can also use the Newton formula for both, more info in [haoze]

[1] Hao Ze, Fast Power Series Inversion: Newton's Iteration and the Middle Product Optimization

von Mises algorithm

The whole code can be found on GitHub [Ragon, 2023] [2]

```
def power_iteration(A: np.array, num_iterations: int, start_vec : np.array, n_s : int):
  b_k = start_vec
  for _ in range(num_iterations):
      b_k = A @ b_k
      b_k = normalize_lc_2(b_k,n_s)
  return b_k
```

[2] Ragon Ebker, https://github.com/RagonEbker/Levi-Civita-Eigenvalue

The only method we have to change is the normalization:

```
def normalize_lc_2(v : np.array, n_s : int):
  dim = v.shape[0]
  norm_b_k = norm_lc_2(v,dim,n_s)
  for i in range(dim):
     v[i] = seriesExpansionEps(v[i]/norm_b_k,n_s)
  return v

def norm_lc_2(v : np.array, dim : int, n_s : int):
    norm = 0
    for i in range(0,dim):
      norm = norm + (v[i]**2)
    return seriesExpansionEps(norm**(1/2),n_s)
```

We also have a custom function to calculate the Rayleigh coefficient:

```
def rayleigh_coeff_lc_w_norm(A : np.array,v : np.array):
  return (v.T @ (A @ v)) / (v.T @ v)
```

Note that we can save ourselves the `v.T @ v` if our vector v is already normalized.
We also implemented the algorithm with the maximum norm, which saves us a Taylor expansion step.

A whole example

```
A = np.array([[1,1+eps],[1-eps,1]])
n_iterations = 10
dim = A.shape[0]
start_vec = np.random.rand(dim)
n_terms = 3
result = power_iteration(A,n_iterations,start_vec,n_terms)
ew = seriesExpansionEps(rayleigh_coeff_lc(A,result),n_terms)
print(ew)
```

The output of this code is:

```
-0.5*eps**2 + 0.e-254*eps + 2.0
```

This coincides with our hand-calculated result $2 - 0.5\epsilon^2$

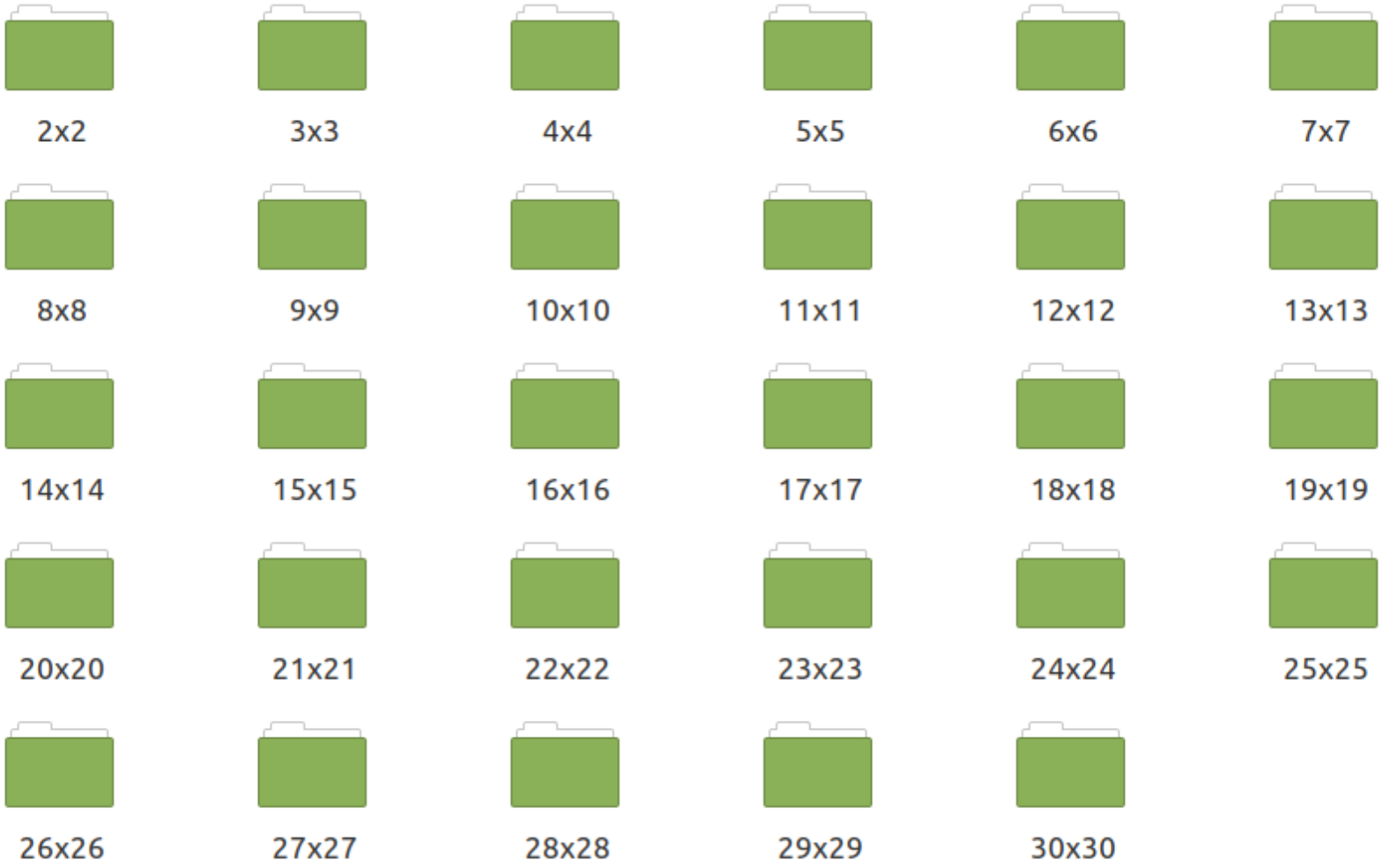
Test Environment

How can we test our algorithm?
$\rightarrow$ We build a test environment for Levi-Civita graphs
$\rightarrow$ We can use companion matrices of polynomials that have distinct roots

Test Environment

```
#dimension of the matrix
dim = 8

#Max Power of epsilon
n_eps = 1

#max number for the random coefficients
max_n = 10

#number of epsilon terms for representation of lc number
n_s = 10

create_graph_example(dim,n_eps,max_n,n_s)
```

Test Environment



Test Environment

Test Environment


3x3_2.dot


3x3_2.pdf


3x3_2_ adjacency.csv


3x3_2_ laplacian.csv

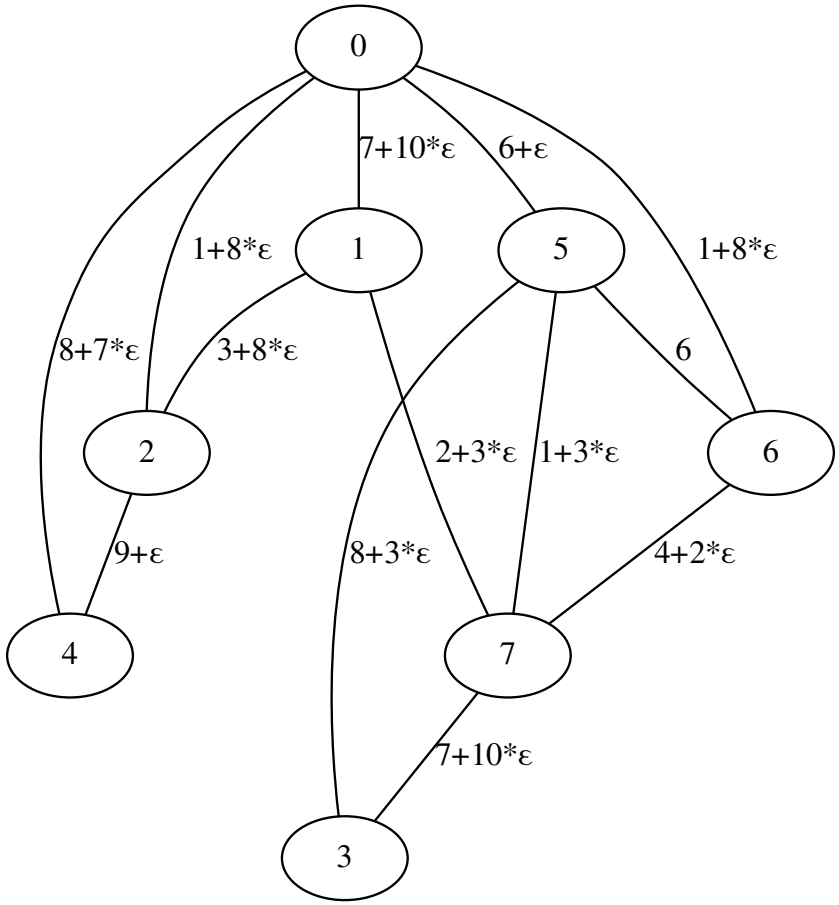

3x3_2_real_ adj.csv


3x3_2_real_ eig_vals.csv


3x3_2_real_ eig_vecs.csv


3x3_2_real_ lap.csv

Picture of the graph



Test Environment for Graphs

- So far we can only calculate eigenvalues if every eigenvalue has a distinct ϵ^0-part
- Thus we need a way to check this condition before we can try calculating the correct eigenvalues

Defintions

Let $\Pi : \mathcal{C} \mapsto \mathbb{C}$ be the projection of a Levi-Civita number into the complex numbers, $\Pi(x) = x[0]$ and
$\lambda(x) = min(supp(x)), x \in \mathcal{C}, \lambda(0) = 0$ describing the orders of magnitude of the number. Both are defined as in [ShaBe210] [3]

[3] Khodr Shamseddine and Martin Berz. Analysis on the Levi-Civita field, a brief overview, 2010

Definitions

Definition

Let $\Pi_A : \mathcal{C}^{n\times n} \to \mathbb{C}^{n\times n}$ be the corresponding function that projects each element of a matrix A into the complex numbers and $\Pi_M : M \to N$ be a function that projects every element of $M \subset \mathcal{C}$ into $\mathbb{C}$. Additionally, we have $\Pi_f(f)$ being a function defined by $\Pi_f : \mathcal{C} \mapsto \mathbb{C}, x \mapsto \Pi(f(x))$

Example

$$\Pi \begin{pmatrix} 1+\epsilon & 2+\epsilon^3 \\ 3-\epsilon^4 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

Lemma (Linearity of Π)

For $a, b \in \mathcal{C}$ with $\lambda(a) \geq 0, \lambda(b) \geq 0$ we have

$$\Pi(ab) = \Pi(a)\Pi(b)$$

and

$$\Pi(a + b) = \Pi(a) + \Pi(b)$$

Theorem

For the spectrum of every square matrix $A \in \mathcal{C}^{n \times n}$, with $\lambda(a_{ij}) \geq 0, \forall i, j \in \{1, ..., n\}$ and $\lambda(\mu) \geq 0$ for all its eigenvalues holds:

$$\Pi(\sigma(A)) = \sigma(\Pi(A))$$

Proof.

By the linearity of Π, we can see that A and $\Pi(A)$ have the characteristic polynomials χ and $\Pi\chi$ respectively.
"$\Pi(\sigma(A)) \subseteq \sigma(\Pi(A))$"
Let

$$\chi(x) = \sum_{i=0}^{n} b_i x^i, x \in \mathcal{C}.$$

Let $\mu \in \sigma(A)$. Then $\chi(\mu) = 0$ and by linearity of Π we get

$$(\Pi\chi)(\Pi\mu) = \sum_{k=0}^{n} (\Pi b_k)(\Pi\mu)^k = \Pi\left(\sum_{k=0}^{n} b_k \mu^k\right) = \Pi(\chi(\mu)) = 0.$$

Therefore, $\Pi\mu \in \sigma(\Pi(A))$ □

Proof.

"$\Pi(\sigma(A)) \supseteq \sigma(\Pi(A))$". Let $\nu \in \sigma(\Pi(A))$, i.e. $\Pi\chi(\nu) = 0$. Since $\mathcal{C}$ is algebraically closed, we can write

$$\chi = \prod_{i=1}^{n}(\mu_i - x),$$

where μ_i are (possibly duplicate) eigenvalues of A. Then again due to the linearity

$$\Pi\chi = \prod_{i=1}^{n}(\Pi(\mu_i - x)),$$

is a linear factor representation of $\Pi\chi$ which is known to be unique. Therefore for from $\Pi(\chi))(\nu) = 0$, we conclude that there exists an $i \in \{1, ..., n\}$ with $\Pi(\mu_i) = \nu$, i.e. $\nu \in \Pi(\sigma(A))$. □

Remark

We can generalize a modified version of this theorem for

$$\Pi_r : \mathcal{C} \to \mathbb{C},$$

$$\Pi_r(x) = x[r], r \in \mathbb{Q}.$$

This results in another algorithm that solves Levi-Civita Polynomials.

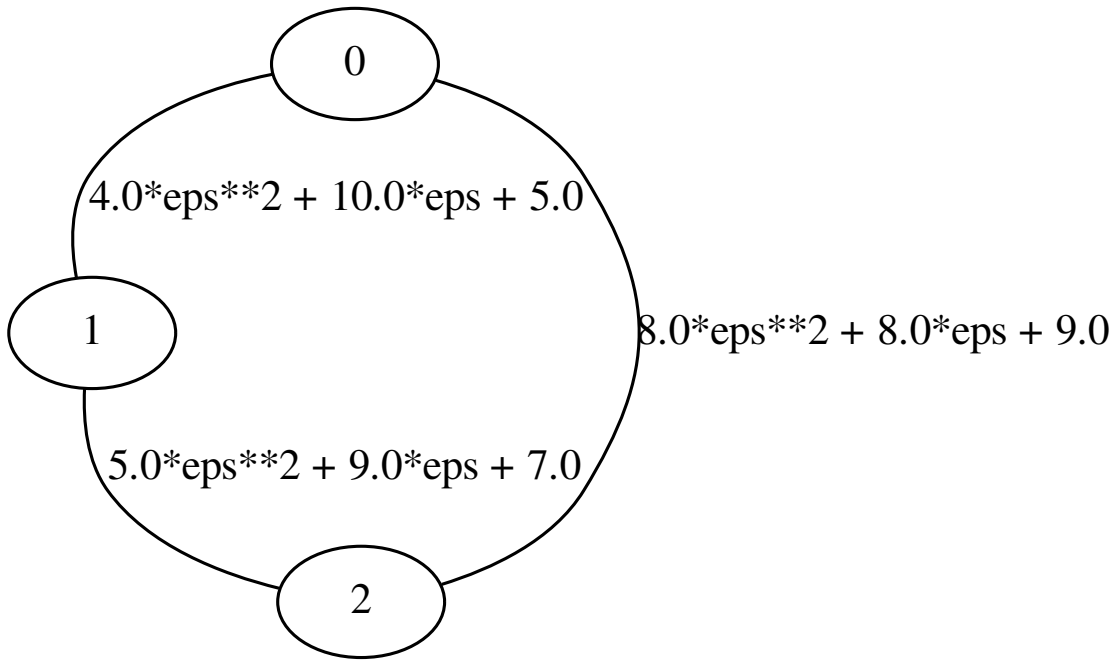
Test for distinct $\Pi(A)$ Eigenvalues

Now we can test that our Matrix really has eigenvalues with $\Pi(\lambda_i) \neq \Pi(\lambda_j), i, j \in \{1, ..., n\}$

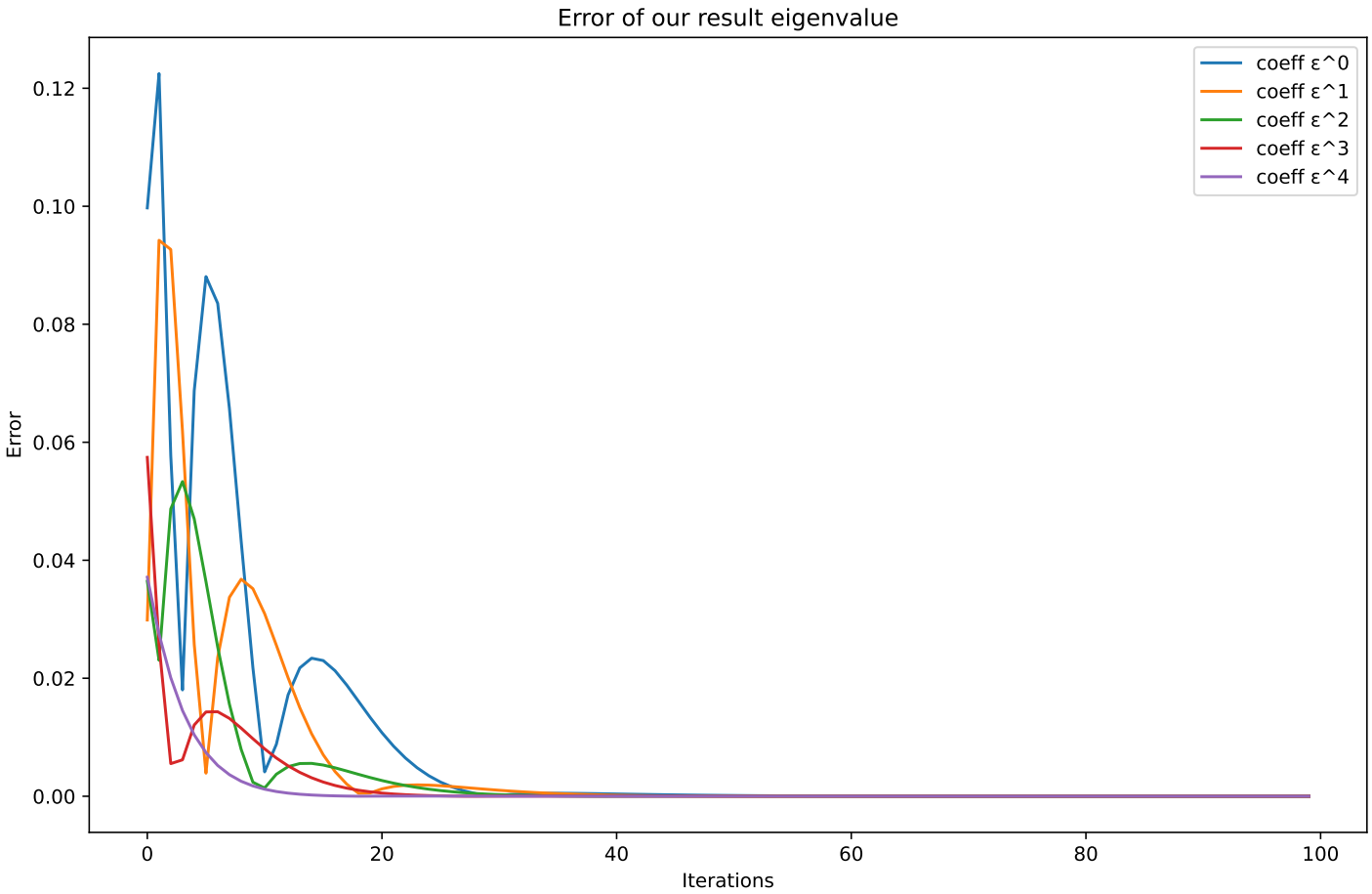
We now have a look at a matrix with our desired conditions

Our first example

The first example is a complete 3×3 graph. Its small size allows us to calculate the eigenvalues and eigenvectors by hand.



Graphing the results



Error Table

Error of our result eigenvalue in the respective components ϵ^0 to ϵ^4

Step	ϵ^0	ϵ^1	ϵ^2	ϵ^3	ϵ^4
0	4.92614e-02	7.62700e-02	5.39487e-02	3.07820e-02	1.21542e-01
10	1.74701e-03	1.10467e-02	2.25737e-03	4.24706e-02	4.59344e-03
20	1.18723e-05	7.68831e-04	3.65521e-03	1.96125e-03	1.45480e-02
30	1.43491e-05	1.95823e-05	3.75454e-04	1.36965e-03	3.53259e-04
40	3.12830e-06	2.01353e-05	6.59142e-06	2.29621e-04	5.17033e-04
50	6.03743e-07	5.97178e-06	1.72529e-05	1.52486e-05	1.58461e-04
60	1.14128e-07	1.48792e-06	6.95890e-06	8.80970e-06	3.29895e-05
70	2.14917e-08	3.46352e-07	2.18509e-06	5.77108e-06	1.25237e-06
80	4.04423e-09	7.75648e-08	6.12409e-07	2.35150e-06	2.95358e-06
90	7.60928e-10	1.69226e-08	1.60148e-07	7.96204e-07	1.87699e-06
100	1.69196e-10	4.22873e-09	4.59506e-08	2.73918e-07	8.84952e-07

Execution

The code execution took roughly 0.55 Seconds for 100 iterations on a Lenovo p50.

Companion Matrices

- Another way to test our algorithm is through companion matrices
- We know that our companion matrix is diagonalizable if we have distinct eigenvalues

A Larger Example

We generate a Polynomial of degree 21 from linear factor representation. It's largest root is:

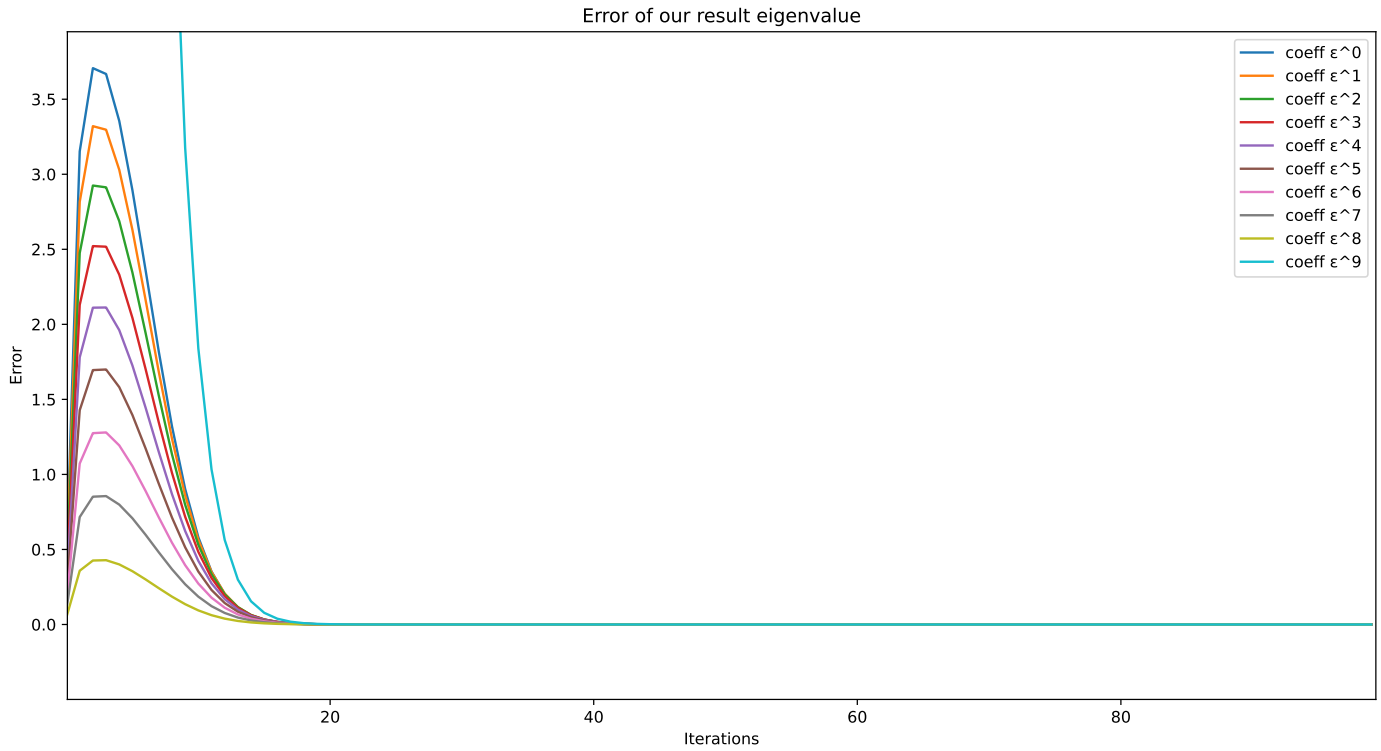
$$100 + \epsilon + 2\epsilon^2 + 3\epsilon^3 + 4\epsilon^4 + 5\epsilon^5 + 6\epsilon^6 + 7\epsilon^7 + 8\epsilon^8 + 9\epsilon^9.$$

We use its companion matrix to test our algorithm.

Graphing the results

The result our algorithm provides is:

```
8.99999999999543*eps**9 + 7.99999999999838*eps**8 + 6.99999999999915*eps**7 +
5.99999999999785*eps**6 + 4.99999999999836*eps**5 + 3.99999999999854*eps**4 +
2.99999999999936*eps**3 + 1.99999999999932*eps**2 + 0.999999999999311*eps + 100.000000000004
```



Error Table

Step	ϵ^0	ϵ^1	ϵ^2	ϵ^3	ϵ^4
0	3.79432e+02	5.05098e-02	1.00464e-01	1.50350e-01	2.00276e-01
10	1.82443e+00	9.32998e-02	1.84316e-01	2.70750e-01	3.50363e-01
20	1.85048e-03	2.74954e-04	5.28935e-04	7.41959e-04	8.96181e-04
30	4.21493e-07	1.04826e-07	1.96292e-07	2.62170e-07	2.92486e-07
40	5.26370e-11	1.81429e-11	3.27154e-11	4.09770e-11	4.21161e-11
50	1.08002e-12	9.39249e-14	1.73417e-13	7.90479e-14	4.52971e-14
60	7.13385e-12	6.54699e-13	8.59979e-13	1.40510e-12	2.35723e-12
70	5.40012e-13	9.41469e-14	1.34115e-13	1.96732e-13	4.40981e-13
80	1.47793e-12	1.30340e-13	3.46390e-14	2.01172e-13	1.19016e-13
90	1.30740e-12	3.49720e-14	4.70735e-13	7.91811e-13	3.06422e-13
100	3.65219e-12	6.88560e-13	6.76792e-13	6.43041e-13	1.46061e-12

Execution

The code execution took roughly 135.06 Seconds for 100 iterations on a Lenovo p50.

Outlook

How can we continue researching this topic?

- Calculate more eigenvalues and eigenvectors
  - Subspace iteration that uses Gram–Schmidt orthonormalization
  - QR algorithm that uses Householder reflections or Gram–Schmidt orthormalization
  - Inverse vector iteration
- Reduction of our matrix into Hessenberg form, to reduce iteration cost
- Comparison of computation cost to other algorithms that achieve similar tasks

Outlook

Since we have proven and implemented the central method of iterative eigenvalue calculation we can now continue to dive into more complex algorithms

Applications

- We can calculate the eigenvectors and eigenvalues of the Laplacian of an LC Graph
- We can use the companion matrix to find the roots of polynomials over the LC field

$$C(p) = \begin{bmatrix} 0 & 0 & \dots & 0 & -c_0 \\ 1 & 0 & \dots & 0 & -c_1 \\ 0 & 1 & \dots & 0 & -c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -c_{n-1} \end{bmatrix}$$

- Since the field of Puiseux series is a subfield of the LC field, we can find roots for polynomials over the Puiseux series

- This problem is usually solved by the **Newton-Puiseux algorithm** [BriKnö] [4]
- The Newton Puiseux algorithm has various applications in algebraic geometry, the study of algebraic curves etc.
- We can also see applications in the study of differential equations [Li, 2005] [5] and [Ayad, 2015] [6]

---

[4] Brieskorn, Egbert, and Horst Knörrer, Plane Algebraic Curves: Translated by John Stillwell. pp. 370–383, Springer Science & Business Media, 2012.

[5] Ali Ayad, Ali Fares, Youssef Ayyad, Raafat Tarraf, Electronic Journal of Differential Equations, Vol. 2015 (2015), No. 135, pp. 1–7, https://ejde.math.txstate.edu/Volumes/2015/135/ayad.pdf

[6] H. Li, P. J. Olver and G. Sommer (Eds.), IWMM-GIAE 2004, LNCS 3519 pp. 5–17, 2005, Springer-Verlag Berlin Heidelberg 2005

Thank you

Thank you for listening

Seeing a theorem develop in our mind is like observing a flower that is touched by the first rays of sunshine in spring

References


Andrew Barnert
https://github.com/abarnert/levicivita

References

Hao Ze

Fast Power Series Inversion: Newton's Iteration and the Middle Product Optimization

http://www.cecm.sfu.ca/CAG/theses/haoze.pdf

Khodr Shamseddine and Martin Berz

Analysis on the Levi-Civita field, a brief overview

2010

References

Egbert Brieskorn and Horst Knörrer
Plane Algebraic Curves: Translated by John Stillwell. pp. 370–383
Springer Science & Business Media, 2012.

Ali Ayad, Ali Fares, Youssef Ayyad, Raafat Tarraf
Electronic Journal of Differential Equations, Vol. 2015 (2015), No. 135, pp. 1–7
https://ejde.math.txstate.edu/Volumes/2015/135/ayad.pdf

H. Li, P. J. Olver and G. Sommer (Eds.)
IWMM-GIAE 2004, LNCS 3519 pp. 5–17, 2005
Springer-Verlag Berlin Heidelberg 2005

References

 Ragon Ebker
https://github.com/RagonEbker/Levi-Civita-Eigenvalue