

Anhang B: Eigenwerte und Eigenvektoren

Eigenwerte und Eigenvektoren reeller symmetrischer Matrizen, Jacobiverfahren

Eine reelle symmetrische Matrix ist als normale Matrix diagonalisierbar mit einem vollen Satz Eigenvektoren, wobei die Eigenvektoren zu verschiedenen Eigenwerten orthogonal sind.⁵

[HMMatrix](#)⁶ stellt zur Berechnung der Eigenwerte und Eigenvektoren von reellen symmetrischen Matrizen in der Programmiersprache C die Funktionen `m_jacobi()` und `m_jacobi2()` zur Verfügung und in der Programmiersprache C++ die Methode `Matrix::Jacobi()`. Diese Funktionen nutzen zur Berechnung der Eigenwerte und Eigenvektoren das zyklische Jacobiverfahren.⁷ Zwei Beispiele in den Programmiersprachen C++ und C zeigen die Anwendung der genannten Funktionen:

```
// Jacobiverfahren in C++
#include "matrix.hpp"
#include <iostream>

int main()
{
    using namespace std;
    using namespace HMMatrix;

    Matrix a(3), eig(3), vec(3);
    a.Row(0) = 2, -1, 0;
    a.Row(1) = -1, 2, -1;
    a.Row(2) = 0, -1, 2;

    a.Jacobi(eig, vec);
    // Die drei Eigenvektoren stehen in den Spalten von "vec",
    // die Eigenwerte auf der Hauptdiagonalen von "eig".

    cout << MatrixFormat(6, 3);
    for(size_t i = 0; i < 3; ++i)
    {
        cout << i + 1 << ". Eigenwert: ";
        cout << eig(i, i) << "\n";

        cout << i + 1 << ". Eigenvektor: ";
        cout << vec.GetCol(i) << "\n";
    }
}
```

```
C:\Windows\
1. Eigenwert: 3.41421
1. Eigenvektor: 0.500; -0.707; 0.500
2. Eigenwert: 0.585786
2. Eigenvektor: 0.500; 0.707; 0.500
3. Eigenwert: 2
3. Eigenvektor: -0.707; -0.000; 0.707
```

Abbildung 2 – Ausgabe des C++-Programmbeispiels zum Jacobiverfahren

⁵ Aus: Wikipedia, Die freie Enzyklopädie. Stand: 12. Mai 2014, 19:19 UTC. URL: http://de.wikipedia.org/w/index.php?title=Symmetrische_Matrix&oldid=130343678 (Abgerufen: 21. Juli 2014, 09:21 UTC)

⁶ Download im Internet: <http://kuepper.userweb.mwn.de/software.htm>

⁷ Siehe z. B. Heinrich Sormann, Numerische Methoden in der Physik (515.421), Skriptum WS2012/13, Kapitel 7, Technische Universität Graz

```

/* Jacobiverfahren in C */
#include "matrix.h"
#include <stdio.h>
#define DIM 3

int main(void)
{
    double v[DIM], eig[DIM][DIM], vec[DIM][DIM];
    double a[DIM][DIM] =
    {
        { 2, -1, 0 },
        { -1, 2, -1 },
        { 0, -1, 2 }
    };

    m_jacobi(DIM, a, eig, vec);
    /* Die drei Eigenvektoren stehen in den Spalten von "vec",
       die Eigenwerte auf der Hauptdiagonalen von "eig". */

    for(size_t i = 0; i < 3; ++i)
    {
        printf("%d. Eigenwert: %f\n", (int)i + 1, eig[i][i]);
        printf("%d. Eigenvektor: ", (int)i + 1);
        m_get_col(DIM, i, vec, v);
        v_print(DIM, v);
    }

    return 0;
}

```

```

C:\Windows\system32\cmd
1. Eigenwert: 3.414214
1. Eigenvektor: 0.50 -0.71 0.50
2. Eigenwert: 0.585786
2. Eigenvektor: 0.50 0.71 0.50
3. Eigenwert: 2.000000
3. Eigenvektor: -0.71 -0.00 0.71

```

Abbildung 3 – Ausgabe des C-Programmbeispiels zum Jacobiverfahren

Betragsgrößer Eigenwert einer Matrix, von-Mises-Vektoriteration

Die Potenzmethode, Vektoriteration oder von-Mises-Iteration (nach Richard von Mises) ist ein numerisches Verfahren zur Berechnung des betragsgrößten Eigenwertes und des dazugehörigen Eigenvektors einer Matrix.⁸

HMMatrix stellt die C-Funktionen `m_mises()`, `m_mises2()` sowie die C++-Methode `Matrix::Mises()` zur Verfügung, um den betragsgrößten Eigenwert (inkl. Eigenvektor) einer Matrix zu berechnen.

Insbesondere bei doppelten, eng benachbarten oder komplexen Eigenwerten konvergiert das Verfahren oft nur langsam oder überhaupt nicht. In diesen Fällen ist es sinnvoll, statt "m_mises" die Funktion "m_mises2" aufzurufen (in C-Programmen) und dort das Iterationsende über die beiden Parameter "max_iter" (maximale Anzahl der Iterationen) bzw. "epsilon" (gewünschte Genauigkeit) gezielt einzustellen. Bei der Methode `Matrix::Mises()` (in C++-Programmen) existieren zu diesem Zweck zwei optionale Parameter.

⁸ Aus: Wikipedia, Die freie Enzyklopädie. Stand: 21. Juni 2014, 15:03 UTC. URL: <http://de.wikipedia.org/w/index.php?title=Potenzmethode&oldid=131501250> (Abgerufen: 21. Juli 2014, 11:40 UTC)

```

// -----
// von-Mises-Vektoriteration in C++
// -----
#include "matrix.hpp"
#include <iostream>

int main()
{
    using namespace std;
    using namespace HMMatrix;

    Matrix a(3);
    Vector v(3);
    double e;

    a.Row(0) = 2, -1, 0;
    a.Row(1) = -1, 2, -1;
    a.Row(2) = 0, -1, 2;
    v = a.Mises(e);

    cout << MatrixFormat(6, 3);
    cout << "Eigenwert mit groesstem Betrag: " << e;
    cout << "\nDazugehoeriger Eigenvektor: " << v;
}

```

```

C:\Windows\system32\cmd.exe
Eigenwert mit groesstem Betrag: 3.41421
Dazugehoeriger Eigenvektor: -0.500; 0.707; -0.500

```

```

/* ----- */
/* von-Mises-Vektoriteration in C */
/* ----- */
#include "matrix.h"
#include <stdio.h>
#define DIM 3

int main(void)
{
    double e, v[DIM];
    double a[DIM][DIM] =
    {
        { 2, -1, 0},
        {-1, 2, -1},
        { 0, -1, 2}
    };

    m_mises(DIM, a, &e, v);
    printf("Eigenwert mit groesstem Betrag: %f\n", e);
    printf("Dazugehoeriger Eigenvektor:");
    v_print(DIM, v);
    return 0;
}

```

```

C:\Windows\system32\cmd.exe
Eigenwert mit groesstem Betrag: 3.414214
Dazugehoeriger Eigenvektor: -0.50 0.71 -0.50

```

Berechnung von Eigenwerten über das charakteristische Polynom

Nicht-symmetrische Matrizen besitzen im Allgemeinen komplexe Eigenwerte und Eigenvektoren. Diese können mit den bisher vorgestellten Funktionen bzw. Methoden allerdings nicht ermittelt werden.

Die Berechnung der komplexen Eigenwerte einer nicht-symmetrischen Matrix ist mit HMMatrix über eine Nullstellenbestimmung des charakteristischen Polynoms möglich. Vorsicht ist bei größeren Matrizen geboten und auch dann, wenn die Eigenwerte mit hoher Genauigkeit benötigt werden. Dieser Rechenweg ist numerisch nicht stabil!

Die folgenden Beispiele zeigen die Bestimmung der Eigenwerte einer 3x3-Matrix. Das erste Beispiel in der Programmiersprache C, das zweite Beispiel in C++. Die in den Beispielen verwendete Matrix besitzt zwei komplexe Eigenwerte $\lambda_{1,2} = 2 \pm 3i$ und einen reellen Eigenwert $\lambda_3 = 2$.

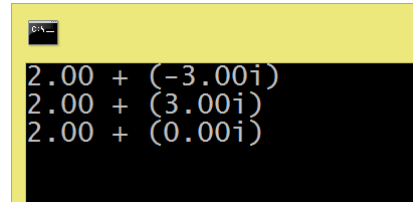
```

/* ----- */
/* Berechnung komplexer Eigenwerte in C */
/* ----- */
#include "matrix.h"
#include <stdio.h>
#define DIM 3

int main(void)
{
    int i;
    double mat[DIM][DIM] = { { 2, -1, 2 }, { 1, 2, -2 }, { -2, 2, 2 } };
    double char_pol[DIM + 1];
    HM_COMPLEX eigval[DIM];

    /* Reelle und komplexe Eigenwerte berechnen und auf dem Bildschirm ausgeben */
    m_charpol(DIM, mat, char_pol);
    v_droots(DIM + 1, char_pol, eigval);
    for(i = 0; i < DIM; ++i) /* Real- und Imaginärteil der Eigenwerte ausgeben */
        printf("%.2f + (%.2fi)\n", MAT_REAL(eigval[i]), MAT_IMAG(eigval[i]));
    return 0;
}

```



```

2.00 + (-3.00i)
2.00 + (3.00i)
2.00 + (0.00i)

```

```

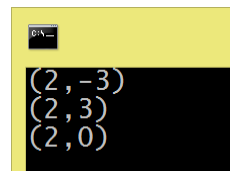
// -----
// Berechnung komplexer Eigenwerte in C++
// -----
#include <iostream>
#include "matrix.hpp"

int main()
{
    using namespace std;
    using namespace HMMatrix;

    Matrix mat(3);
    mat.Row(0) = 2, -1, 2;
    mat.Row(1) = 1, 2, -2;
    mat.Row(2) = -2, 2, 2;

    /* Reelle und komplexe Eigenwerte berechnen und auf dem Bildschirm ausgeben */
    auto char_pol = mat.CharPol();
    auto eigenval = char_pol.Roots();
    for(auto it = eigenval.begin(); it != eigenval.end(); ++it)
    {
        complex<double> e = *it;
        cout << e << "\n";
    }
}

```



```

(2, -3)
(2, 3)
(2, 0)

```