

## Eigenwertprobleme

Zweite Form linearer Gleichungssystem sind **homogene Gleichungssysteme**.

Die Problemstellung hier ist

$$\mathbf{A} \cdot \mathbf{x} = \lambda \mathbf{x}$$

(wir betrachten hier nicht die allgemeinere Gleichung  $\mathbf{A} \cdot \mathbf{x} = \lambda \mathbf{B} \cdot \mathbf{x}$ )

Solche Gleichungen ergeben sich in natürlicher Weise bei der Diskretisierung von Problemen wie

- Eigenschwingungen von Ketten, Stäben, ...
- Quantenmechanischen Bindungszuständen
- ...

Beispielsweise erhält ergibt sich für die radiale Schrödinger Gleichung (mit Drehimpuls 0) wie üblich

$$-u''(r) + 2\mu V(r) u(r) = 2\mu E u(r)$$

in diskretisierter Form findet man dann direkt  
( $h$  ist die Schrittweite und  $\mu$  die reduzierte Masse)

$$-(u_{i-1} - 2u_i + u_{i+1}) + 2\mu h^2 V_i u_i = 2\mu E h^2 u_i$$

Aus den Randbedingungen kann man  $u_0 = 0$  und  $u_{N+1} = 0$  festlegen.

Damit findet man in der Matrixform die folgende Gleichung

$$\begin{pmatrix} 2 + 2\mu h^2 V_1 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 2 + 2\mu h^2 V_2 & -1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -1 & 2 + 2\mu h^2 V_{N-1} & -1 \\ 0 & \cdots & \cdots & 0 & -1 & 2 + 2\mu h^2 V_N \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-1} \\ u_N \end{pmatrix} = 2\mu h^2 E \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-1} \\ u_N \end{pmatrix}$$

Und das entspricht dem Eigenwertproblem

$$\mathbf{A} \cdot \mathbf{u} = 2\mu h^2 E \mathbf{u}$$

wobei in diesem spezielle Fall die Matrix **A reell, symmetrisch und tridiagonal** ist.

Wir führen die Notation für eine allgemeine, reelle Matrix ein. Wir nehmen im folgenden nur an, dass die Eigenvektoren eine vollständige Basis des Vektorraums bilden.

Sei

$$\mathbf{A} \cdot \mathbf{v}_R = \lambda \mathbf{v}_R$$

Dabei sind  $\mathbf{v}_R$  die sogenannten **rechtseitigen Eigenvektoren**.

Die **Eigenwerte** sind die Nullstellen des **charakteristischen Polynoms**

$$p(\lambda) = \det(\mathbf{A} - \lambda \mathbf{I})$$

da nur für diese Werte nicht-triviale Lösungen der Eigenwertgleichung existieren können.

Analog definiert man linksseitige Eigenvektoren:

$$\mathbf{v}_L^T \cdot \mathbf{A} = \lambda \mathbf{v}_L^T$$

Durch Transponieren kann man zeigen, dass die Eigenwerte identisch sind.

Die beiden Sätze an Eigenvektoren kann man zu Matrizen zusammenfassen:

$$\mathbf{A} \cdot \mathbf{V}_R = \mathbf{V}_R \cdot \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_N \end{pmatrix} \quad \text{und} \quad \mathbf{V}_L \cdot \mathbf{A} = \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_N \end{pmatrix} \cdot \mathbf{V}_L$$

Das Produkt  $\mathbf{V}_L \cdot \mathbf{V}_R$  vertauscht damit mit der Diagonalmatrix der Eigenwerte. Unter der Annahme, dass alle Eigenwerte verschieden sind, bedeutet das, dass das Produkt selber diagonal ist. D.h. wir können die Normierung so wählen, dass links- und rechtsseitige Eigenvektoren orthonormiert sind.

Damit  $\mathbf{V}_L \cdot \mathbf{V}_R = \mathbb{I}$ , d.h.  $\mathbf{V}_L = \mathbf{V}_R^{-1}$  und deswegen definiert  $\mathbf{V}_R$  eine **Ähnlichkeitstransformation**, die die Matrix  $\mathbf{A}$  diagonalisiert.

$$\mathbf{A}' = \mathbf{V}_R^{-1} \cdot \mathbf{A} \cdot \mathbf{V}_R = \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_N \end{pmatrix}$$

Wie kann man Eigenwerte und Eigenvektoren bestimmen?

**Generell** gilt: das ist **schwierig** aber es gibt komplizierte Programm bibliotheken, die oft Verfahren kombinieren, um eine stabile Lösung zu finden.

Hier werden nur ein paar grundlegende Ideen vorgestellt.

- 1) Möglichkeit: man ist nur an ein paar Eigenwerten und -vektoren interessiert  
Typische Lösung ist hier "**Inverse Vektoriteration**".

Der Startpunkt ist ein Schätzwert für den Eigenwert, für den die Gleichung

$$(\mathbf{A} - \tau \mathbf{I}) \cdot \mathbf{y} = \mathbf{b}$$

gelöst wird.  $\mathbf{b}$  ist hierbei zunächst ein zufälliger Startvektor, der **normiert** sei.  
Die Lösung und Inhomogenität kann man durch Eigenvektoren ausdrücken

$$\mathbf{y} = \sum_j \alpha_j \mathbf{v}_j \quad \mathbf{b} = \sum_j \beta_j \mathbf{v}_j$$

Damit erhält man für die Lösung die Relation  $\alpha_j = \frac{\beta_j}{\lambda_j - \tau}$

Damit erhält der Eigenwert, der am nächsten an  $\tau$  liegt, ein großes Gewicht in der Lösung. Durch normalisieren und Iteration kann man damit den dazugehörigen Eigenvektor bestimmen in jedem Schritt wird ein verbesserter Eigenvektor (und wenn nötig auch Eigenwert) durch

$$\mathbf{b}' = \frac{\mathbf{y}}{|\mathbf{y}|} \quad \text{und} \quad \tau' = \tau + \frac{1}{\mathbf{b} \cdot \mathbf{y}} \quad \text{bestimmt.}$$

2) Möglichkeit: Man findet eine Ähnlichkeitstransformation, die die Matrix diagonalisiert. Diese kann aus Schritten zusammengesetzt werden. Im Allgemeinen braucht so eine schrittweise Diagonalisierung unendliche viele Schritte. Die Verfahren konvergieren aber in der Regel schnell.

$$\mathbf{A} \longrightarrow \mathbf{P}_1^{-1} \cdot \mathbf{A} \cdot \mathbf{P}_1 \longrightarrow \mathbf{P}_2^{-1} \mathbf{P}_1^{-1} \cdot \mathbf{A} \cdot \mathbf{P}_1 \mathbf{P}_2 \cdots$$

also  $\mathbf{V}_R = \mathbf{P}_1 \mathbf{P}_2 \cdots$

Damit findet man alle Eigenwerte und Eigenvektoren einer Matrix gleichzeitig.

Das **Jacobi Verfahren** für **symmetrische** Matrizen basiert auf **Jacobi Rotationen**

$$\mathbf{P}_{pq} = \begin{pmatrix} 1 & & \cdots & & \\ \vdots & \cos \phi & \cdots & \sin \phi & \\ & \vdots & \ddots & \vdots & \\ & -\sin \phi & \cdots & \cos \phi & \\ & & \vdots & & 1 \end{pmatrix} \begin{matrix} p \\ & q \\ p & q \end{matrix}$$

Die Wahl dieser Matrix garantiert, dass in  $\mathbf{A}' = \mathbf{P}_{pq}^{-1} \cdot \mathbf{A} \cdot \mathbf{P}_{pq}$  nur die Spalten und Zeilen  $p$  und  $q$  geändert werden.

Beim Jacobi Verfahren wählt man den Drehwinkel, so dass  $a'_{pq} = 0$



$$a'_{pp} = \cos^2 \phi a_{pp} + \sin^2 \phi a_{qq} - 2 \cos \phi \sin \phi a_{pq}$$

$$a'_{rp} = \cos \phi a_{rp} - \sin \phi a_{rq}$$

$$a'_{qq} = \cos^2 \phi a_{qq} + \sin^2 \phi a_{pp} + 2 \cos \phi \sin \phi a_{pq}$$

$$a'_{pq} = (\cos^2 \phi - \sin^2 \phi) a_{pq} + \cos \phi \sin \phi (a_{pp} - a_{qq}) = 0$$

$$a'_{rq} = \cos \phi a_{rq} + \sin \phi a_{rp}$$

Man kann zeigen, dass mit solchen Transformationen die Größe der Nicht-Diagonalelemente kontinuierlich abnimmt, wenn man alle Nicht-Diagonalelemente der Reihe nach durch geht. Nach 6-10 dieser Durchgänge erhält man üblicherweise eine konvergierte Diagonalmatrix und durch Multiplikation die dazugehörigen Eigenvektoren.

Solche und ähnliche Transformationen kann man auch nutzen, um die Matrix mit einer endlichen Anzahl von Schritten in tridiagonale oder Hessenberg-Form

$$H = \begin{pmatrix} * & * & * & \cdots & * \\ * & * & * & \cdots & * \\ 0 & * & * & \cdots & \vdots \\ 0 & \ddots & * & * & * \\ 0 & \cdots & 0 & * & * \end{pmatrix}$$

zu bringen.

Beispielsweise kann man nach der **Givens Methode**  $\mathbf{P}_{\mathbf{pq}}$  so wählen, dass  $a'_{q \ p-1} = 0$ .

Wenn man dann die Reihenfolge  $\mathbf{P}_{23}; \mathbf{P}_{24}; \mathbf{P}_{25}; \dots; \mathbf{P}_{2N}; \mathbf{P}_{34}; \dots; \mathbf{P}_{3N}; \dots; \mathbf{P}_{N-1N}$  wählt, bleiben bereits Null gesetzte Elemente Null.

Die **Householder Methode** wird ebenfalls oft verwendet, um die Matrix in Hessenberg-Form zu bringen (basiert auf einer etwas anderen Wahl von Transformationen).

In Hessenberg oder tridiagonaler Form kann man dann effizient den **QR Algorithmus** anwenden, um wieder die Eigenwerte und Eigenvektoren zu erhalten.

Ausgangspunkt ist eine Zerlegung der Matrix (bereits tridiagonal oder Hessenberg Form) in eine orthogonale Matrix und eine obere Dreiecksmatrix

$$\mathbf{A} = \mathbf{Q} \cdot \mathbf{R}$$

Vertauschen der beiden Matrizen definiert eine orthogonale Transformation

$$\mathbf{A}' = \mathbf{R} \cdot \mathbf{Q} = \mathbf{Q}^{-1} \cdot \mathbf{A} \cdot \mathbf{Q}$$

Die wiederholte Anwendung dieser Transformation ergibt eine konvergente Reihe von Matrizen

$$\mathbf{A}_s = \mathbf{Q}_s \cdot \mathbf{R}_s \quad \rightarrow \quad \mathbf{A}_{s+1} = \mathbf{R}_s \cdot \mathbf{Q}_s$$

wobei  $\mathbf{A}_{s+1}$  gegen eine **obere Dreiecksmatrix** konvergiert

(solange die Eigenwerte verschieden sind, nicht leicht zu sehen, siehe Stoer/Bulirsch)

Die Diagonalelemente der Dreiecksmatrix sind bereits die Eigenwerte  
(leicht zu sehen, wenn man das charakteristische Polynom bestimmt).

Für symmetrische Matrizen ist die Matrix bereits diagonal.

(Bei nicht-symmetrischen Matrizen können Paare von komplex konjugierten Eigenwerten auftreten. In diesen Fall, hat die Matrix 2x2 Submatrizen auf der Diagonalen)  
(siehe Stoer/Bulirsch)

Implementierungen des Algorithmus nutzen üblicherweise ebenfalls Verschiebungen, wie bei der inversen Iteration, um die Konvergenz zu beschleunigen und Vertauschungen der Zeilen oder Spalten, um Rundungsfehler zu minimieren.

Man sieht die effiziente Bestimmung von Eigenwerten und Vektoren einer allgemeinen Matrix ist nicht einfach. Glücklicherweise gibt sowohl in GSL, LAPACK als auch in anderen Programm Paketen bereits Implementierungen, die stabil und schnell sind.

Das folgende Beispiel zeigt den Aufruf einer LAPACK Routine zur Diagonalisierung einer allgemeinen, reellen Matrix.

Benötigt wird die Routine

## DGEEV(JOBVL,JOBVR,N,A,LDA,WR,WI,VL,LDVL,VR,LDVR,WORK,LWORK,INFO)

- **S,D,C,Z** zeigen Gleitkommaformat: single, double, complex, double complex
- **GE,SY,GB,PO** unterscheiden general, symmetric, general band, postiv definite
- **EV** schließlich legt die Operation fest: eigenvalues and vectors

Parameter sind:

<b>JOBVL/R</b>	'V' oder 'N' zeigt ob entsprechende Vektoren berechnet werden sollen
<b>N</b>	Dimension der Matrix
<b>A</b>	Matrix beim Aufruf, wird überschrieben
<b>LDA</b>	"Leading Dimension" von A (normalerweise = N)
<b>WR,WI</b>	Real- und Imaginärteil der Eigenwerte
<b>VL,VR</b>	Links- und rechtsseitige Eigenvektoren
<b>LDVL,LDVR</b>	"Leading Dimension" von VL,VR (normalerweise = N)
<b>WORK</b>	Hilfsfeld für Zwischenspeicher (Länge <b>LWORK</b> $\geq 4N$ )
<b>INFO</b>	int Zahl mit Fehlercode, sollte = 0 sein, wenn Aufruf erfolgreich war.

Das Beispiel bestimmt die Eigenwerte und rechts- und linksseitige Eigenvektoren und überprüft die Relation

$$\mathbf{A} = \mathbf{V}_R \cdot \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_N \end{pmatrix} \cdot \mathbf{V}_L$$

Die Eigenwerte können dabei komplex sein. Es treten dann Paare von komplex konjugierten Eigenwerten auf. In diesen Fall sind auch die Eigenvektoren komplex konjugiert zu einander.

(durch Zerlegen in Real- und Imaginärteil lässt sich das leicht zeigen)

$$\lambda_1 = \lambda_R + i\lambda_I \quad \mathbf{v}_1 = \mathbf{v}_R + i\mathbf{v}_I$$

$$\lambda_2 = \lambda_R - i\lambda_I \quad \mathbf{v}_2 = \mathbf{v}_R - i\mathbf{v}_I$$

Die Routine speichert in diesen Fall den Real- und Imaginärteil der Eigenvektoren in aufeinanderfolgenden Vektoren in VR und VL.

## Wieder Programm mit dem Aufruf einer FORTRAN LAPACK Routine

```
/*      Beispiel-5.2-eigen.c    15.05.2012      */

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<float.h>
#include<limits.h>
#include<sys/time.h>

#define FINT int /* definiere den Standard F-integer auf der Maschine */

/* dgeev_ als externe Funktion deklarieren
   Die hier gegebenen Parameter stimmen mit der Definition
   in LAPACK Bibliothek ueberein.
   Uebergabe der Adresse eines Objekts auch fuer nicht-Felder
   (in FORTRAN ist das Standard)
   FORTRAN Name DGEEV    ->    dgeev_ */

extern void dgeev_(char *jobvl,char *jobvr,FINT *n,double *a,FINT *lda,
                  double *wr,double *wi,double *vl,FINT *ldvl,
                  double *vr,FINT *ldvr,
                  double *work,FINT *lwork,FINT *info);
```

**Routine DGEEV berechnet Eigenwert und Vektoren einer reellen, allgemeinen Matrix**

```

int main()
{
    /* Dimension im Beispiel = 3 */
    const long n=3;
    double A[n*n]; /* Feld fuer die Matrix mit 3*3 Elementen, REAL*8 in FORTRAN => double */
    double VL[n*n],VR[n*n]; /* Felder fuer Links- und Rechts-Eigenvektoren
        beide Felder haben Konventionen fuer reelle und imaginaire Anteil */
    double WR[n],WI[n]; /* Felder fuer Real- und Imaginaerteil der Eigenwerte */
    double work[4*n]; /* Hilfsfeld, kann uns egal sein was dort gespeichert wird */

    /* INTEGER Variabel in FORTRAN = long = 4 Bytes */
    FINT info; /* info sollte Null sein beim Ver */
    FINT lda=n,ldv1=n,ldvr=n; /* Dimension der Felder */
    FINT lwork=4*n; /* Dimension des work-Feldes (Sollte mind. 4*n sein) */
    FINT dim=n; /* Dimension der Matrix */
    char jobvl='V',jobvr='V'; /* char Variablen bestimmen ob VL und VR berechnet werden
        sollen -> hier berechnen, 'I' */
    int i,j,k; /* Fuer Ausgabe der Ergebnisse etc. */
    double timeused; /* Variablen zur Zeitmessung */
    struct timeval tvstart,tvend;

    double vright[2*n*n],vleft[2*n*n],w[2*n];
    double norm[2];
    double help[2];
    /* hier zusaetglich Routine aus time, um Zeit fuer Loesung zu messen */
    gettimeofday(&tvstart,NULL);

    /* Speichere Matrix als 1-dimensionales Feld */
    A[0+n*0]=5.0; A[0+n*1]=8.0; A[0+n*2]=7.0;
    A[1+n*0]=4.0; A[1+n*1]=12.0; A[1+n*2]=32.0;
    A[2+n*0]=6.0; A[2+n*1]=4.0; A[2+n*2]=19.0;

    /* Versuch einer lesbaren Ausgabe der Matrix A */
    for(i=0;i<n;i++)
    {
        printf("A: ");
        for(j=0;j<n;j++)
        {
            printf(" %15.6e ",A[i+n*j]);
        }
        printf("\n"); /* neue Zeile */
    }
}

```

wie in DGESV Fall: nutze FINT um Ganzzahl-Datentyp flexible festzulegen

Zeitmessung mit Zeitvariablen als struct timeval

führe hier komplexe VR,VI und lambda (w) ein  
= double Feld mit doppelter Größe

definiere Beispiel Matrix A

... und drucke sie aus

```
/* Aufruf der Bibliotheksfunktion, um Eigenwerte und Vektoren zu bestimmen */
dgeev_(&jobvl,&jobvr,&dim,A,&llda,WR,WI,VL,&ldvl,VR,&ldvr,work,&lwork,&info);
```

```
/* Welchen Wert wurde in info gespeichert ? */
printf(" \n\n info: %ld \n\n",info);
```

```
/* definiere complex vleft,vright,w fuer einfache Algebra */
for(i=0;i<n;i++) /* i zählt die Eigenwert und Vektoren durch */
{ /* Konvention: fuer Imag W == 0 -> Vektoren sind auch reell */
  if(WI[i]==0.0)
  { w[2*i]=WR[i]; /* w[i] = WR[i] + I WI[i] */
    w[2*i+1]=0.0;

    for(j=0;j<n;j++) /* j zählt die Komponenten der Vektoren durch
    { /* V? [j+n*i] -> j Komponente vom i Eigenvektor (links/rechts)
      vleft[2*(j+n*i)]=VL[j+n*i]; /* Realteil */
      vleft[2*(j+n*i)+1]=0.0; /* Imaginärteil */
      vright[2*(j+n*i)]=VR[j+n*i];
      vright[2*(j+n*i)+1]=0.0;
    } else /* w ist komplex -> Eigenvektoren auch */
    { /* komplexe Eigenwerte treten als Paar auf */
      w[2*i] = WR[i]; /* w[i] = WR[i] + I WI[i]
      w[2*i+1] = WI[i];
      w[2*(i+1)] = WR[i]; /* w[i+1] = WR[i] - I WI
      w[2*(i+1)+1]=-WI[i];

      for(j=0;j<n;j++) /* j zählt wieder die Komponenten des Vektors */
      { /* V? [j+n*i] -> j Komponente vom i Eigenvektor (links/rechts) */
        /* hier: V? [*+n*i] = Realteil V? [*+n*(i+1)] = Imaginärteil */
        /* erster Vektor ist real + I imag */
        vleft[2*(j+n*i)] = VL[j+n*i];
        vleft[2*(j+n*i)+1] = VL[j+n*(i+1)];
        vright[2*(j+n*i)] = VR[j+n*i];
        vright[2*(j+n*i)+1] = VR[j+n*(i+1)];
        /* zweiter Vektor ist real - I imag */
        vleft[2*(j+n*(i+1))] = VL[j+n*i];
        vleft[2*(j+n*(i+1))+1] = -VL[j+n*(i+1)];
        vright[2*(j+n*(i+1))] = VR[j+n*i];
        vright[2*(j+n*(i+1))+1] = -VR[j+n*(i+1)]; }

      /* hier wurden zwei i verarbeitet, erhöhe deshalb i++ */
    } }
  }
```

**Aufruf von DGEEV**

**Ausgabe: info**

**Umspeichern der Ergebnisse**

**Eigenwert ist reell:  
Eigenvektor ist ebenfalls reell**

**Eigenwert ist komplex:  
betrachte die nächsten zwei Eigenwerte  
und setze komplex konjugierte Paare  
zusammen**

**betrachte nun die nächsten zwei Vektoren**

**Realteile sind identisch**

**Imaginärteile haben unterschiedliches  
Vorzeichen**

**... überspringe nächste Position ...**

```
/* Routine normiert <VR(k) | VR(k)>=<VL(k) | VL(k)> =1
wir benoetigen hier < VR(k) | VL(k) > = 1
bestimme deswegen diese Norm und skaliere VR entsprechend
Orthogonalitaet <VR(i)|VL(j)> = 0 fuer i!=j gilt automatisch
(ausprobieren!!!) */
```

```
/* bestimme norm und skaliere */
```

```
for(i=0;i<n;i++)
```

```
{
```

```
norm[0]=0.0; /* norm fuer i-tes Paar VL, VR bestimmen,
```

```
norm[1]=0.0;
```

```
for(k=0;k<n;k++)
```

```
{ norm[0]=norm[0]+vleft[2*(k+n*i)] *vright[2*(k+n*i)+1];
+ vleft[2*(k+n*i)+1]*vright[2*(k+n*i)+1];
```

**bestimme Norm aus  $V_L \cdot V_R$**

```
norm[1]=norm[1]-vleft[2*(k+n*i)+1] *vright[2*(k+n*i)] /* Imaginarteil */
+ vleft[2*(k+n*i)]*vright[2*(k+n*i)+1]; }
```

```
for(k=0;k<n;k++) /* alle Komponenten von VR(i) durch norm dividieren */
```

```
{ help[0]=vright[2*(k+n*i)];
help[1]=vright[2*(k+n*i)+1];
```

**setze VR/norm aus Imaginär und Realteil zusammen**

```
vright[2*(k+n*i)] =(help[0]*norm[0]+help[1]*norm[1])/(norm[0]*norm[0]+norm[1]*norm[1]);
vright[2*(k+n*i)+1]=(help[1]*norm[0]-help[0]*norm[1])/(norm[0]*norm[0]+norm[1]*norm[1]);}
```

```
} . . .
```

```

/* Finde orginal Matrix als sum_k |vr(k)><vl(k)| A | vr(k) > < vl(k) | */

for(i=0;i<n;i++)
{ printf("ANEU: ");
 for(j=0;j<n;j++)
{ A[i+n*j]=0.0;
 for(k=0;k<n;k++)
  /*summiere ueber alle Eigenvektorpaare */
  { /* berechnen hier nur den Realteil */
    /* += < i | vl(k) > * w[k] * < vr(k) | j > */
    A[i+n*j]+=w[2*k] *vright[2*(i+k*n)] *vleft[2*(j+k*n)]
    -w[2*k+1]*vright[2*(i+k*n)+1]*vleft[2*(j+k*n)]
    +w[2*k+1]*vright[2*(i+k*n)] *vleft[2*(j+k*n)+1]
    +w[2*k] *vright[2*(i+k*n)+1]*vleft[2*(j+k*n)+1];
  }
  printf(" %15.6le ",A[i+n*j]); /* drucke das Ergebnis fuer A_ij gleich aus */
}
printf("\n"); /* neue Zeile */
}

/* Wie lange hat das in sec gedauert ? */
gettimeofday(&tvend,NULL);
timeused=tvend.tv_sec-tvstart.tv_sec;
timeused=timeused+(tvend.tv_usec-tvstart.tv_usec)*1e-6; /* Zeitdifferenz in sec */

printf(" \n\n Zeit: %15.6le \n\n",timeused); }

```

$$\mathbf{A} = \mathbf{V}_R \cdot \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_N \end{pmatrix} \cdot \mathbf{V}_L$$

**bestimme Laufzeit des Programmes**

*/\* compilieren und aufrufen mit*

*cipw01> gcc beispiel-5.2-eigen.c -l lapack -l gfortran -o eigen  
cipw01> ./eigen*

*Ergebnis:*

A:	5.000000e+00	8.000000e+00	7.000000e+00
A:	4.000000e+00	1.200000e+01	3.200000e+01
A:	6.000000e+00	4.000000e+00	1.900000e+01

*info: 0*

ANEU:	5.000000e+00	8.000000e+00	7.000000e+00
ANEU:	4.000000e+00	1.200000e+01	3.200000e+01
ANEU:	6.000000e+00	4.000000e+00	1.900000e+01

*Zeit: 1.290000e-04*

*\*/*