Musterlösung Aufgabe 37

Aufgabe 37: (Implementierung der Wielandt-Rayleigh-Iteration)
Berechne die 10 größten Eigenwerte der 1000×1000-Matrix

$$A \;=\; \begin{pmatrix} 1 & \alpha & 0 & \dots & 0 \\ \alpha & 2 & \alpha & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \alpha \\ 0 & \dots & 0 & \alpha & 1000 \end{pmatrix} \;, \quad \alpha = 0.26$$

auf mindestens 10 Dezimalstellen relativer Genauigkeit:

a) Seien $\lambda_1 \leq \lambda_2 \leq \dots$ die Eigenwerte. Zeige:

$$\lambda_i \in (i - 0.33, i + 0.33)\;, \quad i = 1, \dots, 1000\;.$$

Anleitung: Betrachte DAD^{-1} mit $D = \mathrm{diag}(1, \dots, 1, \epsilon, 1, \dots, 1)$. Es gibt einen Wertebereich für ϵ, sodaß der Gerschgorin-Kreis um das i.te Diagonalelement von den restlichen Kreisen separiert.

b) Implementiere die inverse Wielandt-Iteration $\vec{x}^{(j+1)} = \dfrac{(A - \mu^{(j)}\,\mathbb{1}\,)^{-1}\vec{x}^{(j)}}{\|(A - \mu^{(j)}\,\mathbb{1}\,)^{-1}\vec{x}^{(j)}\|_2}$ mit Rayleigh-Beschleunigung $\mu^{(j)} = r_A(\vec{x}^{(j)})$. Es bietet sich eine LR-Zerlegung von $A - \mu^{(j)}\,\mathbb{1}$ gemäß Beispiel 4.21 der Vorlesung oder (besser) eine QR-Zerlegung gemäß Aufgabe 28 an (es existiert eine Musterlösung zu Aufgabe 28 im Web). Zur Bestimmung des Eigenwertes $\lambda_i \approx i$ ($i = 991, \dots, 1000$) führe zunächst einen einzelnen inversen Wielandt-Schritt mit dem Spektralparameter $\mu = i$ und $\vec{x}^{(0)} = (1, \dots, 1)^T$ durch, dann gehe zur beschleunigten Iteration über, bis die gewünschte relative Genauigkeit erreicht ist.

Der (wohlkommentierte) Quelltext ist mit abzugeben, notiere die benötigte Rechenzeit, die in der Größenordnung weniger Sekunden liegen sollte.

Anmerkung: Vorsicht bei der Implementierung der Fehlerabschätzung aus Satz 6.3.b): die Fehlerschranke $r_{A^2}(\vec{x}) - r_A(\vec{x})^2$ ist durch Auslöschung gefährdet! Verwende die Darstellung

$$r_{A^2}(\vec{x}) - r_A(\vec{x})^2 = \frac{\langle \vec{y}, \vec{y} \rangle}{\langle \vec{x}, \vec{x} \rangle}\;, \quad \vec{y} = (A - r_A(\vec{x})\;\mathbb{1}\;)\,\vec{x}\;.$$

Musterlösung: a) Für $i = 1, n$ folgt direkt über das Gerschgorin-Kriterium $|\lambda_1 - 1| \leq \alpha$, $|\lambda_n - n| \leq \alpha$, da die Kreise K_1 und K_n von den übrigen separieren:

$$1 + \alpha = 1.26 < 2 - 2\,\alpha = 1.48\ , \quad n - 1 + 2\,\alpha = n - 0.48 < n - \alpha = n - 0.26\ .$$

Betrachte nun $i = 2, \ldots, n-1$. Mit $D = \mathrm{diag}(1, \ldots, \epsilon, \ldots, 1)$ folgt

$$DAD^{-1} \;=\; \begin{pmatrix} 1 & \alpha & & & & & & \\ \alpha & 2 & \alpha & & & & & \\ & \ddots & \ddots & \ddots & & & & \\ & & \alpha & i-1 & \alpha/\epsilon & & & \\ & & & \epsilon\,\alpha & i & \epsilon\,\alpha & & \\ & & & & \alpha/\epsilon & i+1 & \alpha & \\ & & & & & \alpha & \ddots & \ddots \\ & & & & & & \ddots & \ddots \end{pmatrix}$$

Der rechte Rand des Kreises K_{i-1} ist $i - 1 + \alpha + \alpha/\epsilon$, der linke Rand des Kreises K_i ist $i - 2\,\epsilon\,\alpha$. Damit keine Überlappung auftritt, muß

$$i - 1 + \alpha + \frac{\alpha}{\epsilon} \; < i - 2\,\epsilon\,\alpha$$

gelten, d.h.,

$$(-1 + \alpha)\,\epsilon + \alpha < -2\,\epsilon^2\,\alpha \quad \Rightarrow \quad \epsilon^2 - \frac{1-\alpha}{2\,\alpha}\,\epsilon + \frac{1}{2} \; < \; 0\ .$$

Dies liefert die Bedingungen

$$\underbrace{\frac{1-\alpha}{4\,\alpha} - \sqrt{\frac{(1-\alpha)^2}{16\,\alpha^2} - \frac{1}{2}}}_{0.6322..} \; < \; \epsilon \; < \; \underbrace{\frac{1-\alpha}{4\,\alpha} + \sqrt{\frac{(1-\alpha)^2}{16\,\alpha^2} - \frac{1}{2}}}_{0.7908..}$$

Aus Symmetriegründen folgen dieselben Bedingungen für das Separieren der Kreise K_i und K_{i+1}.

Der Gerschgorin-Kreis $K_i = [i - 2\,\epsilon\,\alpha, i + 2\,\epsilon\,\alpha]$ liefert dann mit der Wahl $\epsilon = 0.633$ für $i = 2, \ldots, n-1$ die Abschätzung

$$|\lambda_i - i| \leq 2\,\epsilon\,\alpha = 2 \times 0.633 \times 0.26 = 0.32916\ .$$

b) C-Lösung:

```
#include <stdio.h>
#include <math.h>
#define  n 1000
```

```
#define alpha 0.26
#define alpha2 0.26*0.26

main()
{
  int i,j;
  double mu, x[n+1], d[n+1];
  void QRsolve(double [], double []);
  void normalize(double []);
  double RayleighQuotient(double []);
  double fehler(double [], double);

  for (j=n-9; j<=n; j++)                                /* berechne lambda[j] */
     {
       for (i=1; i<=n; i++) x[i]=1.0;                /* Definiere Startvektor */

       mu=j; for (i=1; i<=n; i++) d[i]=i-mu;    /* einzelner Schritt mit mu=j */
       QRsolve(d,x); normalize(x); mu=RayleighQuotient(x);
       printf("%.14f %.1e\n",mu,fehler(x,mu));
       if (fabs(fehler(x,mu))<1.0e-14*fabs(mu))           /* Abbruchkriterium */
          { printf("\n"); continue; }                /* gehe zum naechsten j */

       for (;;)                     /* Endlos-Schleife bis fehler klein genug */
       { for (i=1; i<=n; i++) d[i]=i-mu;                        /* Rayleigh-     */
         QRsolve(d,x); normalize(x); mu=RayleighQuotient(x); /* beschleunigte */
         printf("%.14f %.1e\n",mu,fehler(x,mu));              /* Schritte      */
         if (fabs(fehler(x,mu))<1.0e-14*fabs(mu))          /* Abbruchkriterium */
          { printf("\n"); break; }                     /* gehe zum naechsten j */
       }
    }
}

/* -------------- subroutine QRsolve -----------------------   */
  void QRsolve(double d[], double x[])
    /* Loese die Gleichung
           ( d[1]  alpha                 ) ( x[1] )    ( b[1] )
           ( alpha  d[2]  alpha          ) ( x[2] )    ( b[2] )
           (       alpha  d[3]  alpha    ) ( x[3] )  = ( b[3] )
           (               ..    ..   .. ) (  ..  )    (  ..  )
         durch QR-Zerlegung der Matrix. Die rechte Seite wird als
         Vektor x uebergeben, die Loesung wird in x gespeichert.
         Die QR-Zerlegung folgt Aufgabe 28.
        */
  {
    int i;
    double t, c, s, gamma, u[n+1], v[n], w[n-1];
```

```
    gamma=alpha;
    for (i=1; i<=n-2; i++)
    {  t=sqrt(d[i]*d[i]+alpha2);      /* Berechnung der   */
       c=d[i]/t; s=alpha/t;           /* Householder-     */
       u[i]=c*d[i]+s*alpha;           /* Transformation   */
       v[i]=c*gamma+s*d[i+1];         /* u,v,w sind die   */
       w[i]=s*alpha;                  /* 3 Baender der    */
       d[i+1]=s*gamma-c*d[i+1];       /* oberen Dreiecks- */
       gamma=-c*alpha;                /* matrix R in A=QR */

       /* Householder-Transformation x -> H^(i)x auf die rechte */
       /* Seite der Gleichung angewendet (siehe Aufgabe 28.c)   */
       t=x[i]; x[i]=c*t+s*x[i+1]; x[i+1]=s*t-c*x[i+1];
    }
    t=sqrt(d[n-1]*d[n-1]+alpha2);  /* letzter Schritt i=n-1 */
    c=d[n-1]/t; s=alpha/t;
    u[n-1]=c*d[n-1]+s*alpha;
    v[n-1]=c*gamma+s*d[n];
    u[n]  =s*gamma-c*d[n];
    t=x[n-1]; x[n-1]=c*t+s*x[n]; x[n]=s*t-c*x[n];  /* x->H^(n-1)x /*

  /* Es gilt      ( u[1] v[1] w[1]                    )
                  (      u[2] v[2] w[2]               )
              R = (            ..   ..    ..          )
                  (                 ..    ..   w[n-2] )
                  (                     u[n-1] v[n-1] )
                  (                             u[n]  )
    Es folgt die R\"ucksubstitution fuer Rx=Q^T b, wo Q^T b im
    Vektor x gespeichert ist: */

    x[n]=x[n]/u[n];
    x[n-1]=(x[n-1]-v[n-1]*x[n])/u[n-1];
    for (i=n-2; i>=1; i--) x[i]=(x[i]-v[i]*x[i+1]-w[i]*x[i+2])/u[i];
  } /* end of QRsolve */

/* -------------- subroutine normalize -----------------------   */
  void normalize(double x[])      /*  x -> x/norm(x) */
  { double norm2;
    int i;
    norm2=0; for (i=1; i<=n; i++) norm2 += x[i]*x[i];
    norm2=sqrt(norm2); for (i=1; i<=n; i++) x[i] /= norm2;
  }

/* -------------- subroutine RayleighQuotient ----------------------- */
  double RayleighQuotient(double x[])         /* x ist als normiert  */
  { double r;                                 /* vorausgesetzt, also */
    int i;                                    /* r = <x,Ax>          */
```

```
    r=x[1]*x[1]; for (i=2;i<=n;i++) r+=i*x[i]*x[i]+2*alpha*x[i]*x[i-1];
    return r;
  }

/* -------------- subroutine fehler  -------------------------------- */
  double fehler(double x[], double mu)    /* berechne Fehlerschranke  */
  { double f, y[n+1];                     /* f = sqrt( <y,y>/<x,x> )  */
                                          /* mit y = (A-mu)x,         */
    int i;                                /* wobei mu= <x,Ax>/<x,x>.  */
    y[1] = (1-mu)*x[1] + alpha*x[2];      /* Es gilt |lambda-mu|<= f. */
    for (i=2; i<=n-1; i++) y[i]=(i-mu)*x[i]+alpha*(x[i-1]+x[i+1]);
    y[n] = alpha*x[n-1] + (n-mu)*x[n] ;
    f = 0;                                /* Anmerkung: der Speicher- */
    for (i=1; i<=n; i++) f += y[i]*y[i];  /* platz fuer den Hilfs-    */
    return sqrt(f) ;                      /* vektor y kann eingespart */
  }                                       /* werden.                  */

--------  Ergebnisse:  ------------------------------------------

990.99999999999977 2.3e-13    |       998.00002445375401 7.7e-04
------------------------------|       998.00002445718917 2.0e-13
991.99999999999966 3.4e-13    | ---------------------------------
------------------------------|       999.00206896517375 6.8e-02
993.00000000000000 1.9e-16    |       999.00209659912741 1.0e-07
------------------------------|       999.00209659912753 1.2e-13
993.99999999999977 2.4e-13    | ---------------------------------
------------------------------|      1000.04475206308587 1.7e+00
995.00000000000125 3.4e-11    |      1000.06547744941031 1.4e-03
995.00000000000080 2.8e-13    |      1000.06547880334278 1.4e-09
------------------------------|      1000.06547880334256 2.0e-14
996.00000000047692 1.5e-08    | ---------------------------------
996.00000000047646 6.1e-14    |
------------------------------|
997.00000013986323 4.4e-06    |
997.00000013986335 2.0e-13    |

time:   0.27u 0.06s 0:00.33 100.0%
```