Numerische Methoden der Physik

8 Eigenwertprobleme

Marc Wagner

Institut für theoretische Physik
Johann Wolfgang Goethe-Universität Frankfurt am Main

SS 2014

8.4 Beispiel für physikalische Anwendung: Kleine Schwingungen

- Eigenwertproblem

$$\hat{K}\vec{v}_j = \hat{\omega}_j^2\vec{v}_j \quad , \quad \hat{K} = \begin{pmatrix} +1 & -1 & 0 & 0 & \dots \\ -1 & +2 & -1 & 0 & \dots \\ 0 & -1 & +2 & -1 & \dots \\ 0 & 0 & -1 & +2 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix}$$

für eine 1D Federkette mit 10 Massenpunkten ($\hat{K}$: dimensionslose Kraftmatrix).

```
1. // **********
2.
3.
4.
5. // Jacobi.C
6.
7. // Berechnet alle Eigenwerte und Eigenvektoren einer reellen symmetrischen
8. // Matrix.
9.
```

```
10.
11.
12. // **********
13.
14.
15.
16. #include <math.h>
17. #include <stdio.h>
18. #include <stdlib.h>
19.
20.
21.
22. // **********
23.
24.
25.
26. // m -- m -- m -- m -- ... -- m
27.
28. const int N = 10;
29.
30. // Symmetrische Matrix deren Eigenwerte und Eigenvektoren berechnet werden;
31. // wird ueberschrieben; am Ende stehen die Eigenwerte auf der Diagonalen.
32. double A[N][N];
33.
34. // "Eigenvektormatrix" (Produkt von Jacobi-Rotationen), deren Spalten am Ende
35. // die Eigenvektoren von A sind.
36. double V[N][N];
```

```
37.
38. const double epsilon = 0.0000000001 * 0.0000000001;
39.
40.
41.
42. // **********
43.
44.
45.
46. int main(int argc, char **argv)
47. {
48.   FILE *file1;
49.   int i1, i2, i3;
50.   char string1[10000];
51.
52.
53.   // *****
54.
55.
56.   // Initialisiere die Massen/Kraft-Matrix.
57.
58.   for(i1 = 0; i1 < N; i1++)
59.     {
60.       for(i2 = 0; i2 < N; i2++)
61.         A[i1][i2] = 0.0;
62.     }
63.
```

```
64.   for(i1 = 0; i1 < N-1; i1++)
65.     {
66.       A[i1  ][i1  ] += 1.0;
67.       A[i1  ][i1+1] -= 1.0;
68.       A[i1+1][i1  ] -= 1.0;
69.       A[i1+1][i1+1] += 1.0;
70.     }
71.
72.   // /*
73.   for(i1 = 0; i1 < N; i1++)
74.     {
75.       for(i2 = 0; i2 < N; i2++)
76.         fprintf(stderr, "%+4.2lf   ", A[i1][i2]);
77.
78.       fprintf(stderr, "\n");
79.     }
80.   // */
81.
82.
83.   // Initialisiere die "Eigenvektormatrix".
84.
85.   for(i1 = 0; i1 < N; i1++)
86.     {
87.       for(i2 = 0; i2 < N; i2++)
88.         {
89.           if(i1 == i2)
90.             V[i1][i2] = 1.0;
```

```
91.           else
92.             V[i1][i2] = 0.0;
93.         }
94.     }
95.
96.
97.   // *****
98.
99.
100.  // Jacobi-Methode (Laufzeitverhalten kann verbessert werden, wenn nur eines
101.  // der beiden Dreiecke modifiziert wird [Symmetrie ausnutzen]).
102.
103.  int ctr = 0;
104.
105.  while(1)
106.    {
107.      // Summe der Quadrate der off-diagonal Elemente.
108.
109.      double S = 0.0;
110.
111.      for(i1 = 0; i1 < N; i1++)
112.        {
113.          for(i2 = 0; i2 < i1; i2++)
114.            S += pow(A[i1][i2], 2.0);
115.        }
116.
117.      S *= 2.0;
```

```
118.
119.       fprintf(stderr, "  S = %.5e.\n", S);
120.
121.       if(S <= epsilon)
122.         break;
123.
124.       // *****
125.
126.       ctr++;
127.
128.       fprintf(stderr, "sweep %4d ...\n", ctr);
129.
130.       // Naechster Sweep ...
131.
132.       for(i1 = 0; i1 < N; i1++)
133.         {
134.           for(i2 = 0; i2 < i1; i2++)
135.             {
136.               // !!!!!!!!!!
137.               // !!!!!!!!!!
138.               // !!!!!!!!!!
139.               // Hier sollte noch getestet werden, ob A[i1][i2] sehr klein,
140.               // ob theta sehr gross, etc. ist.
141.               // !!!!!!!!!!
142.               // !!!!!!!!!!
143.               // !!!!!!!!!!
144.
```

```
145.             // theta
146.
147.             double theta = 0.5 * (A[i2][i2] - A[i1][i1]) / A[i1][i2];
148.
149.             // t
150.
151.             double t = 1.0 / (fabs(theta) + sqrt(pow(theta, 2.0) + 1.0));
152.
153.             if(theta < 0.0)
154.               t = -t;
155.
156.             // c, s
157.
158.             double c = 1.0 / sqrt(pow(t, 2.0) + 1.0);
159.             double s = t * c;
160.
161.             // tau
162.
163.             double tau = s / (1.0 + c);
164.
165.             // Jacobi-Rotation
166.
167.             // Matrix A.
168.
169.             double A_pp = A[i1][i1] - t * A[i1][i2];
170.             double A_qq = A[i2][i2] + t * A[i1][i2];
171.
```

```
172.               double A_rp[N], A_rq[N];
173.
174.               for(i3 = 0; i3 < N; i3++)
175.                 {
176.                   if(i3 != i1 && i3 != i2)
177.                     {
178.                       A_rp[i3] = A[i3][i1] - s * (A[i3][i2] + tau * A[i3][i1]);
179.                       A_rq[i3] = A[i3][i2] + s * (A[i3][i1] - tau * A[i3][i2]);
180.                     }
181.                 }
182.
183.               A[i1][i2] = 0.0;
184.               A[i2][i1] = 0.0;
185.
186.               A[i1][i1] = A_pp;
187.               A[i2][i2] = A_qq;
188.
189.               for(i3 = 0; i3 < N; i3++)
190.                 {
191.                   if(i3 != i1 && i3 != i2)
192.                     {
193.                       A[i3][i1] = A_rp[i3];
194.                       A[i1][i3] = A_rp[i3];
195.
196.                       A[i3][i2] = A_rq[i3];
197.                       A[i2][i3] = A_rq[i3];
198.                     }
```

```
199.                 }
200.
201.               // "Eigenvektormatrix" V.
202.
203.               double V_rp[N], V_rq[N];
204.
205.               for(i3 = 0; i3 < N; i3++)
206.                 {
207.                   V_rp[i3] = V[i3][i1] - s * (V[i3][i2] + tau * V[i3][i1]);
208.                   V_rq[i3] = V[i3][i2] + s * (V[i3][i1] - tau * V[i3][i2]);
209.                 }
210.
211.               for(i3 = 0; i3 < N; i3++)
212.                 {
213.                   V[i3][i1] = V_rp[i3];
214.                   V[i3][i2] = V_rq[i3];
215.                 }
216.             }
217.         }
218.
219.       // /*
220.       for(i1 = 0; i1 < N; i1++)
221.         {
222.           for(i2 = 0; i2 < N; i2++)
223.             fprintf(stderr, "%+4.2lf   ", A[i1][i2]);
224.
225.           fprintf(stderr, "\n");
```

```
226.         }
227.       // */
228.     }
229.
230.
231.   // *****
232.
233.
234.   for(i1 = 0; i1 < N; i1++)
235.     {
236.       fprintf(stderr, "\n\\lambda_%02d = %+10.6lf.\n", i1, A[i1][i1]);
237.
238.       fprintf(stderr, "v_%02d = ( ");
239.
240.       for(i2 = 0; i2 < N; i2++)
241.         {
242.           fprintf(stderr, "%+5.2lf", V[i2][i1]);
243.
244.           if(i2 < N-1)
245.             fprintf(stderr, " , ");
246.           else
247.             fprintf(stderr, " ).\n");
248.         }
249.
250.       // Teste Normierung:
251.
252.       double norm = 0.0;
```

```
253.
254.       for(i2 = 0; i2 < N; i2++)
255.         norm += pow(V[i2][i1], 2.0);
256.
257.       if(fabs(norm - 1.0) > 0.0000000001)
258.         {
259.           fprintf(stderr, "Error: Eigenvektor nicht korrekt normiert.\n");
260.           exit(EXIT_FAILURE);
261.         }
262.     }
263.
264.
265.   // *****
266.
267.
268.   // Bahnkurven fuer ABs r(t=0) = (1,0,0,...).
269.
270.   const double t_max = 20.0;
271.   const int num_samples = 2001;
272.
273.   for(i1 = 0; i1 < N; i1++)
274.     {
275.       sprintf(string1, "r%02d.dat", i1);
276.
277.       if((file1 = fopen(string1, "w")) == NULL)
278.         {
279.           fprintf(stderr, "Error!\n");
```

```
280.           exit(EXIT_FAILURE);
281.         }
282.
283.       for(i2 = 0; i2 <= num_samples; i2++)
284.         {
285.           double t = ((double)i2) / ((double)(num_samples-1)) * t_max;
286.
287.           double r = 0.0;
288.
289.           for(i3 = 0; i3 < N; i3++)
290.             r += V[0][i3] * V[i1][i3] * cos(sqrt(A[i3][i3]) * t);
291.
292.           fprintf(file1, "%+.5e %+.5e\n", t, r);
293.         }
294.
295.       fclose(file1);
296.     }
297.
298.
299.   // *****
300.
301.
302.   return EXIT_SUCCESS;
303. }
304.
305.
306.
```

```
307. // **********
```

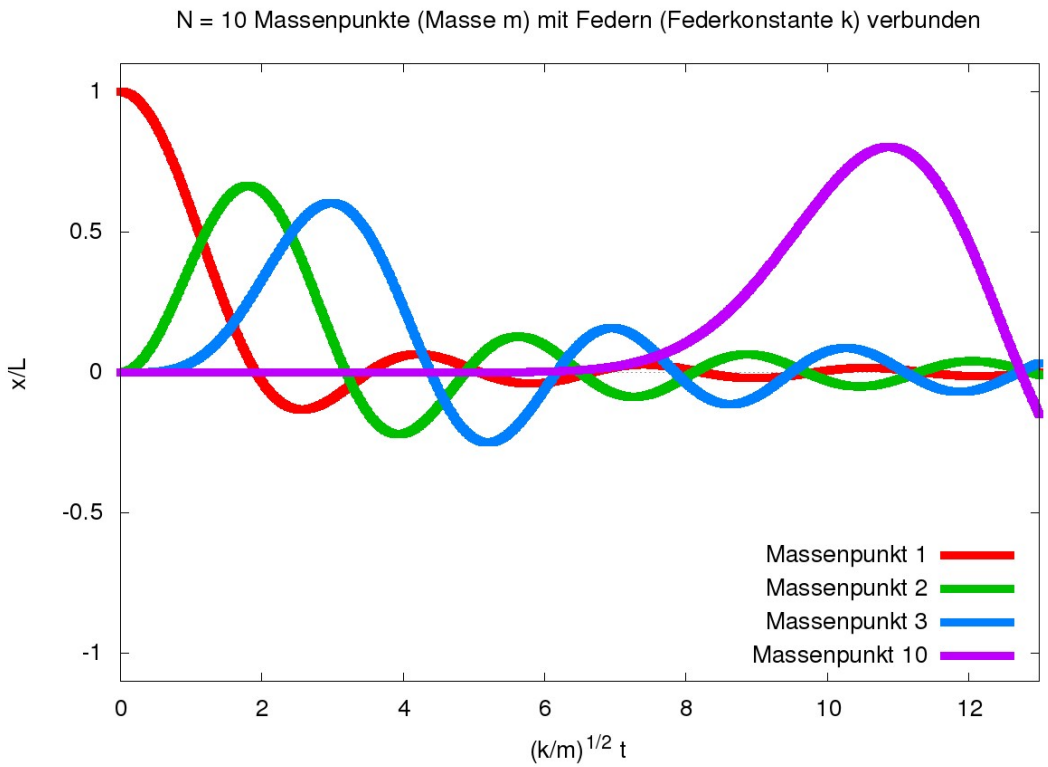
```
+1.00   -1.00   +0.00   +0.00   +0.00   +0.00   +0.00   +0.00   +0.00   +0.00
-1.00   +2.00   -1.00   +0.00   +0.00   +0.00   +0.00   +0.00   +0.00   +0.00
+0.00   -1.00   +2.00   -1.00   +0.00   +0.00   +0.00   +0.00   +0.00   +0.00
+0.00   +0.00   -1.00   +2.00   -1.00   +0.00   +0.00   +0.00   +0.00   +0.00
+0.00   +0.00   +0.00   -1.00   +2.00   -1.00   +0.00   +0.00   +0.00   +0.00
+0.00   +0.00   +0.00   +0.00   -1.00   +2.00   -1.00   +0.00   +0.00   +0.00
+0.00   +0.00   +0.00   +0.00   +0.00   -1.00   +2.00   -1.00   +0.00   +0.00
+0.00   +0.00   +0.00   +0.00   +0.00   +0.00   -1.00   +2.00   -1.00   +0.00
+0.00   +0.00   +0.00   +0.00   +0.00   +0.00   +0.00   -1.00   +2.00   -1.00
+0.00   +0.00   +0.00   +0.00   +0.00   +0.00   +0.00   +0.00   -1.00   +1.00
  S = 1.80000e+01.
sweep    1 ...
+0.16   -0.11   -0.22   +0.09   +0.11   -0.05   -0.04   +0.03   +0.01   +0.02
-0.11   +3.63   -0.11   +0.33   +0.09   -0.13   -0.06   +0.04   +0.04   +0.02
-0.22   -0.11   +0.48   -0.08   -0.36   +0.14   +0.15   +0.02   -0.12   +0.01
+0.09   +0.33   -0.08   +3.40   -0.08   +0.40   +0.07   -0.14   -0.03   -0.07
+0.11   +0.09   -0.36   -0.08   +0.63   +0.01   -0.38   +0.31   -0.14   +0.23
-0.05   -0.13   +0.14   +0.40   +0.01   +3.23   -0.05   +0.34   +0.20   +0.03
-0.04   -0.06   +0.15   +0.07   -0.38   -0.05   +0.72   +0.17   -0.50   -0.27
+0.03   +0.04   +0.02   -0.14   +0.31   +0.34   +0.17   +2.86   -0.09   -0.04
+0.01   +0.04   -0.12   -0.03   -0.14   +0.20   -0.50   -0.09   +1.89   +0.00
+0.02   +0.02   +0.01   -0.07   +0.23   +0.03   -0.27   -0.04   +0.00   +0.99
  S = 2.91374e+00.
sweep    2 ...
+0.03   -0.04   -0.05   +0.01   -0.00   +0.01   +0.01   -0.02   +0.01   -0.01
-0.04   +3.89   -0.03   +0.03   +0.02   +0.05   +0.01   -0.04   +0.02   -0.02
-0.05   -0.03   +0.13   -0.03   -0.06   -0.04   -0.10   +0.01   -0.01   +0.08
+0.01   +0.03   -0.03   +2.58   -0.01   +0.05   +0.26   -0.01   -0.05   -0.09
-0.00   +0.02   -0.06   -0.01   +1.39   -0.08   -0.01   +0.04   +0.05   +0.00
+0.01   +0.05   -0.04   +0.05   -0.08   +3.62   +0.02   +0.00   -0.00   -0.01
+0.01   +0.01   -0.10   +0.26   -0.01   +0.02   +0.37   +0.01   +0.03   -0.00
-0.02   -0.04   +0.01   -0.01   +0.04   +0.00   +0.01   +3.18   +0.00   -0.00
+0.01   +0.02   -0.01   -0.05   +0.05   -0.00   +0.03   +0.00   +2.00   +0.00
-0.01   -0.02   +0.08   -0.09   +0.00   -0.01   -0.00   -0.00   +0.00   +0.82
  S = 2.53839e-01.
sweep    3 ...
+0.00   -0.01   -0.01   +0.02   -0.00   +0.00   +0.00   -0.00   +0.00   -0.00
-0.01   +3.90   -0.01   +0.01   -0.01   +0.00   +0.00   -0.00   -0.00   -0.00
-0.01   -0.01   +0.10   +0.10   -0.00   +0.01   -0.01   -0.00   +0.01   +0.01
+0.02   +0.01   +0.10   +2.61   +0.00   +0.00   -0.00   -0.00   +0.00   +0.00
-0.00   -0.01   -0.00   +0.00   +1.38   +0.00   -0.00   -0.00   -0.00   +0.00
```

```
+0.00   +0.00   +0.01   +0.00   +0.00   +3.62   -0.00   -0.00   +0.00   +0.00
+0.00   +0.00   -0.01   -0.00   -0.00   -0.00   +0.38   -0.00   -0.00   -0.00
-0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   +3.18   -0.00   -0.00
+0.00   -0.00   +0.01   +0.00   -0.00   +0.00   -0.00   -0.00   +2.00   +0.00
-0.00   -0.00   +0.01   +0.00   +0.00   +0.00   -0.00   -0.00   +0.00   +0.82
  S = 2.12206e-02.
sweep    4 ...
+0.00   -0.00   -0.00   -0.00   -0.00   -0.00   +0.00   -0.00   -0.00   -0.00
-0.00   +3.90   +0.00   +0.00   -0.00   +0.00   -0.00   -0.00   +0.00   +0.00
-0.00   +0.00   +0.10   -0.00   -0.00   -0.00   +0.00   +0.00   -0.00   +0.00
-0.00   +0.00   -0.00   +2.62   -0.00   +0.00   -0.00   -0.00   +0.00   +0.00
-0.00   -0.00   -0.00   -0.00   +1.38   +0.00   -0.00   -0.00   +0.00   +0.00
-0.00   +0.00   -0.00   +0.00   +0.00   +3.62   -0.00   -0.00   +0.00   +0.00
+0.00   -0.00   +0.00   -0.00   -0.00   -0.00   +0.38   -0.00   +0.00   -0.00
-0.00   -0.00   +0.00   -0.00   -0.00   -0.00   -0.00   +3.18   -0.00   +0.00
-0.00   +0.00   -0.00   +0.00   +0.00   +0.00   +0.00   -0.00   +2.00   +0.00
-0.00   +0.00   +0.00   +0.00   +0.00   +0.00   -0.00   +0.00   +0.00   +0.82
  S = 7.26279e-06.
sweep    5 ...
+0.00   +0.00   -0.00   -0.00   +0.00   -0.00   +0.00   +0.00   -0.00   -0.00
+0.00   +3.90   -0.00   +0.00   +0.00   +0.00   -0.00   -0.00   +0.00   -0.00
-0.00   -0.00   +0.10   +0.00   +0.00   +0.00   +0.00   -0.00   -0.00   -0.00
-0.00   +0.00   +0.00   +2.62   -0.00   +0.00   -0.00   -0.00   +0.00   +0.00
+0.00   +0.00   +0.00   -0.00   +1.38   +0.00   -0.00   +0.00   +0.00   +0.00
-0.00   +0.00   +0.00   +0.00   +0.00   +3.62   +0.00   -0.00   -0.00   -0.00
+0.00   -0.00   +0.00   -0.00   -0.00   +0.00   +0.38   +0.00   +0.00   -0.00
+0.00   -0.00   -0.00   -0.00   +0.00   -0.00   +0.00   +3.18   +0.00   -0.00
-0.00   +0.00   -0.00   +0.00   +0.00   -0.00   +0.00   +0.00   +2.00   +0.00
-0.00   -0.00   -0.00   +0.00   +0.00   -0.00   -0.00   -0.00   +0.00   +0.82
  S = 2.26242e-10.
sweep    6 ...
-0.00   -0.00   +0.00   +0.00   +0.00   -0.00   -0.00   -0.00   +0.00   -0.00
-0.00   +3.90   +0.00   +0.00   +0.00   -0.00   -0.00   +0.00   -0.00   +0.00
+0.00   +0.00   +0.10   +0.00   -0.00   -0.00   +0.00   +0.00   -0.00   -0.00
+0.00   +0.00   +0.00   +2.62   -0.00   -0.00   +0.00   -0.00   +0.00   -0.00
+0.00   +0.00   -0.00   -0.00   +1.38   -0.00   -0.00   +0.00   -0.00   +0.00
-0.00   -0.00   -0.00   -0.00   -0.00   +3.62   -0.00   -0.00   +0.00   +0.00
-0.00   -0.00   +0.00   +0.00   -0.00   -0.00   +0.38   -0.00   +0.00   -0.00
-0.00   +0.00   +0.00   -0.00   +0.00   -0.00   -0.00   +3.18   -0.00   +0.00
+0.00   -0.00   -0.00   +0.00   -0.00   +0.00   +0.00   -0.00   +2.00   +0.00
-0.00   +0.00   -0.00   -0.00   +0.00   +0.00   -0.00   +0.00   +0.00   +0.82
  S = 1.12777e-32.

\lambda_00 =  -0.000000.
```

```
v_44362432 = ( +0.32 , +0.32 , +0.32 , +0.32 , +0.32 , +0.32 , +0.32 , +0.32 , +0.32 , +0.32 ).

\lambda_01 =  +3.902113.
v_44362432 = ( -0.07 , +0.20 , -0.32 , +0.40 , -0.44 , +0.44 , -0.40 , +0.32 , -0.20 , +0.07 ).

\lambda_02 =  +0.097887.
v_44362432 = ( -0.44 , -0.40 , -0.32 , -0.20 , -0.07 , +0.07 , +0.20 , +0.32 , +0.40 , +0.44 ).

\lambda_03 =  +2.618034.
v_44362432 = ( +0.26 , -0.43 , +0.00 , +0.43 , -0.26 , -0.26 , +0.43 , +0.00 , -0.43 , +0.26 ).

\lambda_04 =  +1.381966.
v_44362432 = ( +0.36 , -0.14 , -0.45 , -0.14 , +0.36 , +0.36 , -0.14 , -0.45 , -0.14 , +0.36 ).

\lambda_05 =  +3.618034.
v_44362432 = ( +0.14 , -0.36 , +0.45 , -0.36 , +0.14 , +0.14 , -0.36 , +0.45 , -0.36 , +0.14 ).

\lambda_06 =  +0.381966.
v_44362432 = ( +0.43 , +0.26 , +0.00 , -0.26 , -0.43 , -0.43 , -0.26 , +0.00 , +0.26 , +0.43 ).

\lambda_07 =  +3.175571.
v_44362432 = ( -0.20 , +0.44 , -0.32 , -0.07 , +0.40 , -0.40 , +0.07 , +0.32 , -0.44 , +0.20 ).

\lambda_08 =  +2.000000.
v_44362432 = ( +0.32 , -0.32 , -0.32 , +0.32 , +0.32 , -0.32 , -0.32 , +0.32 , +0.32 , -0.32 ).

\lambda_09 =  +0.824429.
v_44362432 = ( -0.40 , -0.07 , +0.32 , +0.44 , +0.20 , -0.20 , -0.44 , -0.32 , +0.07 , +0.40 ).
```

- Trajektorien der Massenpunkte (Anfangsbedingungen $\hat{\vec{x}}(\hat{t} = 0) = (1, 0, 0, \ldots, 0)$, $\dot{\hat{\vec{x}}}(\hat{t} = 0) = (0, 0, \ldots, 0)$):

$$\hat{\vec{x}}(\hat{t}) \quad = \quad \sum_j v_{j,0} \vec{v}_j \cos(\hat{\omega}_j \hat{t}).$$


N = 10 Massenpunkte (Masse m) mit Federn (Federkonstante k) verbunden
x/L
1
0.5
0
-0.5
-1
0
2
4
6
8
10
12
(k/m)^1/2 t
Massenpunkt 1
Massenpunkt 2
Massenpunkt 3
Massenpunkt 10