Numerische Methoden der Physik

6 Lineare Gleichungssysteme

Marc Wagner

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

SS 2014

6.3 Gauß-Elimination mit Rückwärtssubstitution

- Vergleich verschiedener Pivotstrategien:

```
1. // **********
2.
3.
4.
5. // Gauss_backsubs.C
6.
7. // Loest das lineare Gleichungssystem A x = b mit Hilfe des Gauss-Algorithmus
8. // mit Rueckwaertssubstitution.
9.
10.
11.
12. // **********
13.
14.
15.
16. #include <math.h>
17. #include <stdio.h>
18. #include <stdlib.h>
19. #include <time.h>
20.
21.
```

```
22.
23. // **********
24.
25.
26.
27. // #define __TEILPIVOTISIERUNG__
28.
29. // #define __SKALIERTE_TEILPIVOTISIERUNG__
30.
31.
32.
33. // **********
34.
35.
36.
37. // Dimension von A, b und x.
38. const int N = 4;
39. // const int N = 100;
40.
41. // Matrix A (wird im Verlauf der Rechnung ueberschrieben).
42. double A[N][N];
43.
44. // Rechte-Seite-Vektor (wird im Verlauf der Rechnung ueberschrieben).
45. double b[N];
46.
47. // Ergebnis-Vektor.
48. double x[N];
```

```
49.
50. // Permutation der Zeilen aufgrund von Pivotisierung.
51. int p[N];
52.
53.
54.
55. // **********
56.
57.
58.
59. // Erzeugt eine gleichmaessig verteilte Zufallszahl zwischen min und max.
60.
61. double DRand(double min, double max)
62. {
63.   return min + (max-min) * ( (rand() + 0.5) / (RAND_MAX + 1.0) );
64. }
65.
66.
67.
68. // **********
69.
70.
71.
72. void Print()
73. {
74.   int i1, i2;
75.
```

```
  for(i1 = 0; i1 < N; i1++)
    {
      for(i2 = 0; i2 < N; i2++)
        {
          fprintf(stdout, "%+5.2lf  ", A[p[i1]][i2]);
        }

      fprintf(stdout, "|  %+5.2lf\n", b[p[i1]]);
    }

  fprintf(stdout, "\n");
}



// **********



int main(int argc, char **argv)
{
  double d1, d2, d3;
  int i1, i2, i3;

  srand(0);
  // srand((unsigned int)time(NULL));

```

```
103.
104.   // *****
105.
106.
107.   // Generiere zufaellige Matrix A und zufaelligen Vektor b, Eintraege in
108.   //   [-1.0 , +1.0].
109.
110.   for(i1 = 0; i1 < N; i1++)
111.     {
112.       for(i2 = 0; i2 < N; i2++)
113.         {
114.           A[i1][i2] = DRand(-1.0, +1.0);
115.         }
116.
117.       b[i1] = DRand(-1.0, +1.0);
118.     }
119.
120.   // Intitialisiere Zeilenpermutation.
121.
122.   for(i1 = 0; i1 < N; i1++)
123.     p[i1] = i1;
124.
125.   Print();
126.
127.
128.   // *****
129.
```

```
130.
131.   // Sichere Matrix A und Vektor b fuer spaeteren Test.
132.
133.   double A_org[N][N];
134.   double b_org[N];
135.
136.   for(i1 = 0; i1 < N; i1++)
137.     {
138.       for(i2 = 0; i2 < N; i2++)
139.         {
140.           A_org[i1][i2] = A[i1][i2];
141.         }
142.
143.       b_org[i1] = b[i1];
144.     }
145.
146.
147.   // *****
148.
149.
150.   // Spaltenausraeumen.
151.
152.   // Vorlesungsnotation:
153.   // i1 ~ n-1
154.   // i2 ~ j-1
155.   // i3 ~ k-1
156.
```

```
157. #ifdef __SKALIERTE_TEILPIVOTISIERUNG__
158.
159.   // Maximum fuer jede Zeile von A speichern, bevor A durch Gauss-Algorithmus
160.   // ueberschrieben wird.
161.
162.   double A_ij_max[N];
163.
164.   for(i1 = 0; i1 < N; i1++)
165.     {
166.       A_ij_max[i1] = fabs(A[i1][0]);
167.
168.       for(i2 = 1; i2 < N; i2++)
169.         {
170.           if(fabs(A[i1][i2]) > A_ij_max[i1])
171.             A_ij_max[i1] = fabs(A[i1][i2]);
172.         }
173.     }
174.
175. #endif
176.
177.   for(i1 = 0; i1 < N-1; i1++)
178.     // N-1 Spaltenausraeumschritte.
179.     {
180.       // Finde beste Zeile gemaess Pivotstrategie.
181.
182.       int index = i1;
183.
```

```
184. #ifdef __TEILPIVOTISIERUNG__
185.
186.       for(i2 = i1+1; i2 < N; i2++)
187.         {
188.           if(fabs(A[p[i2]][i1]) > fabs(A[p[i1]][i1]))
189.             index = i2;
190.         }
191.
192. #endif
193.
194. #ifdef __SKALIERTE_TEILPIVOTISIERUNG__
195.
196.       d1 = fabs(A[p[i1]][i1]) / A_ij_max[p[i1]];
197.
198.       for(i2 = i1+1; i2 < N; i2++)
199.         {
200.           d2 = fabs(A[p[i2]][i1]) / A_ij_max[p[i2]];
201.
202.           if(d2 > d1)
203.             index = i2;
204.         }
205.
206. #endif
207.
208.       i2 = p[i1];
209.       p[i1] = p[index];
210.       p[index] = i2;
```

```
211.
212.       // ***
213.
214.       for(i2 = i1+1; i2 < N; i2++)
215.         // Fuer alle verbleibenden Zeilen ...
216.         {
217.           d1 = A[p[i2]][i1] / A[p[i1]][i1];
218.
219.           // Zu Beginn jeder verbleibenden Zeile wird 0.0 "erzeugt".
220.           A[p[i2]][i1] = 0.0;
221.
222.           for(i3 = i1+1; i3 < N; i3++)
223.             {
224.               A[p[i2]][i3] -= d1 * A[p[i1]][i3];
225.             }
226.
227.           b[p[i2]] -= d1 * b[p[i1]];
228.         }
229.
230.       Print();
231.     }
232.
233.
234.   // *****
235.
236.
237.   // Rueckwaertssubstitution.
```

```
238.
239.   for(i1 = N-1; i1 >= 0; i1--)
240.     // Fuer alle Komponenten von x ...
241.     {
242.       x[i1] = b[p[i1]];
243.
244.       for(i2 = i1+1; i2 < N; i2++)
245.         {
246.           x[i1] -= A[p[i1]][i2] * x[i2];
247.         }
248.
249.       x[i1] /= A[p[i1]][i1];
250.     }
251.
252.   fprintf(stdout, "x = ( ");
253.
254.   for(i1 = 0; i1 < N-1; i1++)
255.     {
256.       fprintf(stdout, "%+5.2lf  ", x[i1]);
257.     }
258.
259.   fprintf(stdout, "%+5.2lf ).\n\n", x[N-1]);
260.
261.
262.   // *****
263.
264.
```

```
265.   // Test.
266.
267.   double b_check[N];
268.
269.   for(i1 = 0; i1 < N; i1++)
270.     {
271.       b_check[i1] = 0.0;
272.
273.       for(i2 = 0; i2 < N; i2++)
274.         {
275.           b_check[i1] += A_org[i1][i2] * x[i2];
276.         }
277.     }
278.
279.   fprintf(stdout, "b_check = ( ");
280.
281.   for(i1 = 0; i1 < N-1; i1++)
282.     {
283.       fprintf(stdout, "%+5.2lf  ", b_check[i1]);
284.     }
285.
286.   fprintf(stdout, "%+5.2lf ).\n\n", b_check[N-1]);
287.
288.   fprintf(stdout, "b_check - b = ( ");
289.
290.   // Abweichung fuer jede Komponente.
291.
```

```
292.   for(i1 = 0; i1 < N-1; i1++)
293.     {
294.       fprintf(stdout, "%+.1e  ", b_check[i1] - b_org[i1]);
295.     }
296.
297.   fprintf(stdout, "%+.1e ).\n\n", b_check[N-1] - b_org[N-1]);
298.
299.   // Norm der Abweichung.
300.
301.   double norm = 0.0;
302.
303.   for(i1 = 0; i1 < N; i1++)
304.     {
305.       norm += pow(b_check[i1] - b_org[i1], 2.0);
306.     }
307.
308.   norm = sqrt(norm);
309.
310.   fprintf(stdout, "|b_check - b| = %+.5e.\n", norm);
311.
312.
313.   // *****
314.
315.
316.   return EXIT_SUCCESS;
317. }
318.
```

```
319.
320.
321. // **********
```

```
N = 4
-----



Keine Pivotisierung:

+0.68  -0.21  +0.57  +0.60  |  +0.82
-0.60  -0.33  +0.54  -0.44  |  +0.11
-0.05  +0.26  -0.27  +0.03  |  +0.90
+0.83  +0.27  +0.43  -0.72  |  +0.21

+0.68  -0.21  +0.57  +0.60  |  +0.82
+0.00  -0.52  +1.04  +0.09  |  +0.84
+0.00  +0.24  -0.23  +0.07  |  +0.96
+0.00  +0.53  -0.26  -1.45  |  -0.79

+0.68  -0.21  +0.57  +0.60  |  +0.82
+0.00  -0.52  +1.04  +0.09  |  +0.84
+0.00  +0.00  +0.26  +0.11  |  +1.35
+0.00  +0.00  +0.81  -1.36  |  +0.07

+0.68  -0.21  +0.57  +0.60  |  +0.82
+0.00  -0.52  +1.04  +0.09  |  +0.84
+0.00  +0.00  +0.26  +0.11  |  +1.35
+0.00  +0.00  +0.00  -1.69  |  -4.19

x = ( -2.22  +7.31  +4.24  +2.47 ).

b_check = ( +0.82  +0.11  +0.90  +0.21 ).

b_check - b = ( -2.2e-16  +1.5e-16  -1.1e-16  -1.7e-15 ).

|b_check - b| = +1.74535e-15.



Teilpivotisierung:
```

```
+0.68  -0.21  +0.57  +0.60  |  +0.82
-0.60  -0.33  +0.54  -0.44  |  +0.11
-0.05  +0.26  -0.27  +0.03  |  +0.90
+0.83  +0.27  +0.43  -0.72  |  +0.21

+0.83  +0.27  +0.43  -0.72  |  +0.21
+0.00  -0.13  +0.85  -0.97  |  +0.26
+0.00  +0.27  -0.25  -0.01  |  +0.92
+0.00  -0.43  +0.21  +1.18  |  +0.65

+0.83  +0.27  +0.43  -0.72  |  +0.21
+0.00  -0.43  +0.21  +1.18  |  +0.65
+0.00  +0.00  -0.11  +0.73  |  +1.32
+0.00  +0.00  +0.79  -1.33  |  +0.07

+0.83  +0.27  +0.43  -0.72  |  +0.21
+0.00  -0.43  +0.21  +1.18  |  +0.65
+0.00  +0.00  +0.79  -1.33  |  +0.07
+0.00  +0.00  +0.00  +0.54  |  +1.33

x = ( -2.22  +7.31  +4.24  +2.47 ).

b_check = ( +0.82  +0.11  +0.90  +0.21 ).

b_check - b = ( -1.1e-16  -3.6e-16  -3.3e-16  +1.1e-16 ).

|b_check - b| = +5.15537e-16.



Skalierte Teilpivotisierung:

+0.68  -0.21  +0.57  +0.60  |  +0.82
-0.60  -0.33  +0.54  -0.44  |  +0.11
-0.05  +0.26  -0.27  +0.03  |  +0.90
+0.83  +0.27  +0.43  -0.72  |  +0.21

+0.68  -0.21  +0.57  +0.60  |  +0.82
+0.00  -0.52  +1.04  +0.09  |  +0.84
+0.00  +0.24  -0.23  +0.07  |  +0.96
+0.00  +0.53  -0.26  -1.45  |  -0.79

+0.68  -0.21  +0.57  +0.60  |  +0.82
```

```
+0.00  +0.24  -0.23  +0.07  |  +0.96
+0.00  +0.00  +0.55  +0.23  |  +2.88
+0.00  +0.00  +0.25  -1.59  |  -2.88

+0.68  -0.21  +0.57  +0.60  |  +0.82
+0.00  +0.24  -0.23  +0.07  |  +0.96
+0.00  +0.00  +0.55  +0.23  |  +2.88
+0.00  +0.00  +0.00  -1.69  |  -4.19

x = ( -2.22  +7.31  +4.24  +2.47 ).

b_check = ( +0.82  +0.11  +0.90  +0.21 ).

b_check - b = ( -2.2e-16  -6.9e-17  +1.1e-16  -1.5e-15 ).

|b_check - b| = +1.52081e-15.



N = 100
-------



Keine Pivotisierung:

|b_check - b| = +2.25693e-11.


Teilpivotisierung:

|b_check - b| = +1.46047e-12.


Skalierte Teilpivotisierung:

|b_check - b| = +3.28886e-13.
```