Programmierpraktikum

Claudius Gros, SS2012

Institut für theoretische Physik
Goethe-University Frankfurt a.M.

Elimination

linear equations

$$\left.\begin{array}{rrrrcr} 2x & -3y & -z & +2w & = & 4 \\ 4x & -4y & -z & +4w & = & 4 \\ 2x & -5y & -3z & +3w & = & 9 \end{array}\right\} \qquad A\,\vec{x} = \vec{b} \qquad \vec{x} = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

- 3×4 matrix A
- augmented matrix $(A|\vec{b})$

$$\left(\begin{array}{cccc|c} 2 & -3 & -1 & 2 & 4 \\ 4 & -4 & -1 & 4 & 4 \\ 2 & -5 & -3 & 3 & 9 \end{array}\right)$$

- solution via iterative elimination

Gauss elimination

$$\left(\begin{array}{cccc|c} 2 & -3 & -1 & 2 & 4 \\ 4 & -4 & -1 & 4 & 4 \\ 2 & -5 & -3 & 3 & 9 \end{array}\right)$$

- solve for x in first equation and substitute into the other equations
- substract (multiples of) first row

$$\left(\begin{array}{cccc|c} 2 & -3 & -1 & 2 & 4 \\ 0 & 2 & 1 & 0 & 5 \\ 0 & -2 & -2 & 1 & 5 \end{array}\right)$$

- solve for 7 in second equation and substitute into third and forth equation
- substract (multiples of) second row

$$\left(\begin{array}{cccc|c} 2 & -3 & -1 & 2 & 4 \\ 0 & 2 & 1 & 0 & -4 \\ 0 & 0 & -1 & 1 & 1 \end{array}\right)$$

- in matrix form

$$A\vec{x} = b \qquad \begin{pmatrix} 2 & -3 & -1 & 2 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} 4 \\ -4 \\ 1 \end{pmatrix}$$

back substitution

$$\begin{pmatrix} 2 & -3 & -1 & 2 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} 4 \\ -4 \\ 1 \end{pmatrix}$$

- last equation

$$w - z = 1$$

- take z as free parameter and substitute back

$$\begin{array}{rrcl} & w & = & z + 1 \\ 2y + z = -4 & y & = & -\frac{z}{2} - 2 \\ 2x - 3y - z + 2w = 4 & x & = & -\frac{5z}{4} - 2 \end{array}$$

with

$$2x = 4 - \frac{3}{2}\,z - 6 + z - 2z - 2 = -4 - \frac{5}{2}\,z$$

- final (one-dimensional) solution

$$\vec{x} = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} -5z/4 - 2 \\ -z/2 - 2 \\ z \\ z + 1 \end{pmatrix} \qquad \text{for any } \; z$$

pivoting

- Gauss elimination

$$\begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix} \Rightarrow \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix} \Rightarrow \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{pmatrix} \Rightarrow \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & 0 & * \end{pmatrix}$$

- **pivoting**
  select largest element $[*]$ in row

$$\begin{pmatrix} * & [*] & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix} \Rightarrow \begin{pmatrix} * & * & * & * \\ * & 0 & * & [*] \\ * & 0 & * & * \\ * & 0 & * & * \end{pmatrix} \Rightarrow \begin{pmatrix} * & * & * & * \\ * & 0 & * & * \\ * & 0 & [*] & 0 \\ * & 0 & * & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} * & * & * & * \\ * & 0 & * & * \\ * & 0 & * & 0 \\ * & 0 & 0 & 0 \end{pmatrix}$$

- one needs to divide by the pivot selected,
  pivot need to be non-vanishing
- numerical stable if pivot is large

- **partial pivoting**
  only largest element in given row/column
- **complete pivoting**
  largest element in matrix

Gauss elimination with pivoting

- with swaping of pivot rows

```
 1. public class GaussElimination {
 2.
 3.   private static final double smallNumber = 1e-10;  // a small number
 4.
 5. /** Gauss elimination of Ax=b with pivoting.
 6.   * On input quadratic matrix A and vector b.
 7.   * On ouput solution x.
 8.   */
 9.   public static double[] gaussElimination(double[][] A, double[] b)
10.     {
11. //--- missing: testing that A is a NxN matrix
12.     int N  = b.length;
13.
14. //--- loop over all columns A[][p]
15.     for (int p = 0; p < N; p++)
16.       {
17. //--- find pivot row and swap rows
18.       int max = p;
19.       for (int i = p + 1; i < N; i++)
20.         if (Math.abs(A[i][p]) > Math.abs(A[max][p])) max = i;
```

```
21.
22. //--- swap row A[p][] with A[max][]
23.       double[] temp = A[p];
24.       A[p] = A[max];
25.       A[max] = temp;
26.
27. //--- swap elements b[p] with b[max]
28.       double tt = b[p];
29.       b[p] = b[max];
30.       b[max] = tt;
31.
32. //--- throw a warning if matrix is singular or nearly singular
33.       if (Math.abs(A[p][p]) <= smallNumber) {
34.         throw new RuntimeException("Matrix is singular or nearly singular");
35.         }
36.
37. //--- Gauss with pivot within A and b
38.       for (int i = p + 1; i < N; i++) {
39.         double alpha = A[i][p] / A[p][p];  // divide by pivot
40.         b[i] -= alpha * b[p];
41.         for (int j = p; j < N; j++)
42.           A[i][j] -= alpha * A[p][j];
43.         }
44.       }  // end of loop over all columns
45.
46. //--- back substitution
```

```
47.     double[] x = new double[N];
48.     for (int i = N - 1; i >= 0; i--) {
49.       double sum = 0.0;
50.       for (int j = i + 1; j < N; j++)
51.          sum += A[i][j] * x[j];
52.       x[i] = (b[i] - sum) / A[i][i];   // pivots on diagonal after swapping
53.       }
54.     return x;
55.     }  // end of GaussElimination.gaussElimination
56.
57.
58.   public static void main(String[] args) {
59. //--- testing
60.     double[][] A = { { 0, 1,  1 },
61.                      { 2, 4, -2 },
62.                      { 0, 3, 15 }
63.                    };
64.      double[] b = { 4, 2, 36 };
65.      double[] x = gaussElimination(A, b);
66.
67.      for (int i = 0; i < b.length; i++)
68.         System.out.println(x[i]);
69.
70.     }  // end of GaussElimination.main()
71. }      // end of GaussElimination
```

- add formatted output
- add testing

matrices

- diagonal

$$\begin{pmatrix} * & 0 & 0 & 0 & 0 \\ 0 & * & 0 & 0 & 0 \\ 0 & 0 & * & 0 & 0 \\ 0 & 0 & 0 & * & 0 \\ 0 & 0 & 0 & 0 & * \end{pmatrix}$$

- tri-diagonal

$$\begin{pmatrix} * & * & 0 & 0 & 0 \\ * & * & * & 0 & 0 \\ 0 & * & * & * & 0 \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \end{pmatrix}$$

- block-diagonal (blocks are independent

$$\begin{pmatrix} * & * & 0 & 0 & 0 \\ * & * & 0 & 0 & 0 \\ 0 & 0 & * & * & 0 \\ 0 & 0 & * & * & 0 \\ 0 & 0 & 0 & 0 & * \end{pmatrix}$$

- L (lower left) and R (upper right) triangular matrices

$$L = \begin{pmatrix} * & 0 & 0 & 0 & 0 \\ * & * & 0 & 0 & 0 \\ * & * & * & 0 & 0 \\ * & * & * & * & 0 \\ * & * & * & * & * \end{pmatrix} \qquad R = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * \end{pmatrix}$$

- sparse (few non-zero entries, there are special algorithms)

$$\begin{pmatrix} 0 & * & 0 & 0 & 0 \\ * & 0 & 0 & * & 0 \\ 0 & * & 0 & 0 & 0 \\ * & 0 & 0 & 0 & * \\ 0 & 0 & * & 0 & 0 \end{pmatrix}$$

LR decomposition

- decompose matrix into product of L and R matrices

$$A = L\,R \qquad\qquad A\,\vec{x} = \vec{b}$$

- always possible (Gauss, Hausholder, ...)

$$L\,R\,\vec{x} = \vec{b} \qquad\qquad L\,\vec{y} = \vec{b} \qquad\qquad \vec{y} = R\,\vec{x}$$

- forward substitution: solve for $\vec{y}$

$$L\,\vec{y} = \vec{b} \qquad\qquad L = \begin{pmatrix} * & 0 & 0 & 0 \\ * & * & 0 & 0 \\ * & * & * & 0 \\ * & * & * & * \end{pmatrix}$$

- backward substitution: solve for $\vec{x}$

$$R\vec{x} = \vec{y} \qquad\qquad R = \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & 0 & * \end{pmatrix}$$