Prof. Dr. Stefan Funken
M.Sc. Andreas Bantle
Dipl.-Math. oec. Klaus Stolle

Universität Ulm
Institut für Numerische Mathematik
Wintersemester 2014/2015

# Numerische Lineare Algebra - Matlab-Blatt 2
## Lösung

(Besprechung in den MATLAB-Tutorien in KW 45/46)

**Hinweise**
Die Hinweise zur Abgabe der Übungsblätter finden Sie auf dem ersten Übungsblatt!

**Aufgabe 4** *(Matrix-Produkt)* (15 Punkte)

Der Strassen-Algorithmus ist ein Algorithmus zur Berechnung der Matrix-Matrix-Multiplikation von zwei quadratischen Matrizen $A, B \in \mathbb{R}^{n\times n}$, wobei $n = 2^k$ mit $k \in \mathbb{N}$ ist. Sei also $C = A \cdot B$.
Wir betrachten im Folgenden die Matrizen $A, B$ und $C$ als Blockmatrizen

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \quad \text{und} \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

wobei $A_{ij}, B_{ij}, C_{ij} \in \mathbb{R}^{n/2\times n/2}$ gilt. Formal ergibt sich dann:

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \qquad C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$
$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \qquad C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}.$$

Anstatt nun die 8 Matrix-Matrix-Multiplikationen (der halben Dimension) auszurechnen definieren wir weiter die Matrizen

$$M_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \qquad M_2 = (A_{21} + A_{22}) \cdot B_{11}$$
$$M_3 = A_{11} \cdot (B_{12} - B_{22}) \qquad M_4 = A_{22} \cdot (B_{21} - B_{11})$$
$$M_5 = (A_{11} + A_{12}) \cdot B_{22} \qquad M_6 = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$$
$$M_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$

und erhalten

$$C_{11} = M_1 + M_4 - M_5 + M_7 \qquad C_{12} = M_3 + M_5$$
$$C_{21} = M_2 + M_4 \qquad C_{22} = M_1 - M_2 + M_3 + M_6.$$

Wir müssen also nur 7 Matrix-Produkte der halben Größe berechnen, die wir rekursiv wieder mit dem Strassen-Algorithmus berechnen können. Auf der untersten Rekursionsebene werden dann nur noch Skalare multipliziert und die Rekursion bricht ab.
Asymptotisch erhalten wir folgenden Aufwand:

$$\text{FLOP's(Strassen)} \approx \mathcal{O}(n^{2.8}).$$

Bearbeiten Sie die folgenden Aufgaben:

(i) Schreiben Sie eine MATLAB-Funktion `C = matProd(A,B)`, in der die Matrizen $A, B \in \mathbb{R}^{n\times n}$ multipliziert werden. Verwenden Sie bei der Implementierung drei geschachtelte `for`-Schleifen.

(ii) Schreiben Sie eine MATLAB-Funktion `C = matProdVec(A,B)`, in der die Matrizen $A, B \in \mathbb{R}^{n\times n}$ multipliziert werden. Vektorisieren Sie bei der Implementierung das Skalarprodukt einer Zeile von $A$ mit einer Spalte von $B$ (innerste `for`-Schleife aus Teilaufgabe (i) fällt weg.)

(iii) Schreiben Sie eine MATLAB-Funktion `C = matProdStr(A,B)`, die die Matrizen $A, B \in \mathbb{R}^{n\times n}$ mit dem Strassen-Algorithmus rekursiv multipliziert.

(iv) Laden Sie sich das MATLAB-Skript `mainAufgabe4.m` von der Vorlesungshomepage
`http://www.uni-ulm.de/mawi/mawi-numerik/lehre/ws1415/numla1.html`
herunter und vervollständigen Sie die fehlenden Zeilen:

- Zeile 29-31: Zufalls-Matrizen $A, B \in \mathbb{R}^{n_k \times n_k}$ anlegen (siehe `rand`).
- Zeile 35-37: Matrix-Produkt $A \cdot B$ mit 3-facher `for`-Schleife berechnen.
- Zeile 42-44: Matrix-Produkt $A \cdot B$ mit vektorisierter Funktion berechnen.
- Zeile 49-51: Matrix-Produkt $A \cdot B$ mit dem Strassen-Algorithmus berechnen.
- Zeile 57-59: Die Laufzeit über die Matrix-Dimension $n_k$ in doppelt-logarithmischer Skala plotten, Legende einzeichnen und Achsen beschriften (siehe `loglog`).

(v) Analysieren Sie die Laufzeiten. Erklären Sie, was Sie in der doppelt-logarithmischen Darstellung sehen und achten Sie insbesondere auf das asymptotische Verhalten.

(vi) Ist das Strassen-Verfahren in der Praxis zu empfehlen?

**Lösung:**

(i) Function `matProd.m`

```
function C = matProd(A,B)

n = size(A,1);
C = zeros(n);
for j=1:n
  for k=1:n
    for ell=1:n
      C(j,k) = C(j,k)+A(j,ell)*B(ell,k);
    end
  end
end
```

(ii) Funktion `matProdVec.m`

```
function C = matProdVec(A,B)
n = size(A,1);
C = zeros(n);
for j=1:n
  for k=1:n
      C(j,k) = A(j,:)*B(:,k);
  end
end
```

(iii) Funktion `matProdStr.m`

```
function C = matProdStr(A,B)
if size(A,1)==1
  C = A*B;
else
  %*** Indizes berechnen
  n=size(A,1);
  ind1 = 1:n/2;  ind2 = n/2+1:n;

  %*** Hilfsmatrizen aufstellen
  M1 = matProdStr(A(ind1,ind1)+A(ind2,ind2), B(ind1,ind1)+B(ind2,ind2));
  M2 = matProdStr(A(ind2,ind1)+A(ind2,ind2), B(ind1,ind1)              );
  M3 = matProdStr(A(ind1,ind1)             , B(ind1,ind2)-B(ind2,ind2));
  M4 = matProdStr(A(ind2,ind2)             , B(ind2,ind1)-B(ind1,ind1));
  M5 = matProdStr(A(ind1,ind1)+A(ind1,ind2), B(ind2,ind2));
  M6 = matProdStr(A(ind2,ind1)-A(ind1,ind1), B(ind1,ind1)+B(ind1,ind2));
  M7 = matProdStr(A(ind1,ind2)-A(ind2,ind2), B(ind2,ind1)+B(ind2,ind2));
  %*** Matrix C berechnen
  C = [M1+M4-M5+M7, M3+M5;...
       M2+M4, M1-M2+M3+M6];
end
```

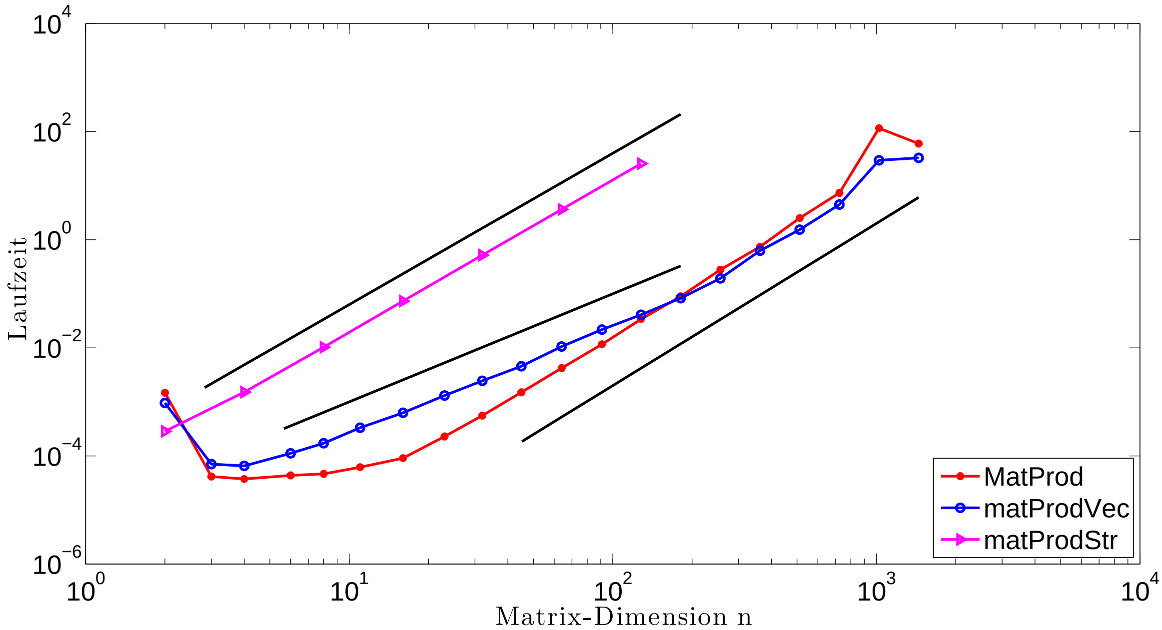
(iv) Skript `mainAufgabe4.m`

```
clc, clear all, close all
%*** Functionen auf Richtigkeit testen
A = rand(2^4);
B = rand(2^4);
C = A*B;
C1 = matProd(A,B);
C2 = matProdVec(A,B);
C3 = matProdStr(A,B);
if max(max(abs(C-C1)))<1e-14
  disp('matProd......  ok!')
end
if max(max(abs(C-C2)))<1e-14
  disp('matProdVec...  ok!')
end
if max(max(abs(C-C3)))<1e-12
  disp('matProdStr...  ok!')
end

%*** Laufzeit-Test
n = round(2.^(1:0.5:11));
timeMPStr = zeros(size(n));

for k=1:length(n)
  %*** Zufalls-Matrizen anlegen
  A = rand(n(k));
  B = rand(n(k));
  %*** Matrix-Produkt mit 3-facher for-schleife
  tic
  C1 = matProd(A,B);
  timeMP(k) = toc;
  %*** Matrix-Produkt vektorisiert
  tic
  C2 = matProdVec(A,B);
  timeMPVec(k) = toc;
  %*** Matrix-Produkt mit Strassen-Algorithmus
  if k<=13 && mod(k,2)==1
    tic
    C3 = matProdStr(A,B);
    timeMPStr(k) = toc;
  end
  figure(1)
  loglog(n(1:k),timeMP(1:k),'-r*',n(1:k),timeMPVec(1:k),'-bo',...
         n(1:2:k),timeMPStr(1:2:k),'-m>')
  legend('MatProd','matProdVec','matProdStr','location','SouthEast')
  xlabel('Matrix-Dimension n')
  ylabel('Laufzeit')
  title('Laufzeiten f r die Matrix-Matrix-Mulitplikation')

  clear A B C1 C2 C3
end
%*** Verhalten einzeichnen
hold on
loglog(n(5:end), 2e-6*n(5:end).^2.8, '-k')
loglog(n(3:end), 6e-9*n(3:end).^3  , '-k')
loglog(n(3:8)  , 8e-6*n(3:8).^2    , '-k')
```




(v) Laufzeitanalyse:

- Jeweils asymptotisch eine Gerade ($\Rightarrow$ polynomialer Aufwand!)
- Asymptotisches Verhalten:
  - Straßen-Algorithmus $\mathcal{O}(n^{2.8})$
  - `MatProd`-Funktion: $\mathcal{O}(n^3)$
  - `MatProdVec`-Funktion: bis $n \approx 800$ $\mathcal{O}(n^2)$, danach $\mathcal{O}(n^3)$
- Präasymptotischer Bereich bis $n \approx 20$ bei `matProd` und $n \approx 800$ bei `matProdVec`.

Erklärung:

- Straßen-Algorithmus verhält sich sofort wie $O(n^{2.8})$
- Die `matProdVec`-Funktion hat zunächst nur quadratisches Verhalten:
  Bei Vektorisierung in MATLAB werden intern optimierte Routinen (BLAS-Routinen) aufgerufen, die im Verhalten die Konstante vor $n^3$ reduzieren, sodass für kleine $n$ der Term $n^2$ im Verhalten dominiert. Passt die Matrix nicht mehr in den Cache-Speicher (ab $n \approx 800$), dominieren die teuren Speicherzugriffe und wir erhalten das erwartete $O(n^3)$-Verhalten.
- Da der Straßen-Algorithmus ein rekursiver Algorithmus ist, werden keine optimierten BLAS-Routinen aufgerufen. D.h. der Algorithmus ist wesentlich langsamer, hat aber von der Steigung das beste Verhalten.

(vi) Der Straßen-Algorithmus ist so nicht zu empfehlen. Er hat zwar asymptotisch ein besseres Verhalten, aber bei unserer Implementierung eine zu großen Konstante! Eine Variante, um den Straßen-Algorithmus für die Praxis tauglicher zu machen, wäre die Rekursion früher abzubrechen und die normale Matrix-Matrix-Mulitplikation zu verwenden.

**Aufgabe 5** *(LR-Zerlegung)* (5 Punkte)

Bearbeiten Sie die folgenden Aufgaben:

(i) Schreiben Sie eine MATLAB-Funktion `[L,R]=computeLR(A)`, die für eine gegebene quadratische Matrix die $LR$-Zerlegung berechnet und die Matrizen $L$ und $R$ zurückgibt. Vektorisieren Sie ihrer Code so weit wie möglich. Existiert zu einer Matrix keine $LR$-Zerlegung, soll das Programm eine Fehlermeldung ausgeben und abbrechen.

(ii) Schreiben Sie ein Skript `aufgabe5.m`, in dem Sie Ihre Funktion aus Teilaufgabe (i) mit einer $(10\times 10)$-Zufallsmatrix und der Matrix

$$A = \begin{pmatrix} 1 & 0 & 3 \\ 2 & 0 & 4 \\ 3 & 2 & 5 \end{pmatrix}$$

testen.

**Lösung:**

(i) Die Funktion `computeLR.m`:

```matlab
function [L,R]= computeLR(A)

%*** Matrix-Dimension bestimmen
n = size(A,1);
%*** Matrix L initialisieren
L=eye(n);

%*** restliche Eintraege berechnen
for k=1:n-1
  %*** Abbrechen falls keine Zerlegung existiert
  if abs(A(k,k))<eps
    R=A;
    disp('LR-Zerlegung existiert nicht!')
    return
  end

  for j = k+1 : n
    %*** compute entries l_{j,k} and store in A
    L(j,k) = A(j,k)/A(k,k);
    %*** update other entries of j-th row
    A(j,k+1 : n) = A(j,k+1 : n) - L(j,k)*A(k,k+1 : n);
  end
end


%*** Matrix R extrahieren
R=triu(A);
```

(ii) Das Skript `mainAufgabe5.m`:

```matlab
clc, clear all, close all

%*** Teste Funktionen auf Richtigkeit
A1 = rand(10);
[L1,R1] = computeLR(A1);

if max(max(abs(L1*R1-A1)))<1e-14
  disp('computeLR...... ok!')
end

A2 = [1,0,3;2 0 4;3,2,5];
[L2,R2] = computeLR(A2);
```