WiMa-Praktikum 1

Universität Ulm, Sommersemester 2018

Woche 8

Lernziele

In diesem Praktikum sollen Sie üben und lernen:

- Besonderheiten der `For`-Schleife in MATLAB
- Wiederholung des Umgangs mit Matrizen und Vektoren

Am Anfang geben wir Ihnen einen kurzen Überblick über die benötigten MATLAB-Anweisungen.

Matlab - Unterschiede zu anderen Programmiersprachen

Mit diesem achten Praktikumsblatt werden wir den MATLAB-Teil des WiMa-1-Praktikums abschliessen. Wir möchten zwei wichtige Punkte wiederholen, in denen sich MATLAB von anderen Programmiersprachen unterscheidet.

`For`-Schleife in Matlab

Wie in anderen Programmiersprachen dient auch in MATLAB die `for`-Schleife dazu, eine Gruppe von Anweisungen zu wiederholen. Die Anzahl der Wiederholungen steht dabei bereits vor dem ersten Schleifendurchlauf fest. Die Syntax der `for`-Schleife lautet:

```
for index = werte
    befehle
end
```

`werte` kann von der Form `anfangswert:endwert` oder `anfangswert:inkrement:endwert` sein. Wie in anderen Programmiersprachen auch, nimmt dann die Variable `index` nacheinander die Werte zwischen `anfangswert` und `endwert` an, gegebenenfalls mit einem Abstand, der durch `inkrement` bestimmt ist.
Im Unterschied zu vielen anderen gängigen Programmiersprachen kann in MATLAB `werte` auch eine Matrix sein. Dann nimmt `index` nacheinander als Werte die Spalten der Matrix an:

```
≫ A = [1 2 3; 4 5 6; 7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9

≫ for i = A
     disp('aktuelle Variable i:')
     disp(i)
end
aktuelle Variable i:
     1
     4
     7

aktuelle Variable i:
     2
     5
     8

aktuelle Variable i:
     3
     6
     9
```

Die Matrix A kann dabei natürlich auch ein Spaltenvektor sein. Dann wird die Schleife eben nur einmal durchlaufen:

```
≫ A = [1; 2; 3]

A =

     1
     2
     3

≫ for i = A
     disp('aktuelle Variable i:')
     disp(i)
end
aktuelle Variable i:
     1
     2
     3
```

Oder auch ein Zeilenvektor. Dann nimmt index nacheinander alle Werte des Zeilenvektors an:

```
≫ str = 'Text'

str =

Text

≫ for i = str
     disp('aktuelle Variable i:')
     disp(i)
end
aktuelle Variable i
T
aktuelle Variable i
e
aktuelle Variable i
x
aktuelle Variable i
t
```

Beachte: Die Matrixelemente können jeden beliebigen Datentyp einschließlich Strukturen und Cell Arrays haben. Im obigen Beispiel ist str ein String, also ein Zeilenvektor mit Zeichen. Im folgenden Beispiel ist v ein Cell Array der Dimension 1×4, also ein Zeilenvektor mit 4 Zellen, die jeweils vom Typ String sind. Die Schleife wird also vier Mal durchlaufen:

```
≫ clear
≫ v{1} = 'Dies';
≫ v{2} = 'ist';
≫ v{3} = 'ein';
≫ v{4} = 'Cell Array aus Strings';
≫ v

v =

    'Dies'    'ist'    'ein'    'Cell Array aus S...'

≫ whos
  Name      Size            Bytes  Class     Attributes

  v         1x4               512  cell

≫ for i = v
     disp('aktuelle Variable i:')
     disp(i)
end
aktuelle Variable i:
    'Dies'

aktuelle Variable i:
    'ist'

aktuelle Variable i:
    'ein'

aktuelle Variable i:
    'Cell Array aus Strings'
```

Operationen auf Matrizen

Im letzten Praktikumsblatt haben wir gesehen, dass in MATLAB alle Variablen (mit Ausnahme von function handles) Matrizen sind.
Im Unterschied zu vielen anderen Programmiersprachen sind in MATLAB viele Operatoren nicht nur für Skalare, sondern auch für Matrizen definiert. Stellvertretend wollen wir hier die Multiplikation betrachten.

Matrix-Matrix-Produkt, Matrix-Vektor-Produkt, Skalarprodukt, Dyadisches Produkt

In MATLAB ist das Produkt zweier Matrizen für diejenigen Matrizen definiert, für die es auch im mathematischen Sinne definiert ist. Beispielsweise ergibt die Multiplikation einer 3×2-Matrix mit einer 2×4-Matrix eine 3×4-Matrix:

```
≫ A = [1 2; 3 4; 5 6]
A =
     1     2
     3     4
     5     6
≫ B = ones(2,4)
B =
     1     1     1     1
     1     1     1     1
≫ A * B
ans =
     3     3     3     3
     7     7     7     7
    11    11    11    11
```

Für einen Spaltenvektor `v` sind sowohl das Skalarprodukt `v'*v` als auch das dyadische Produkt `v*v'` definiert, nicht aber die Produkte `v*v` und `v'*v'`.

```
≫ v = [1; 2; 5]
v =
     1
     2
     5
≫ v'*v
ans =
    30
≫ v*v'
ans =
     1     2     5
     2     4    10
     5    10    25
≫ v*v
Error using  *
Inner matrix dimensions must agree.
≫ v'*v'
Error using  *
Inner matrix dimensions must agree.
```

Außerdem erlaubt der Multiplikationsoperator `*` auch die Multiplikation eines Vektors oder einer Matrix mit einem Skalar:

```
≫ 3*A
ans =
     3     6
     9    12
    15    18
```

Elementweises Produkt

Der MATLAB-Operator `.*` multipliziert zwei Matrizen elementweise: Jedes Element der ersten Matrix wird mit dem entsprechenden Element der zweiten Matrix multipliziert. Die beiden Matrizen müssen die gleiche Dimension haben, die Ergebnismatrix hat dann auch diese Dimension. Im folgenden Beispiel werden die Elemente der ersten Zeile der Matrix `A` mit 2 und die der zweiten Zeile mit 4 multipliziert. Das Matrix-Matrix-Produkt ist nicht definiert.

```
≫ clear
≫ A = [1 2 3; 4 5 6]
A =
     1     2     3
     4     5     6
≫ B = [2 2 2; 4 4 4]
B =
     2     2     2
     4     4     4
≫ A.*B
ans =
     2     4     6
    16    20    24
≫ A*B
Error using  *
Inner matrix dimensions must agree.
```

Beachte: Für quadratische Matrizen ist sowohl das Matrix-Matrix-Produkt als auch das elementweise Produkt definiert. Eine falsche Verwendung ergibt also keine MATLAB-Fehlermeldung, die Ergebnisse der beiden Operationen sind aber nicht gleich.

```
≫ clear
≫ A = [1 2; 3 4]
A =
     1     2
     3     4
≫ E = eye(2)
E =
     1     0
     0     1
≫ A*E
ans =
     1     2
     3     4
≫ A.*E
ans =
     1     0
     0     4
```

Beachte: Auch bei den elementweisen Operationen kann einer der beiden Operanden ein skalarer Wert sein.

Vektorisierung

MATLAB greift zur Berechnung der auf Matrizen definierten Operationen auf Funktionen zurück, die sehr hardwarenah implementiert sind. Diese Funktionen berücksichtigen beispielsweise die Größe der im Prozessor vorhandenen Caches (z. B. L2-Cache) sowie die Zahl und Größe der einzelnen Register. Die Operationen auf Matrizen werden blockweise durchgeführt. Die Blöcke sind dabei so groß, dass sie gerade in die vorhandenen Caches passen und die Daten möglichst selten aus dem Hauptspeicher nachgeladen werden müssen. Denn bei heutigen Rechnerarchitekturen ist der Zeitaufwand zum Laden der Daten in der Regel höher als der Zeitaufwand für die eigentlichen Rechenoperationen.
Daher sind die MATLAB-Operationen auf Matrizen in der Regel viel schneller, als wenn sie mit `for`-Schleifen selbst programmiert werden.
Beim Programmieren in MATLAB sollte man also darauf achten, Schleifen möglichst zu vermeiden und stattdessen die von MATLAB zur Verfügung gestellten Operatoren für Matrizen zu verwenden.

Praktikumsaufgaben

Lesen Sie die Aufgabenstellung, studieren Sie dann die vorgegebenen Programmzeilen. Ersetzen Sie dann die %% Kommentare im vorgegebenen Code durch Matlab-Anweisungen und führen Sie das Programm aus.

---

1. Die MATLAB-Funktion `primes(n)` berechnet die Primzahlen kleiner oder gleich `n`. Schreiben Sie eine MATLAB-Funktion, die die Primzahlen kleiner oder gleich `n` ausgibt. Dabei soll jede Primzahl p in einer eigenen Zeile mit dem Text `p ist eine Primzahl` ausgegeben werden. Geben Sie mit Ihrer Funktion die Primzahlen kleiner oder gleich 20 aus.

2. In dieser Praktikumsaufgabe sollen Sie vier MATLAB-Funktionen zur Berechnung eines Produkts zweier Matrizen implementieren. Diese sollen das Produkt jeweils mit einer unterschiedlichen Anzahl von `for`-Schleifen berechnen. Testen Sie Ihre Funktionen jeweils an geeigneten Matrizen.

   (a) Schreiben Sie eine MATLAB-Funktion, die das Produkt zweier Matrizen mit Hilfe von drei geschachtelten `for`-Schleifen berechnet.

   (b) Schreiben Sie eine MATLAB-Funktion, die das Produkt zweier Matrizen mit Hilfe von zwei geschachtelten `for`-Schleifen berechnet. Ersetzen Sie die innerste `for`-Schleife des ersten Aufgabenteils durch ein Skalarprodukt.

   (c) Schreiben Sie eine MATLAB-Funktion, die das Produkt zweier Matrizen mit Hilfe von einer `for`-Schleife berechnet. Ersetzen Sie die innerste `for`-Schleife des zweiten Aufgabenteils durch ein Vektor-Matrix-Produkt oder ein Matrix-Vektor-Produkt.

   (d) Schreiben Sie eine MATLAB-Funktion, die das Produkt zweier Matrizen ohne `for`-Schleife direkt mit dem von MATLAB zur Verfügung gestellten Matrix-Matrix-Produkt berechnet.

   (e) Mit den MATLAB-Befehlen `tic` und `toc` können Sie die Zeit zwischen `tic` (startet die Stoppuhr) und `toc` (stoppt die Stoppuhr) messen. Messen Sie für alle vier Varianten die benötigte Zeit. Achten Sie dabei darauf, dass immer gleiche Startbedingungen vorliegen. Rufen Sie also beispielsweise `clear C` vor jedem Funktionsaufruf auf. Starten Sie die Uhr innerhalb der Funktion unmittelbar vor Beginn der Berechnung und stoppen Sie die Uhr sofort nach Ende der Berechnung wieder. Abhängig von Ihrer Hardware bietet sich beispielsweise eine Matrixgröße von 1000×1000 zur Messung an.

   Ihre Antwort:

3. Schreiben Sie eine MATLAB-Funktion, die eine $m \times n$-Matrix A so initialisiert, dass jedes Matrixelement $a_{i,j},\ i = 1, \ldots, m,\ j = 1, \ldots, n$ den Wert $a_{i,j} = (i-1) * 10 + j$ hat, und diese Matrix A als Rückgabewert zurück liefert. `m` und `n` sollen ihrer Funktion beim Aufruf als Parameter übergeben werden.

   Vermeiden Sie die Verwendung von Schleifen.

   Hinweis: Denken Sie an das dyadische Produkt.

4. Schreiben Sie eine MATLAB-Funktion, die einen Vektor mit allen Quadratzahlen k^2 für $k = m, \ldots, n$ berechnet und diesen Vektor als Rückgabewert zurück liefert. `m` und `n` sollen ihrer Funktion beim Aufruf als Parameter übergeben werden. Vermeiden Sie die Verwendung von Schleifen.

5. Schreiben Sie eine MATLAB-Funktion, die einen Vektor mit allen Zweierpotenzen 2^k für $k = m, \ldots, n$ berechnet und diesen Vektor als Rückgabewert zurück liefert. `m` und `n` sollen ihrer Funktion beim Aufruf als Parameter übergeben werden. Vermeiden Sie die Verwendung von Schleifen.