Einführung in die Programmierung (MA8003)

Theorie 2.2: Schleifen, Vektorisierung, bedingte Ausführung, Debug

Dr. Laura Scarabosio

Technische Universität München
Fakultät Mathematik, Lehrstuhl für Numerische Mathematik M2

09.10.2019

Theorie 2.2: Inhalt

1. Schleifen
   - for-Schleifen
   - Beispiel: Jacobi-Verfahren
   - While Schleifen
2. Bedingte Ausführung
   - if, elseif, else
   - Test der Anzahl von Ein- und Ausgangsgrößen
3. Debugging

Inhalt

Schleifen I

Eine Schleife wiederholt eine Folge von Anweisungen bis eine Abbruchsbedingung erfüllt ist.

In Matlab gibt es zwei unterschiedliche Arten von Schleifen:

- `for` Schleife
- `while` Schleife

Schleifen können beliebig geschachtelt werden.

Schleifen (`for` Schleife)

Syntax

```
for i=v
  <auszufuehrende Anweisungen>
end
```

Mit `v` Vektor (oder Matrix), häufig von der Form `<start>:<ende>`.

Bei einer `for` Schleife werden die Anweisungen so oft ausgeführt wie es Elemente in `v` gibt. Ist z.B. `length(v) == 5` wird der Körper der Schleife 5 Mal ausgeführt. `i` durchläuft dabei jeweils die Elemente von `v`.

Achtung: `for` Schleifen können in vielen Fällen durch vektorwertige Ausdrücke ersetzt werden

Schleifen (for Schleife)

```
>> x=[1,2,10,-1];
>> k=0;
>> for l=x
k=k+2*l
end

k =
     2

k =
     6

k =
    26

k =
    24

>> sum(2*x)

ans =
    24
```

```
>> for k=1:5
for l=k:5
A(k,l)=k/l;
A(l,k)=k/l;
end
end
>> A

A =
    1.0000    0.5000    0.3333    0.2500    0.2000
    0.5000    1.0000    0.6667    0.5000    0.4000
    0.3333    0.6667    1.0000    0.7500    0.6000
    0.2500    0.5000    0.7500    1.0000    0.8000
    0.2000    0.4000    0.6000    0.8000    1.0000
```

Beispiel: Jacobi-Verfahren I

Zum Lösen eines linearen Gleichungssystems $Ax = b$, mit $A \in \mathbb{R}^{n \times n}$ bieten sich in manchen Fällen iterative Verfahren an.

$$\begin{aligned} a_{11} \cdot x_1 + \cdots + a_{1n} \cdot x_n &= b_1 \\ a_{21} \cdot x_1 + \cdots + a_{2n} \cdot x_n &= b_2 \\ &\vdots \\ a_{n1} \cdot x_1 + \cdots + a_{nn} \cdot x_n &= b_n \end{aligned}$$

Ein einfaches Beispiel für diese Klasse ist das Jacobi-Verfahren.

Jacobi-Verfahren

Ausgehend von einem beliegigen Startvektor $x_i^{(0)}$ löse für alle i die i-te Gleichung nach der i-ten Variablen x_i auf und iteriere dies.

Hinreichende Bedingung für Konvergenz: A strikt diagonaldominant.

$$\begin{aligned} x_i^{(k+1)} &:= \frac{1}{a_{ii}} \Big(b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^{(k)} \Big), \quad i = 1, \ldots, n. \\ &= \frac{1}{a_{ii}} \Big(b_i + a_{ii} x_i^{(k)} - \sum_{j=1}^{n} a_{ij} x_j^{(k)} \Big) = x_i^{(k)} + \frac{1}{a_{ii}} \left(b_i - a_i^T x^{(k)} \right) \end{aligned}$$

Beispiel: Jacobi-Verfahren I

```
function x = jacobi1(A, b, x0, iter)
%JACOBI Verfahren Version 1
x = x0;
xn = 0*x;
n = length(A);
for k = 1:iter
  for i = 1:n
    xn(i) = x(i) + (b(i) - A(i,:)*x)./A(i,i);
  end
  x = xn;
end
```

```
>> A = gallery('poisson',10);
>> b=(1:100)';
>> x = jacobi(A,b,zeros(100,1),500);
>> norm(A*x - b)

ans =
   4.6606e-07

>> tic; x = jacobi1(A,b,zeros(100,1),500); toc
Elapsed time is 0.640984 seconds.
```

Jacobi-Verfahren II

Das Jacobi-Verfahren kann man weiter umschreiben:

$$x_i^{(k+1)} := x_i^{(k)} + \frac{1}{a_{ii}} \left(b_i - a_i^T x^{(k)} \right), \quad i = 1, \ldots, n$$

$$\Rightarrow \quad x^{(k+1)} := x^{(k)} + D^{-1} \left(b - A x^{(k)} \right), \quad \text{mit } D = \text{Diag}(A)$$

```
function x = jacobi2(A, b, x0, iter)
%JACOBI Verfahren Version 2
x = x0;
D = diag(A); %diag(A) gibt Vektor der Diagonalelemente zurück
for k = 1:iter
  x = x + (b - A*x)./D;
end
```

```
>> tic; x = jacobi2(A,b,zeros(100,1),500); toc
Elapsed time is 0.017832 seconds.
>> norm(A*x - b)
ans =
   4.6606e-07
```

Eliminieren der inneren Schleife beschleunigt Routine um Faktor ≈ 36!

Schleifen II (`while` Schleifen)

`while` Schleifen werden solange ausgeführt *solange* eine Bedingung wahr ist. Zum Beispiel bei einem iterativen Verfahren solange wie eine festgelegte Genauigkeit nicht erreicht wurde.

Achtung: Gefahr von Endlosschleifen!

Syntax

```
while (<Bedingung>)
  <auszufuehrende Anweisungen>
end
```

```
>> j=1;
>> while (j>0)
epsilon = j;
j = j/2;
end
>> epsilon

epsilon =
  4.9407e-324
```

Jacobi-Verfahren III

- Beim Jacobi-Verfahren haben wir bisher die Anzahl der Iterationen vorgegeben.
- Besser abbrechen, wenn die Norm des Residuums klein ist

```
function x = jacobi3(A, b, x0, tol)
%JACOBI Verfahren Version 3
x = x0;
D = diag(A); %diag(A) gibt Vektor der Diagonalelemente zurück
r = b - A*x;
while (norm(r) >= tol)
  x = x + r./D;
  r = b - A*x;
end
```

```
>> tic; x = jacobi3(A,b,zeros(100,1),1e-6); toc
Elapsed time is 0.021556 seconds.

>> norm(A*x - b)
ans =
   9.8105e-07
```

Jacobi-Verfahren IV

Wir haben ein Problem, wenn das Verfahren nicht konvergiert:
Gefahr einer Endlosschleife!

```
function x = jacobi4(A, b, x0, tol, iter)
%JACOBI Verfahren Version 4
x = x0;
D = diag(A); %diag(A) gibt Vektor der Diagonalelemente zurück
r = b - A*x;
k = 1;
while (norm(r) >= tol & k < iter)
  x = x + r./D;
  r = b - A*x;
  k = k + 1;
end
k
```

```
>> tic; x = jacobi4(A,b,zeros(100,1),1e-6,500); toc
k =
    483
Elapsed time is 0.027932 seconds.

>> norm(A*x - b)
ans =
   9.8105e-07
```

Schleifen III

- `break`: Eine Schleife kann durch `break` sofort verlassen werden
- `continue`: Bricht den aktuellen Schleifendurchlauf ab und springt zum Kopf der Schleife.

Bei geschachtelten Schleifen wirken die Befehle nur auf die „innerste“.

```
>> j=1;
>> for i=1:2000, epsilon=j; j=j/2;
if (j == 0), break; end,
end
>> epsilon

epsilon =
  4.9407e-324

>> i

i =
        1075
```

Inhalt

Bedingte Ausführung I

Häufig möchte man Codeteile nur dann ausführen, wenn bestimmte Bedingungen erfüllt sind. Dies kann man mit `if`-Abfragen realisieren.

Syntax

```
if (<logischer Ausdruck 1>)
  <Anweisungen 1>
elseif (<logischer Ausdruck 2>)
  <Anweisungen 2>
elseif ...
else
  <Anweisungen 3>
end
```

Bedingte Ausführung II

```
function x = foo(x)
  if (x(1) > x(2))
    temp = x(1);
    x(1) = x(2);
    x(2) = temp;
  else
    disp 'Schon sortiert'
  end
```

```
>> x=[2,1];
>> x = foo(x);

x =
     1     2

>> x = foo(x)
Schon sortiert
```

Bedingte Ausführung III

if-Abfragen werden z. B. verwendet um Voraussetzungen an die Variablen am Anfang einer Funktion zu überprüfen:

```
function fun(x)
   if (~isscalar(x))
      error('x muss Skalar sein')
   elseif (isnan(x) | isinf(x))
      error('x ist keine zulässige Zahl')
   else
      disp('Juhu');
   end
```

Hilfreich: `is*` Funktionen, wie `isnan`, `iscomplex`, `isvector`...

```
>> fun([0,1])

??? Error using ==> fun at 3
x muss Skalar sein

>> fun(magic(3))

??? Error using ==> fun at 3
x muss Skalar sein
```

```
>> fun(1/0)

??? Error using ==> fun at 7
x ist keine zulässige Zahl

>> fun(log(-1))
Juhu

>> log(-1)
ans =
        0 + 3.1416i
```

Jacobi-Verfahren V

```
function x = jacobi5(A, b, x0, tol, iter)
%JACOBI Verfahren Version 5
d = size(A);
if (d(1) ~= d(2))
  error('A muss quadratische Matrix sein');
elseif (~isvector(b))
  error('b muss ein Vektor sein');
elseif (length(b) ~= d(1))
  error('Vektor b muss Länge length(A) haben');
elseif (length(x0) ~= d(1))
  error('Vektor x0 muss Länge length(A) haben');
...
end
x = x0(:);
D = diag(A);
r = b - A*x;
k = 1;
while (norm(r) >= tol)
  x = x + r./D;
  r = b - A*x;
  k = k + 1;
  if (k >= iter), break; end
end
```

Funktionen: Verschiedene Argumentanzahl

- Wir haben schon gesehen, dass sich manche Funktionen unterschiedlich verhalten, wenn die Anzahl der Eingabe- oder Ausgabeparameter unterschiedlich ist.
- Eine Funktion mit der Deklaration
  `function foo(E1, E2, E3)`
  wird auch aufgerufen, wenn weniger als 3 Parameter übergeben werden. Die übrigen Variablen sind dann nicht definiert.

```
>> x = jacobi(A,b)
??? Input argument "x1" is undefined.

Error in ==> jacobi at 10
x = x1(:);
```

- Mit `nargin` kann die Anzahl der übergebenen Argumente abgefragt werden. Analog `nargout`.

Jacobi-Verfahren VI

```
function x = jacobi6(A, b, x0, tol, iter)
%JACOBI Verfahren Version 6
d = size(A);
if (nargin < 2)
  error('Mindestens A und b müssen übergeben werden');
elseif (d(1) ~= d(2))
  error('A muss quadratische Matrix sein');
...
end
if (nargin < 5), iter = 1000; end
if (nargin < 4), tol = 1e-6; end
if (nargin < 3),
    x = zeros(length(b),1);
else
    x = x0(:);
end
D = diag(A);
r = b - A*x;
k = 1;
while (norm(r) >= tol)
  x = x + r./D;
  r = b - A*x;
  k = k + 1;
  if (k >= iter), break; end
end
```

Jacobi-Verfahren VI

```
>> x = jacobi6(A,b);
>> norm(A*x -b)
ans =

   9.8105e-07

>> x = jacobi6(A,b, 1e-10);
??? Error using ==> jacobi at 13
Vektor x muss Länge length(A) haben

>> x = jacobi6(A,b, zeros(100,1), 1e-10);

>> norm(A*x - b)
ans =
   9.7000e-11
```

Inhalt

Debugging

In komplizierten Codes ist es oft nicht einfach, Fehler zu finden. Das wird einfacher, wenn man Debug-Werkzeuge benutzt:

- Auswahl und recht Maustaste $\rightarrow$ *Evaluate Selection* (oder F9) um einen Teil eines Codes auszuführen
- Benutzung von *Breakpoints*

Ende Theorie 2.2

Fragen?