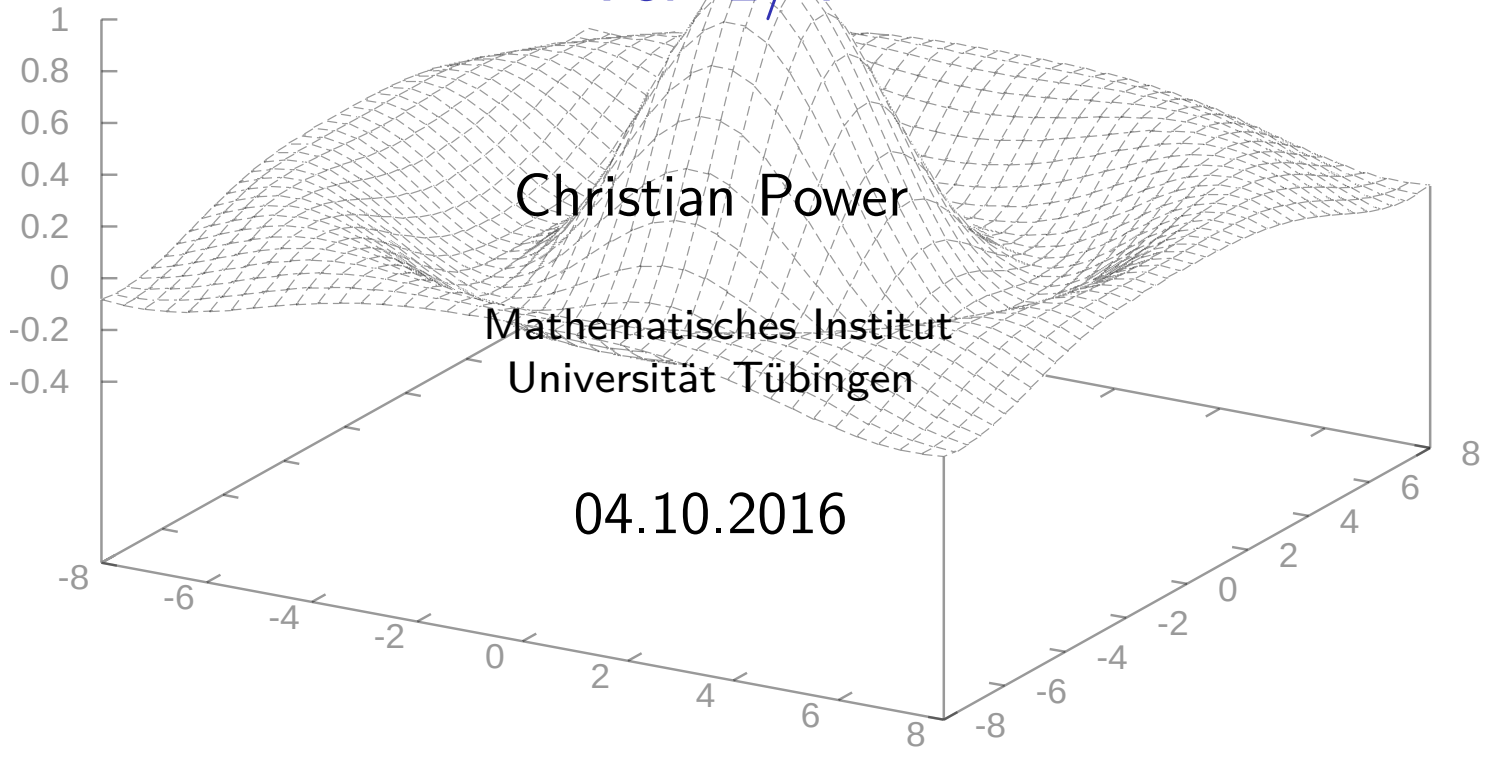
Programmiervorkurs für die Numerik
Teil 1/4



Christian Power

Mathematisches Institut
Universität Tübingen

04.10.2016

Gliederung

Was bedeutet Programmieren?

Übersetzung von Menschensprache in Computersprache.

Analog zum Übersetzungen.

- Der Sinn der Anweisungen in Menschensprache muss dazu nicht notwendigerweise verstanden werden!
- Aber: Das Wissen darum schadet nicht.
- Hingegen: (Minimale) Programmierkenntnisse sind notwendig!

Unterschiede beim Programmieren:

- Man muss sich **vorher** überlegen, was alles passieren kann (z.B. Division durch Null führt zum Abbruch des Programms)
- Man muss sich präzise ausdrücken. Ungenaue Umgangssprache reicht nicht.

Ein Beispiel

Berechnen Sie $S = \sum_{i=0}^{n} \frac{(-1)^i}{(2i+1)}$ für $n = 5$.

```
1  n = 5;   % n+1 == Anzahl der Summanden
2  S = 0;   % Summe
3  for i = 0 : n
4      S = S + (-1) ^ i / (2 * i + 1)
5  end
6  % Ausgabe Summe
7  disp('Ergebnis: S = '); disp(S);
```

Erste Programmiertechniken

- Vernünftiger Einsatz von Kommentaren und Einrückung machen ein Programm wesentlich angenehmer zu lesen und zu verstehen.
- Ein Kommentar sollte
  - sinnvoll sein,
  - das Programm beschreiben
  - und aktuell sein.
- Gute Kommentare sind kurz und knackig.
- Seid konsistent mit der Art eures Einrückens.
- *Warnung:* Die meisten Programme enthalten Kommentare, die zu lang, mehrdeutig oder schlicht falsch sind. Schlechte Kommentare sind schlimmer als keine Kommentare!

Kurze Geschichte zu MATLAB

- MATLAB wurde ursprünglich von Cleve Moler (Univ. of New Mexico) entwickelt.
- Der erste Version war 1984 erhältlich.
- Ziel war es, dass Studenten LINPACK und EISPACK benutzen konnten ohne Fortran lernen zu müssen.
- MATLAB ist kommerziell sehr erfolgreich und wird stetig weiter entwickelt.

Kurze Geschichte zu Julia

- Julia wurde ursprünglich von Jeff Bezanson, Stefan Karpinski, Viral B. Shah, Alan Edelman entwickelt (MIT und Julia Computing).
- Die erste Version war 2012 erhältlich.
- Folgende Ziele wurden versucht zu realisieren.
  1. Julia ist eine schnelle dynamische Programmiersprache mit hohen Abstraktionsgrad.
  2. Julia ist immun gegen das *zwei Sprachen Problem*.
  3. Bei Julia sind *Benutzer gleichzeitig Entwickler*.

Beide Programmiersprachen umfassen:

- Variablen und Daten (Skalare, Vektoren, Matrizen)
- Anweisungen
  - Operatoren (arithmetische Op. z.B. $+$, $-$ und relationale Op. $<$, $>$)
  - Funktionen (sin, Ausgaben von Variablen, eigen Funktionen)
  - Kontrollanweisungen (Schleifen, Verzweigungen)
- Kommentare (zur besseren Lesbarkeit)

Erste Hilfe

Bei MATLAB

- Hilfe in der Menuleiste wählen
- Eingabe von `help` bzw. `doc` im Kommando-Fenster führt auf die Hilfeseiten von MATLAB .

Bei Julia

- Im Kommando-Fenster `?Befehlname` eingeben.
- `http://docs.julialang.org/en/release-0.5/`

Allgemeiner Tipp: Bei Problemen im Internet suchen!
(Stichwort matlab/ julia + gesuchter Begriff).

Auszüge der Syntax

1/2

- Variablen sind Zeichenfolgen ohne `+ - * / %`, die keine Syntax-Elemente sind, z.B.

```
funktionswert = 1 + 3;
andererwert = 2 * funktionswert;
```

- Kommentar ist alles hinter einem Prozentzeichen bzw. einem Kreuz

```
a = 1; % Das ist ein Matlab Kommentar

b = 2; # Das ist ein Julia Kommentar
```

- Strings (Zeichenketten):

```
string1 = 'Dies ist ein Matlab String';

string2 = "Dies ist ein Julia String";
```

Auszüge der Syntax

2/2

- Abschließen eines Ausdrucks und Unterdrücken von Ausgaben mittels Semikolon (;)

```
a = 1 + 1; b = 2 * a; # Keine Ausgabe
a = 1 + 2  # Ausgabe 3
```

- Bei MATLAB werden Code-Zeilen auf mehrere Zeilen via dem 3-Punkt-Operator (...) aufgeteilt

```
f = 1 + 2 + 3 + 4 + ...
    5 + 6  + 7 + 8;
```

- Bei Julia werden unvollständige Code-Zeilen automatisch aufgeteilt.

Operatoren in MATLAB und Julia

- Zuweisungsoperator: =

```
a =3; b = a;
```

- Arithmetische Operatoren: + - * / ^

```
3 + 2; 5 - 1; x = 2^3
```

- Relationale Operatoren: $==$ $\sim=$ $>$ $>=$ $<$ $<=$ (Julia : != anstatt $\sim=$)

```
x = 3 != 4;
println(x) # Ausgabe: true
x = 3; x > 3 # Ausgabe: false
```

- Logische Operatoren: & | $\sim$ (and, or, not); Julia : ! für *not*

```
x = 3;
~(x == 1) & (x < 5)  % Ausgabe: 1
```

if-Abfrage

if *logischer Ausdruck*
    Anweisung(en)
elseif *logischer Ausdruck*
    Anweisung(en)
        ⋮
else
    Anweisung(en)
end

```
1  x = 1;
2  y = 2;
3  if x < y
4    disp('x < y')
5  elseif x > y
6    disp('x > y')
7  else % x == y
8    disp('x == y')
9  end
```

while-Schleife

while *logischer Ausdruck*
    Anweisung(en)
end

```
1  i = 1;
2  while i <= 5 # 5 Itererationen
3        println(i)
4        i += 1
5  end
```

for-Schleife

for *Variable* = *Startwert* : *Inkrement* : *Endwert*
    Anweisung(en)
end

```
1  for i = 1:5
2      println(i)
3  end
```

Ähnlichkeiten zwischen while- und for-Schleife

Eine while-Schleife ist allgemeiner: Es gilt

$$\text{for-Schleife} \quad \Longrightarrow \quad \text{while-Schleife}$$

aber nicht immer

$$\text{for-Schleife} \quad \Longrightarrow \quad \text{while-Schleife}$$

Äquivalente Schleifen:

```
1  % for-Schleife: einfacher zu lesen
2  for n = 4 : 2 : 10
3    n
4  end
5  % while-Schleife: flexibler
6  m = 4;
7  while m < 11
8    m
9    m = m + 2;
10 end
```

break und continue in Schleifen

Die beiden Schlusselwörter veranlassen die Bearbeitung des Schleifeninneren an der Stelle ihres Vorkommens abzubrechen.

- **break:** Setze die Programmabarbeitung unmittelbar hinter der Schleife fort.
- **continue:** Springe an den Schleifenanfang und setzt die Programmabarbeitung mit der Auswertung der Schleifenbedingung fortgesetzt.

```
1  for i = 1:10
2      if i == 4
3          continue
4      end
5      if i == 8
6          break
7      end
8      i
9  end
```

Bemerkung
1/3

- Wo *Whitespaces*, also Leerzeichen, Tabulatoren erlaubt oder erforderlich sind, können beliebig viele davon zusammenhängend verwendet werden. (Also ein Leerzeichen oder z.B. 10 für schöne Einrückung)
- In *Terminal-Fenstern* oder *Command Windows* muss nach der Texteingabe immer die Entertaste gedrückt werden, damit der Text bearbeitet (ausgeführt) wird.
- Vor dem Ausführen des eigenen Programms in MATLAB das Speichern im Texteditor nicht vergessen.
- Vorsicht bei der Benutzung von `i` und `j`: beide können in MATLAB für die komplexe Zahl `i` stehen. Bei Julia ist `im` die imaginäre Einheit.

Bemerkung
2/3

- Der Sinn von Variablen besteht darin, Daten (eine Zahl, ein Vektor, etc.) für eine spätere Verarbeitung aufzubewahren, z.B. für den nächsten Schleifendurchgang.
- Nicht vergessen, einer Variable einen Wert zuzuweisen, bevor sie verwendet wird.
- Um innerhalb eines Programms den Wert einer (existierenden) Variable zu kontrollieren, fügt man an der interessierenden Stelle ein Zeile mit nur dem Namen der Variable ein.
- Das Malzeichen zwischen zwei Faktoren darf nicht weggelassen werden.

Bemerkung
3/3

- Operatoren haben immer einen oder mehrere Operanden. Z.B. Der `+` – Operator erwartet links und rechts von ihm einen Operanden: `a + b`.
- Sind mehrere Additionen verkettet, etwa `a + b + c + d` klammert MATLAB implizit und berechnet folgenden Ausdruck: `((a + b) + c) + d`. Die i-ten Partialsummen S_i in den Übungen berechnen sich mit $q_i = q(t)$ als

$$S_i = S_{i-1} + q_i = ((((q_1 + q_2) + q_3) + \cdots + q_{i-1}) + q_i).$$

Postskriptum

Wir haben gelernt:

- Variablen zu benennen und Werten zu zuweisen.
- `if`-Bedingung zu benutzen.
- `for`-Schleifen zu benutzen.

Vom philosophischen Standpunkt aus kann man jetzt jedes Programm auf einen Computer schreiben — der Rest sind Details! D.h. die Details sind sehr wichtig.