Einführung in die Programmierung mit Matlab
Teil 4/4

Thomas Dunst

Mathematisches Institut
Universität Tübingen

(e–mail: progtutor@na.uni-tuebingen.de)

12. Oktober 2012

Wiederholung:

Was haben wir bislang gemacht:

- Kontrollanweisungen (Schleifen, Verzweigungen)
- Vektoren und Matrizen (Initialisierung, Operationen)
- Funktionen und Skripte (Aufbau, Funktionsweise)
- Einfache Visualisierungen

```
1  function [s] = leib_partsum(n)
2       s = 0;
3       for i=0:1:n
4            s = s + ((-1)^i)/(2*i+1);
5       end
6  end
```

```
1  n=1:1:20;
2  for i=1:20
3   S(i)= leib_partsum(n(i));
4   T(i)= pi/4;
5  end
6  plot(n,S)
7  hold on
8  plot(n,T)
```

Gliederung

Lösungen der Aufgaben des 3. Übungsblattes

Aufgabe 1:

Datei `qua.m`:

```
1  function [] = qua(q)
2  % Funktion gibt q^2 aus
3  % Input:
4  % Skalar q
5     q^2
6  end
```

Die Zahlen funktionieren, 'aha' gibt natürlich einen Fehler.

Lösungen der Aufgaben des 3. Übungsblattes

Aufgabe 1 – 4

Datei qua.m:

```
1  function [res] = qua(q)
2  % Funktion gibt q^2 zurueck
3  % Input:
4  % Skalar q
5  % Output:
6  % Skalar res
7  res = q^2;  % Zuweisung "stumm" wegen ";"
8  end
```

Skript quaskript.m:

```
1  % Testvektor
2  v = [-2  0  3  0.2];
3
4  s = 0; % Initialisieren der Summe
5  n = length(v); % Laenge des Vektors bestimmen
6  for k = 1 : n
7    s = s + qua(v(k));
8  end
9
10 sqrt(s) % Ausgabe des Ergebnisses
```

Lösungen der Aufgaben des 3. Übungsblattes

Aufgabe 5 – 7

Datei sum_part.m:

```
 1  function [s] = sum_part(q, n)
 2  % Input:
 3  % Skalar q: zu entwickelnde Zahl
 4  % Skalar n: Zahl der aufsummierten Folgen -1
 5  % Output:
 6  % Skalar s: Partialsumme S_n
 7  s = 0; % Initialisierung Summe
 8  for i = 0 : n
 9    s = s + q ^ i;
10  end
```

Lösungen der Aufgaben des 3. Übungsblattes

Skript `script_7.m`:

```
 1  % Testvektor
 2  p = [-2   0   3   0.2];
 3
 4  s = 0; % Initialisierung der Summe
 5  n = length(p); % Dim. des Vektors
 6  for k = 1 : n
 7    v(k) = sum_part(p(k),4);
 8  end
 9  v % Ausgabe
10  save('mein_fau', 'v');
```

Lösungen der Aufgaben des 3. Übungsblattes

Aufgabe 9

Datei geo_R.m:

```
 1  function [C] = geo_R(z,n)
 2  % Input:
 3  % Skalar z: zu entwickelnde Zahl
 4  % Skalar n: ersten betrachten n+1 Summanden
 5
 6  C = 0; % Initialisierung der Summe
 7  for k=0:n
 8      C = C + z^i;
 9  end
10
11  end
```

Skript geo_R_skript.m:

```
1  z = 0.25; % vorgegebener Entwicklungspunkt
2  n = [5 10 50 100]; % vorgegebene n's
3  m = length(n); % Laenge des Vektors bestimmen
4  for i=1:m
5      geo_R(z,n(i))
6  end
```

Lösungen der Aufgaben des 3. Übungsblattes

Aufgabe 10:

Datei `mmult.m`:

```
 1  function [C] = mmult(A, B)
 2  % Input: Matrix A und Matrix B
 3  % Output: Matrix C
 4  [n m] = size(A); % Dim. der Matrix
 5  [r p] = size(B); % Dim. der Matrix
 6
 7  C = zeros(n,p);    % Initialisieren von C
 8  for ii = 1 : n     % ueber alle Zeilen von C
 9    for jj = 1 : p   % ueber alle Spalten von C
10      for k = 1 : m
11        % Aufsummieren der n Spalten A und Zeilen B.
12        C(ii,jj) = C(ii,jj) + A(ii,k) * B(k, jj)
13      end
14    end
15  end
16
17
18  end
```

Lösungen der Aufgaben des 3. Übungsblattes

Aufgabe 11:

Datei `f.m`:

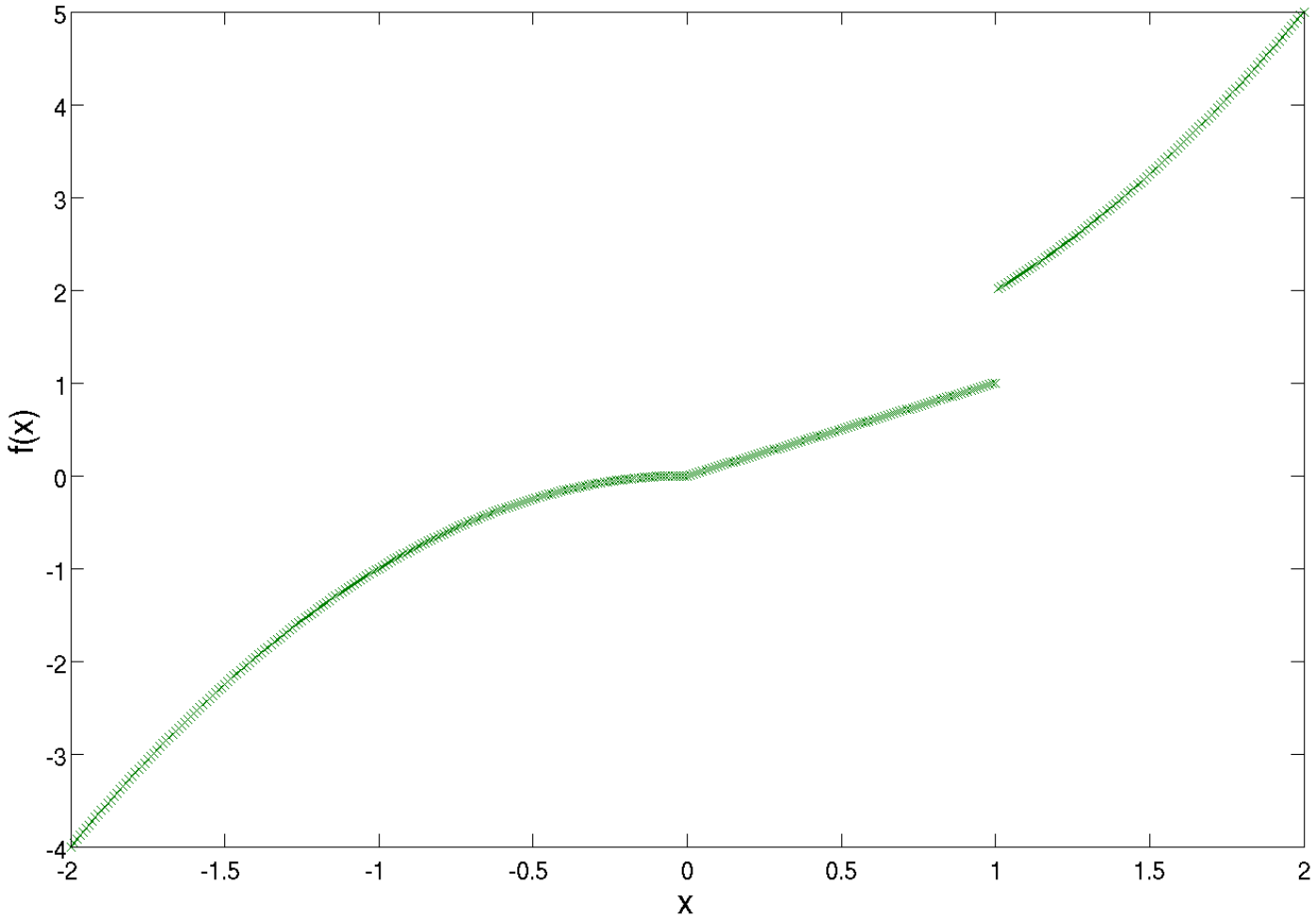
```
 1  function [y] = f(x)
 2  % Input: Skalar x
 3  % Output: Skalar y
 4
 5  if x<0   % Fall 1: x<0
 6    y=-x^2;
 7  elseif x<=1 % Fall 2: x>=0 und x<=1
 8    y=x;
 9  else % Fall 3: sonst
10    y = x^2 +1;
11  end
12  end
```

Skript `plottenf.m`:

```
1  x=-2:0.01:2; % Diskretisieren von [-2,2]
2  n = length(x); % Dim. des Vektors
3  for ii=1:n
4    y(ii)=f(x(ii));
5  end
6  plot(x,y)
```

Lösungen der Aufgaben des 3. Übungsblattes

Resultierendes Schaubild:



Lösungen der Aufgaben des 3. Übungsblattes

Aufgabe 12:

Datei `fschar.m`:

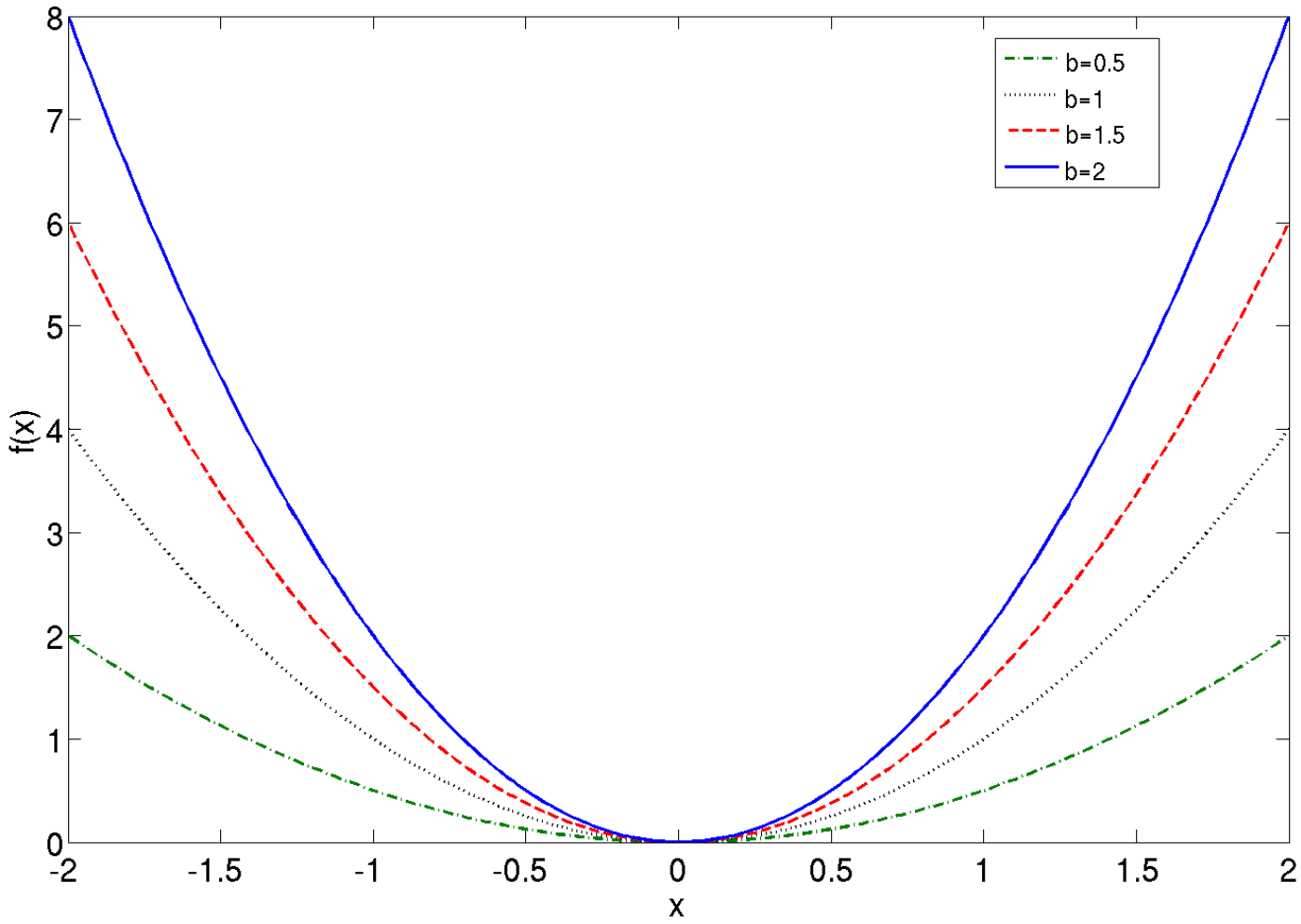
```
1  function [y] = fschar(x,b)
2  % Input: Vektor x, Skalar b
3  % Output: Vektor y
4  y = b*x.^2;
5  end
```

Skript `plottenfschar.m`:

```
1  x=-2:0.01:2; % Diskretisieren von [-2,2]
2  b = [0.5 1 1.5 2]; % Werte des Parameter
3  n = length(b); % Dim. des Vektors
4  m = length(x); % Dim. des Vektors
5  for jj=1:n % Fuer alle Parameter b
6      y(:,jj)=fschar(x,b(jj));
7  end
8  for jj=1:n
9    plot(x,y(:,jj))
10   hold on % Alle in das selbe Schaubild
11 end
```

Lösungen der Aufgaben des 3. Übungsblattes

Resultierendes Schaubild:



Variablen laden und speichern

Befehle `save` und `load`

- Speichern und Laden von Variablen (also den Daten)
- Benutzung in Skripten oder im Command Window
- Syntax für `save`: (eine der möglichen)
  - `save('dateiname')`
    speichert alle aktuellen Variablen in der Datei namens `dateiname.mat`
  - `save('dateiname','A',B','b')`
    speichert genau die Variablen `A, B, b` in der Datei namens `dateiname.mat`
- Syntax für `load`: (eine der möglichen)
  - `load('dateiname')`
    lädt alle aktuellen Variablen der Datei namens `dateiname.mat` in Matlabs Workspace.
  - `load('dateiname','A',B','b')`
    lädt nur die angegebenen Variablen `A, B, b` aus der Datei namens `dateiname.mat` in Matlabs Workspace.

Wo ist der Fehler?!?

Fehlersuche

Jeder hat seine eigene Methode, Fehler zu suchen. Dennoch ein paar Tipps:

- Ganz Ruhig: Es ist nur ein Fehler, nichts schlimmes. Fehler sind absolut normal.
- Systematik: gehen Sie systematisch vor!
- Fehlermeldungen: lesen Sie die Fehlermeldungen! - Zumindest die Zeilenangaben sind verständlich. Verlassen Sie sich dennoch nicht hundertprozentig auf die Fehlermeldungen.
- Fehlereinschätzung: Ist es eher ein einfacher Syntaxfehler? Was soll denn überhaupt gemacht werden?
- Fehlereingrenzung: isolieren Sie den Fehler.
  - Kommentieren Sie Code aus, der nicht unbedingt nötig ist. Taucht der Fehler immer noch auf?
  - Hat der Codeabschnitt vorher in veränderter Form schon mal funktioniert. Was genau ist jetzt anders. Versuchen Sie, vom ursprünglichen Zustand aus den neu angestrebten zu erreichen.
- Fragen Sie Andere: Das ist manchmal die schnellste Methode.

Programmieraufgaben Numerik

Allgemeines zu den Programmieraufgaben

- Bearbeitung in Teams (2-3 Personen) wird empfohlen.
- Man darf maximal zu Dritt abgeben.
- Sprechstunde des Programmiertutors: Mo 15.00 - 16.00.
- Zulassungskriterium für die Klausur:
  50% der Programmieraufgaben müssen abgegeben und akzeptiert sein.
- Nur lauffähige Programme werden bewertet.

Alle Infos stehen auf der Homepage:
`http://na.uni-tuebingen.de`

Allgemeines zu den Programmieraufgaben: Die Abgabe.

Abzugebende Funktionen sollten folgendes Format haben:

```
1  function [C] = Funktionsname(A,B)
2  % Ein paar Worte was die Funktion macht
3  % Input:
4  % Matrix A Beschreibung
5  % Matrix B Beschreibung
6  % Output:
7  % Matrix C Beschreibung
8  %
9  % von: Name1, Name2, Name3
10 % Datum: xx.xx.xx
11
12 end
```

Abzugebende Skripte sollten folgendes Format haben:

```
1  % Ein paar Worte was das Skript macht
2  %
3  % von: Name1, Name2, Name3
4  % Datum: xx.xx.xx
```

Allgemeines zu den Programmieraufgaben: Die Abgabe.

- Die Funktionsnamen sowie Input- bzw. Outputvariablen-namen sind fest durch die Aufgabenstellung vorgegeben.
- Versendet wird:
  - Ein Zip- bzw Rar-File mit dem Namen:
    `ProgA1_name1_name2_name3.zip`
    bzw. `ProgA1_name1_name2_name3.rar`
    (A1 steht dabei für Programmieraufgabe 1)
  - Darin befindet sich ein Ordner mit dem Namen `ProgA1_name1_name2_name3` indem die abzugebenden Dateien sind.

  an `progtutor@na.uni-tuebingen.de`.
  Betreff der Email: ProgA1 Abgabe name1 name2 name3
- Alles sollte gut kommentiert sein!
- Es dürfen nur skalarwertige Rechenoperationen benutzt werden.

Allgemeines zu den Programmieraufgaben: Das Testat.

- Jeder Übungsteilnehmer wird mindestens einmal einzeln testiert.
- Thema sind alle von ihm abgegebenen Programmieraufgaben.
- Man wird ca. eine Woche vor dem Testat dazu eingeladen.
- Termine für die Testate:
  - Mo 16.00 - 18.00
  - Di 18.00 - 20.00
  - Do 18.00 - 20.00

  (In der ersten Vorlesungsstunde kann man sich für einen Termin eintragen)
- War die Teilnahme am Testat erfolglos, werden alle entsprechenden Programmieraufgaben als nicht akzeptiert gewertet.

Fragen