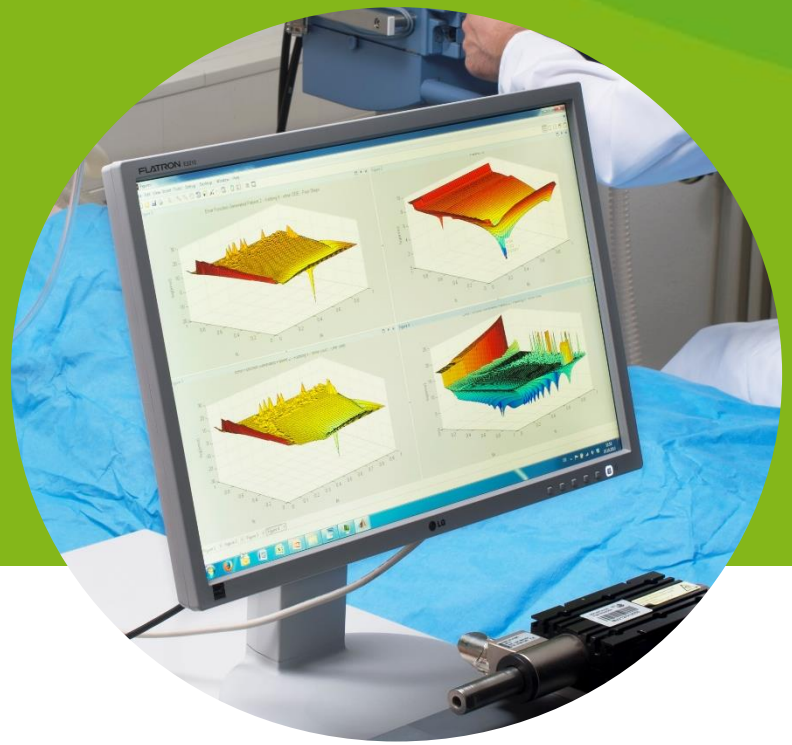
Einsatz von MATLAB Grader zur Ergänzung der akademischen Lehre

Jörn Kretschmer[1]



[1]Hochschule Furtwangen, Fakultät Mechanical and Medical Engineering, Villingen-Schwenningen

Übersicht

- Vorlesung *Computermathematik*
- Einsatz von MATLAB Grader in Vorlesung / Übungsaufgaben
- Erstellen von Aufgaben / Skripten in Grader
- Testen von Lösungen

Studiengänge

Bachelor

Maschinenbau und Mechatronik

- Vorlesung über 2 Semester (2 x 1 ECTS)
- Enge Verknüpfung mit Vorlesung *Mathematik*
- Vorbereitung auf Vorlesung *Ingenieursmathematik*

Lernziele

- Grundlagen Programmierung
  - Variablen
  - Funktionen
  - Ablaufsteuerung
- Mathematische Berechnungen
- Grafische Ausgabe

Medical Engineering

Master

Biomedical Engineering

- Vorlesung über 1 Semester (3 ECTS)
- Vorbereitung auf Vorlesungen *Simulation, Systemidentifikation, Modellbildung*

Lernziele

- Grundlagen Programmierung
- Mathematische Berechnungen
- Grafische Ausgabe
- Parameteridentifikation
- Grafische Oberflächen
- SIMULINK

Bachelor

Master

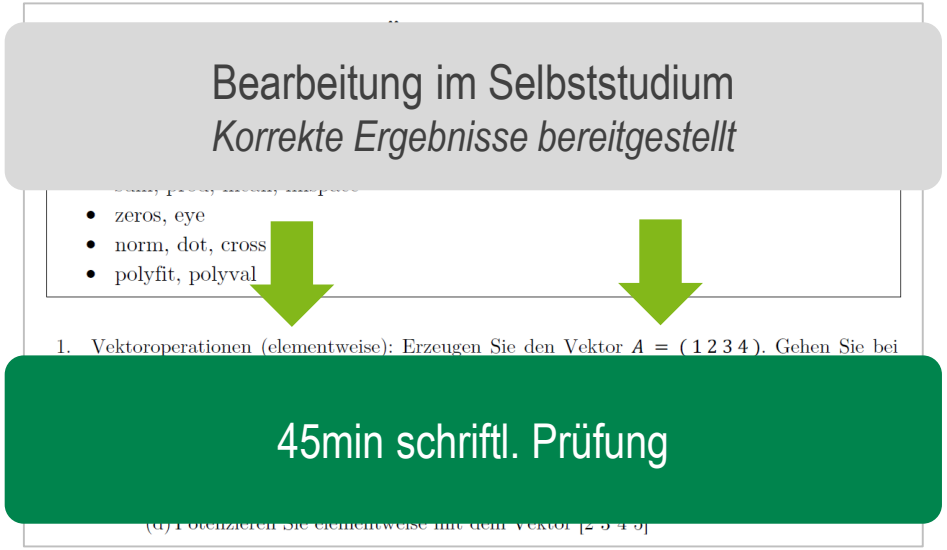
Maschinenbau und Mechatronik

Medical Engineering

Biomedical Engineering

Ablauf *(früher)*

Skript +
6 Übungsblätter mit Aufgaben



Bearbeitung im Selbststudium
Korrekte Ergebnisse bereitgestellt

45min schriftl. Prüfung

Vorlesung

Übungsblatt

Abschlußprojekt

Probleme

Studierenden-Feedback

- Kein Feedback über Bearbeitungsstand der Übungsblätter
- Wenig Rückmeldung über Probleme

Motivation

- Abarbeiten der Übungsblätter ohne Feedback über Richtigkeit der (Teil-)Lösungen
- Übungsblätter bieten kaum Lösungshinweise bei Problemen

Personalaufwand

Bachelor

Maschinenbau und Mechatronik

Medical Engineering

Master

Biomedical Engineering

HOCHSCHULE FURTWANGEN UNIVERSITY HFU

Ablauf *(neu)*

MATLAB Grader

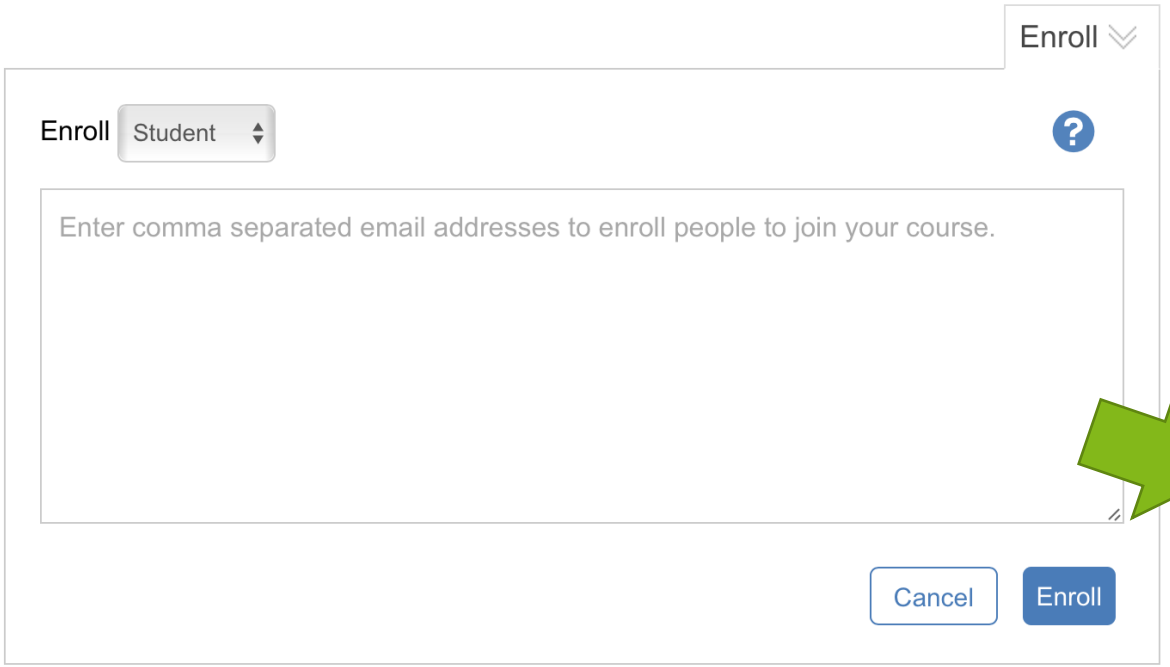
- Skript mit Erklärungen zum Thema
- Thematisch abgestimmte Aufgaben

- Vorlesung

MATLAB Grader

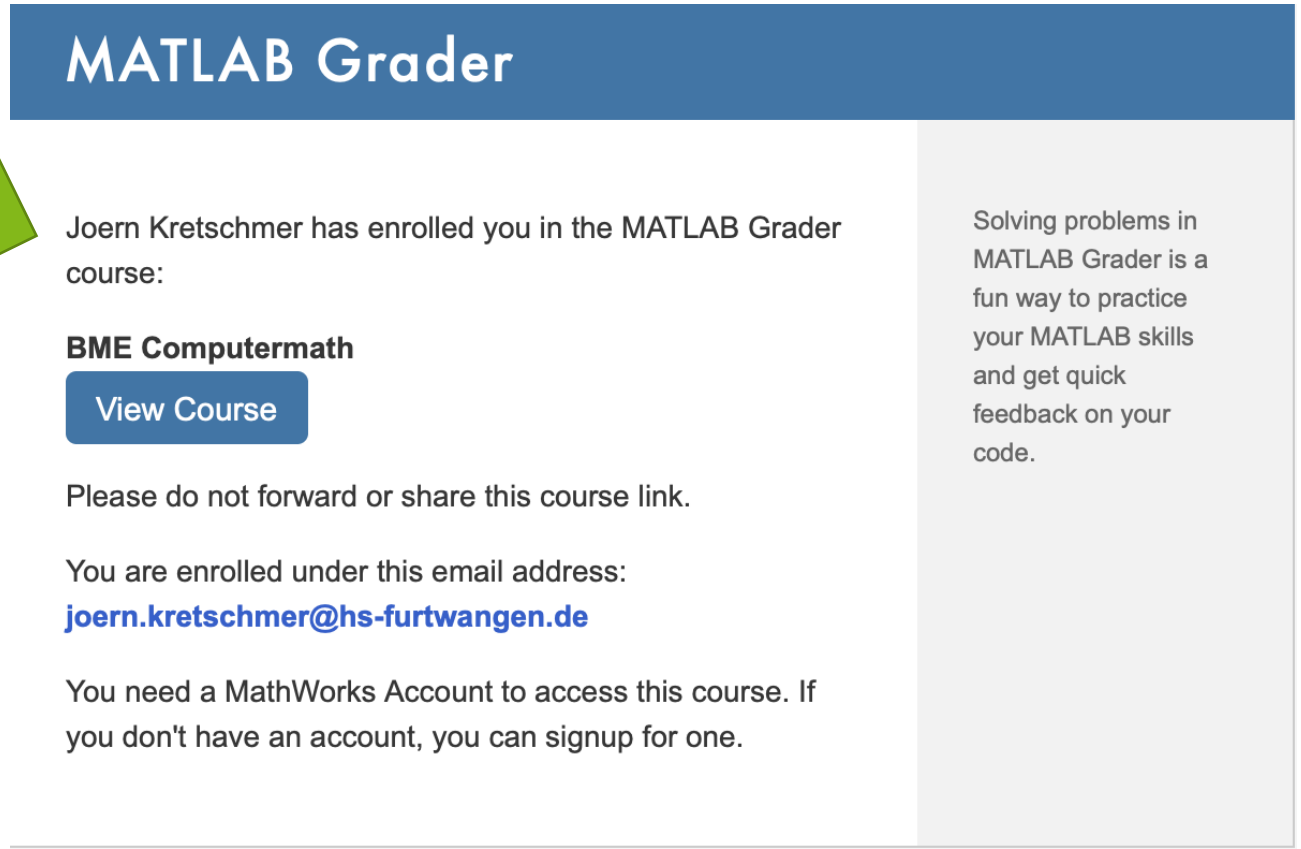
- Übungsblatt

Manage People



- Erstellen eigener Kurse mit Aufgaben unter *grader.mathworks.com*
- Studierende erhalten Einladung per Email
- Studierende benötigen Mathworks-Account aber keine Lizenz

MATLAB Grader



HOCHSCHULE FURTWANGEN UNIVERSITY | HFU

Vorlesung
Ablaufsteuerung

Vorlesungsinhalt

- Bedingungen
- Nutzung von Schleifen zur wiederholten Durchführung von Rechenoperationen
- Unterschiede zählergesteuerte vs. bedingungsgesteuerte Schleifen

Aufgaben

- Bedingungen (*if*, *if*/*else*)
- Verknüpfen von Bedingungen
- Diskrete Fälle (*switch-case*)
- Auswahl des richtigen Schleifentyps
- Geschachtelte Schleifen

Beispiel: Numerische Berechnung der Cosinus-Funktion

$$\cos(x) = \sum_{i=0}^{n} (-1)^i \frac{x^{2i}}{(2i)!}$$

Problem Description and Instructions*

Text | Code | B | I | U | M | Abcd | Head | Σ

To numerically calculate the cosine function, calculators use the following approximation:

$$cos(x) = \sum_{i=0}^{n} (-1)^i \frac{x^{2i}}{(2i)!}$$

The larger *n* is, the closer that approximation is to the exact result.

Calculate the approximation for $cos(2)$ with $n = 4$. To calculate the factorial of a number , use `factorial`. Save the result in a variable ***c***. Which type of loop suits the task better?

Beispiel: Numerische Berechnung der Cosinus-Funktion

Problem Type* ?

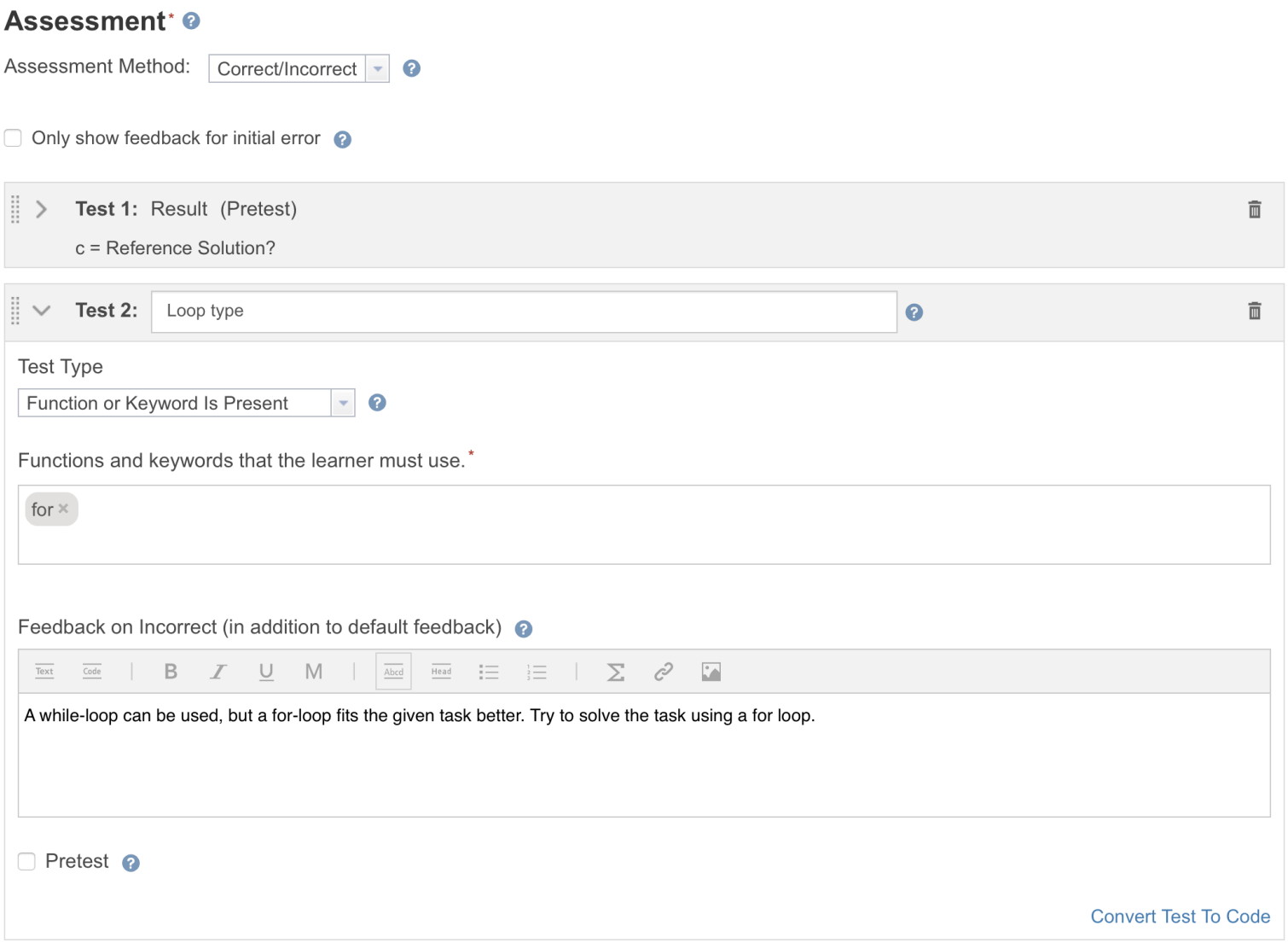
(•) Script ( ) Function

Code

Reference Solution ? | Learner Template ?

```
1 c = 0;
2 n = 4;
3 x = 2;
4 for i = 0:n
5     c = c+(-1)^i*(x^(2*i))/(factorial(2*i));
6 end
```

Beispiel: Numerische Berechnung der Cosinus-Funktion



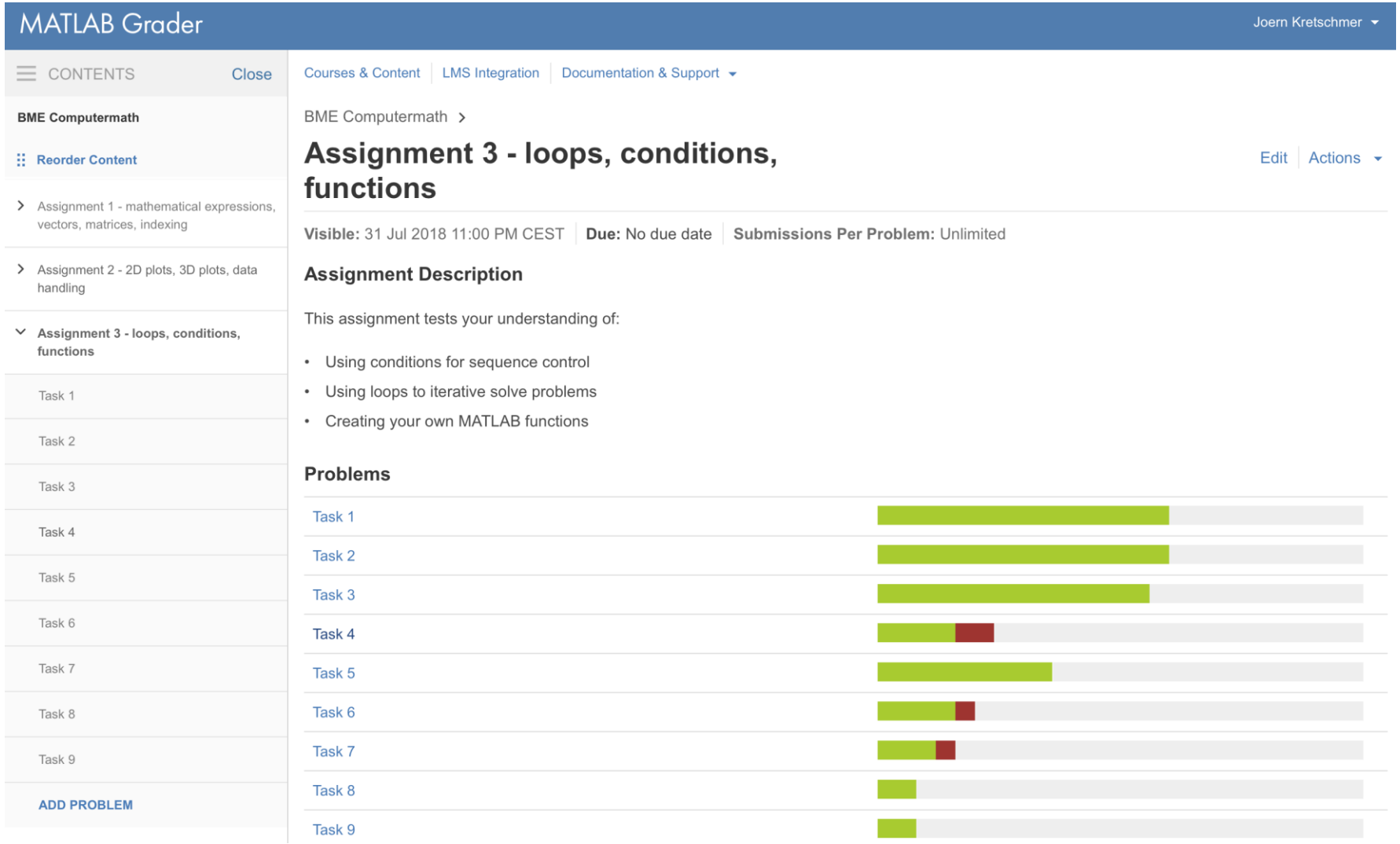
Tests

- Korrektes Ergebnis
- Einsatz einer `for`-Schleife
- Lösung ohne Benutzen der `cos()` Funktion

Überblick Bearbeitungsstatus der Übungsblätter

HOCHSCHULE FURTWANGEN UNIVERSITY HFU

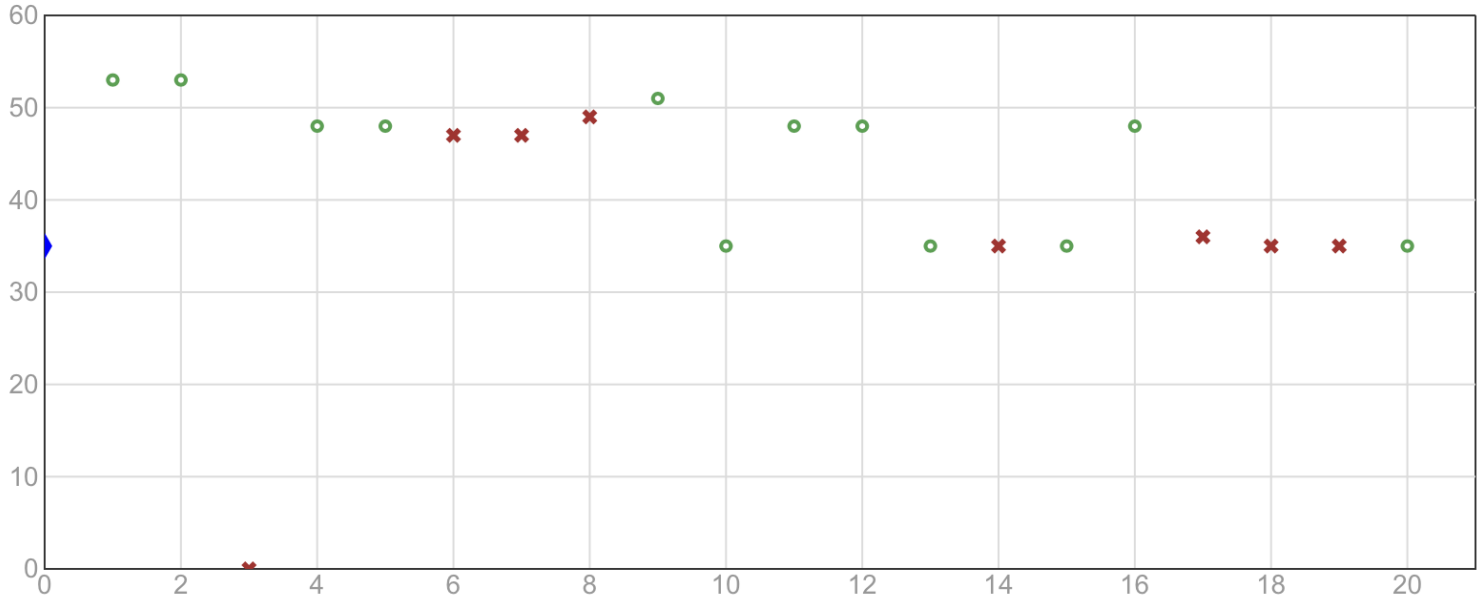
Vorlesung
Ablaufsteuerung



Überblick Bearbeitungsstatus der Übungsblätter

Student Solutions

View Student Solutions



Überblick Studierendenlösungen

$$\cos(x) = \sum_{i=0}^{n} (-1)^i \frac{x^{2i}}{(2i)!}$$

HOCHSCHULE FURTWANGEN UNIVERSITY | HFU

Vorlesung
Ablaufsteuerung

Solution 1: 1 of 2 tests passed
Submitted on 8 Nov 2018 by [redacted] | ID: [redacted] | Size: 57
Test Results

```
x=2;
n=4;
c=0;
i=1;
for k = i:n
    result(k)= ((-1).^k)*((x.^(2*k))/(factorial(2*k)));
    c = c+result(k);
end
```

Schleife startet bei i = 1 → Index 0 in MATLAB nicht möglich, für die Aufgabe aber kein Vektor nötig

Solution 1: All tests passed
Submitted on 27 Oct 2018 by [redacted] | ID: [redacted] | Size: 45
Test Results

```

R0=1   % matlab can't calculate R(0)
for i=1:4

    R(i)=((-1)^i)*(2^(2*i))/factorial([2*i])

c=sum(R)+R0;
end


```

Beste Lösung, aber für die Aufgabe kein Vektor nötig & Berechnung von *c* außerhalb der Schleife besser

Solution 1: All tests passed
Submitted on 25 Oct 2018 by [redacted] | ID: [redacted] | Size: 79
Test Results

```


x=2
sum1=[0 0 0 0 0]
for n = 0:4


    if n == 0
        sum1(1) = (1).*( 1 /1);
    else
```

Lösung korrekt, aber für die Aufgabe kein Vektor nötig & Unterscheidung n == 0 / n~=0 über if/else zu aufwendig

Homogene LGS

Homogene LGS, also beispielsweise:

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 0 \end{array}\right)$$

haben immer eine triviale Lösung (im Beispiel oben: $x = 0, y = 0, z = 0$). Darüber hinaus können Sie unendlich viele Lösungen besitzen. Wie im vorigen Übungsblatt erläutert, ist dies aus der Determinante ersichtlich.

```
A = [1 2 3;4 5 6; 7 8 9];
det(A)

Ausgabe:

ans =

  -9.5162e-16
```

Die Determinante der Matrix ist 0 (vom numerischen Rundungsfehler abgesehen). Das Gleichungssystem ist also unterbestimmt (die dritte Zeile entspricht der Differenz des Doppelten der zweiten Zeile und der ersten Zeile) und besitzt damit unendlich viele Lösungen. Die Determinante ist jedoch nur bei quadratischen Matrizen ermittelbar. Zum gleichen Ergebnis käme man, indem der Rang der Matrix ermittelt wird:

```
rank(B)

Ausgabe:

ans =

    2
```

Der Rang der Matrix ist kleiner als ihre Dimension, d.h. sie ist nicht eindeutig lösbar.

Besitzt ein homogenes LGS unendlich viele Lösungen, können diese nicht über Linksdivision berechnet werden. Zur Lösung wird der Befehl `null` eingesetzt, der eine Matrix bestehend aus Basis aller Lösungen des LGS zurückliefert.

Beispiel:

$$\begin{pmatrix} 2 & -1 & 4 \\ -4 & 5 & 3 \\ -2 & 4 & 7 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

```
M = [2 -1 4;-4 5 3;-2 4 7];

det(M)
```

Aufgaben:

a) Welche Lösungen hat folgendes LGS?

$$\left(\begin{array}{ccc|c} 6 & 4 & 8 & 0 \\ 3 & 9 & 3 & 0 \\ 6 & 18 & 6 & 0 \end{array}\right)$$

Geben Sie den Lösungsraum an.

b) Prüfen Sie, ob $x_1 = \begin{pmatrix} -10 \\ 1 \\ 7 \end{pmatrix}$ und $x_2 = \begin{pmatrix} -5 \\ \frac{1}{2} \\ \frac{7}{2} \end{pmatrix}$ Lösungen des LGS aus a) sind. Berechnen Sie dazu das Ergebnis, wenn Sie x_1 und x_2 in das LGS einsetzen.

c) Untersuchen Sie das Lösungsverhalten des folgenden LGS:

$$\left(\begin{array}{cccc|c} 2 & 5 & -1 & 0 & -1 \\ -1 & 8 & 8 & -4 & -13 \\ 4 & 2 & -16 & 10 & 0 \\ 0 & 1 & 1 & -4 & -1 \end{array}\right)$$

Berechnen Sie dazu den Rang der Matrix M. Falls das Gleichungssystem lösbar ist, berechnen Sie den Lösungsvektor x mit der Linksdivision, dem Gaussverfahren und symbolisch.

d) Untersuchen Sie folgendes LGS auf Lösbarkeit. Berechnen Sie dazu den Rang und den Lösungsraum. Berechnen Sie auch die spezielle Lösung x_s.

$$\left(\begin{array}{ccc|c} 3 & 8 & 4 & -1 \\ 2 & 13 & 7 & -4 \\ 4 & 3 & 1 & 2 \end{array}\right)$$

Der Vektor $x_1 = \begin{pmatrix} 4.775 \\ -13.2688 \\ 22.7063 \end{pmatrix}$ löst das LGS ebenfalls. Wie berechnet sich x_1 aus x_s und N? Geben Sie den Faktor t an.

Your Script

Reset MATLAB Documentation

```
% a)
N_a =  %Lösungsraum

% b)
erg_x1 =  % Validierung der Lösung x1
```

HOCHSCHULE FURTWANGEN UNIVERSITY HFU

Thema *Gleichungen*

Kurze Wiederholung der mathematischen Grundlagen in Form eines Skripts

Kurze Erklärung der benötigten Befehle

Abschließende Aufgaben zum Thema

Beispiel: Funktion

Aufgabe

Schreiben Sie eine Funktion mit dem Namen **circumference**, die den Umfang eines Kreises basierend auf dem Radius berechnet. Welche Funktionsparameter und Rückgabewerte benötigen Sie?

How to Call the Function (when the learner clicks 'Run')

```
% Testaufruf der Funktion
result = circumference(10)
```

```
% Run learner solution.
r = 5;
erg = circumference(r);

% Run reference solution.
ergReference =
reference.circumference(r);

% Compare.
assessVariableEqual('erg', ergReference);
```

Beispiel: Logische Indizierung

Aufgabe
Speichern Sie alle Elemente des Vektors ***v***, die zwischen 2 und 8 (beides exklusive) liegen, in einen Vektor ***v28***

Testen *Versteckte Variablen*

Hochladen eines Skripts, das den Vektor *v* erzeugt. Ist von den Studierenden nicht einzusehen.

`secretvector3.m`

```
v = [1 9 3 2 8 7 3 1 0 2 3 1 -4 3 -2 10];
```

Your Script

Reset MATLAB Documentation

```
1 secret_vector3;
2
3 %Aufgabe a)
4
```

→ Eigene Funktionen können hochgeladen und den Studierenden zur Verwendung vorgegeben werden

Beispiel: Logische Indizierung

Aufgabe
Speichern Sie alle Elemente des Vektors ***v***, die zwischen 2 und 8 (beides exklusive) liegen, in einen Vektor ***v28***

Testen *Versteckte Variablen*

Hochladen eines Skripts, das den Vektor *v* erzeugt. Ist von den Studierenden nicht einzusehen.

`secretvector3.m`

```
v = [1 9 3 2 8 7 3 1 0 2 3 1 -4 3 -2 10];
```

Your Script

Reset | MATLAB Documentation

```
1 secret_vector3;
2 v
3 %Aufgabe a)
4
```

Studierende können den Code ausführen und durch Weglassen des Semikolon den Inhalt von *v* einsehen

Run Script

Lösung: Zufallsvektoren bzw. zufällige Auswahl aus mehreren hinterlegten Vektoren

Testen
Vorlagen

Beispiel: Verknüpfte Bedingungen

Aufgabe

Vervollständigen Sie die Funktion unten, so dass x durch 6 geteilt wird, wenn es durch 2 und durch 3 teilbar ist oder wenn x größer als 1000 ist. Speichern Sie das Ergebnis dann jeweils in y. Verwenden Sie nur eine einzige if- Anweisung.

Teillösungen können vorgegeben werden

Your Function

Reset MATLAB Documentation

```
1 function y = TestX(x)
2     y = x;
3
4
5
6 end
7
```

Beispiel: Verknüpfte Bedingungen

Aufgabe

Vervollständigen Sie die Funktion unten so, dass x durch 6 geteilt wird, wenn es durch 2 und durch 3 teilbar ist oder wenn x größer als 1000 ist. Speichern Sie das Ergebnis dann jeweils in y. Verwenden Sie nur **eine einzige if- Anweisung**.

Über `fileread()` Studierendenlösung einlesen und mit `strfind()` Anzahl der `if`-Aufrufe zählen. Prüfung mit `assert()`

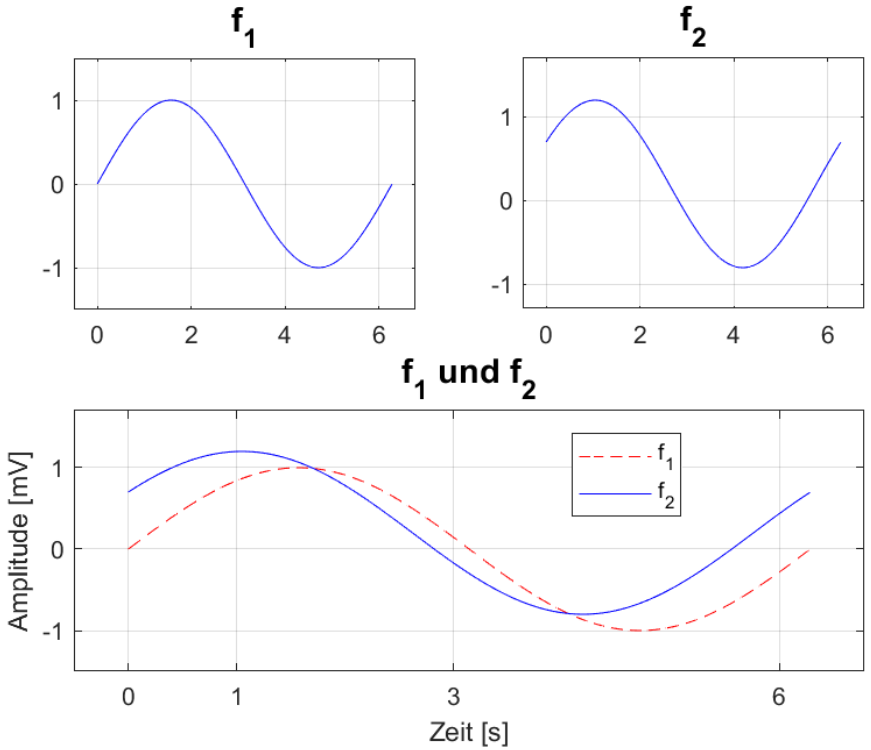
```
% Studierendenlösung einlesen
F = fileread('TestX.m');
% Aufrufe von if zählen
ifs=strfind(F,'if');
% Mehr als ein Aufruf gefunden?
iffind = length(ifs)<2;
% Prüfung
assert(iffind,'Ihre Funktion enthält mehr als eine if-Anweisung');
```

Testen
Plots

Beispiel: 2D Plots

Aufgabe

Versuchen Sie, folgende Grafik nachzustellen. Die Funktionen sind $f_1(x) = \sin(x)$ und $f_2(x) = \sin\left(x + \frac{\pi}{6}\right) + 0.2$. Stellen Sie die beiden Diagramme im Bereich 0 bis $\boldsymbol{\pi}$ mit einem Inkrement von 0.01 dar. Der Abstand der Diagramme ist zu allen Seiten jeweils 0.5.



Tests
- Funktionswerte
- Farben
- Linientypen
- Überschriften
- Achsenbeschriftungen
- Legende
- Gitter
- Aufteilung der Subplots

Test: Funktionswerte f_1 im oberen linken Fenster

```
% Referenzwerte importieren
load('Data_Plot_A2.mat');

% Studierendenlösung öffnen und Funktionswerte extrahieren
figure(2)
subplot(2,2,1);
h = gca;
hc = h.get('Children')
y1_1_t = hc(1).YData;

% Mit Referenzwerten vergleichen
test1 = y1 == y1_1_t;
assert(sum(test1==false)==0,...
'Daten von f1 im oberen linken Fenster nicht korrekt');
```

Test: Darstellung f_1 im oberen linken Fenster

Testen
Plots

```
% Studierendenlösung öffnen
figure(2)
subplot(2,2,1);
h = gca;
hc = h.get('Children')
l1_1 = hc(1);

% Linientyp
assert(strcmp(l1_1.LineStyle,'-'),...
'Linientyp in Subplot links oben nicht korrekt');
% Linienfarbe
test = l1_1.Color == [0 0 1];
assert(sum(test==false)==0,...
'Linienfarbe in Subplot links oben nicht korrekt');
% Markertyp
assert(strcmp(l1_1.Marker,'none'),...
'Markertyp in Subplot links oben nicht korrekt');
```

Testen der Teillösungen mit `assert()`

Fazit

- Plattform bietet viele Vorteile gegenüber klassischen Übungsblättern
  - Motivation
  - Automatisiertes Testen und Feedback
  - Anzeige des individuellen Fortschritts
- Studierende benötigen keine eigene MATLAB Lizenz bzw. Installation
  - MATLAB Grader auch auf Mobilgeräten (Tablets, Smartphones,…) ausführbar
- Viele Tools zum Erstellen der Aufgaben oder von Skripten

Fazit

- In Aufgabenblättern vom Typ *Funktion* nur eine Funktion erstellbar
  - Nachteil: Komplexe Aufgaben mit mehreren Funktionen müssen aufgeteilt werden
  - Vorteil: Schrittweises Vorangehen der Studierenden mit Tests nach jedem Schritt
- Kein Debugging möglich
- Tests, die über die Standardtests hinausgehen, müssen über `assert()` geprüft werden
  - Komplexe Tests (Beispiel: Plots)
- SIMULINK derzeit noch nicht implementiert
- Keine zufällige Zuweisung von Aufgaben (z.B. *e*Klausuren)
  - Integration in LMS notwendig

Kontakt

Dr. Jörn Kretschmer
Akademischer Mitarbeiter
Fakultät Mechanical and Medical Engineering
Hochschule Furtwangen
Campus Villingen-Schwenningen
Email: krj@hs-furtwangen.de