Cody Coursework in der Lehre

MATLAB Expo 2016

Dr. Martin Bracke, Dr. Andreas Roth

10. Mai 2016

1. Veranstaltung Einführung in wissenschaftliches Programmieren
2. Einbindung Cody Coursework
3. Nützliche Techniken zur Aufgabenerstellung
4. Nachklang

Veranstaltung EWP

Auszug Modulhandbuch Bachelor Mathematik - Block Modellierung:

"**Lernergebnisse / Kompetenzen:**
Die Studierenden sind in der Lage, selbstständig Teilaspekte exemplarischer Anwendungsprobleme aus Industrie und Wirtschaft zu behandeln; dies betrifft insbesondere die Wahl des mathematischen Modells, die Wahl geeigneter Lösungsverfahren sowie die Interpretation der Ergebnisse.
Durch die Teilnahme am Programmierkurs wurden die Studierenden mit einer Programmiersprache, grundlegenden Programmiertechniken und Datenstrukturen vertraut gemacht."

- Pflichveranstaltung im Bachelor Mathematik und Lehramt Mathematik
- Viele Anfänger
- Steigende Beliebtheit bei Studierenden anderer Fachbereiche

Vorlesungsmodus früher

Vorlesung (Hörsaal, Vortrag)

Folien
Beispiele
Script

Übung (Computerraum, Kleingruppen)

Übung 1 | Übung 2 | Übung 3

Übungsblätter
Fragen

Inhalte:

- 50% MATLAB
- 50% C

Scheinerwerb:

- Übungspunkte: Testatzulassung
- 2 Testate
  $\rightarrow$ Schein

Vorlesungsmodus jetzt

Vorlesung (Computerraum, interaktiv)

VL 1 | VL 2 | VL 3

Folien
Beispiele
Script

CODY COURSEWORK

Übung (Computerraum, Kleingruppen)

Übung 1 | Übung 2 | Übung 3

Übungsblätter
Fragen

Inhalte:

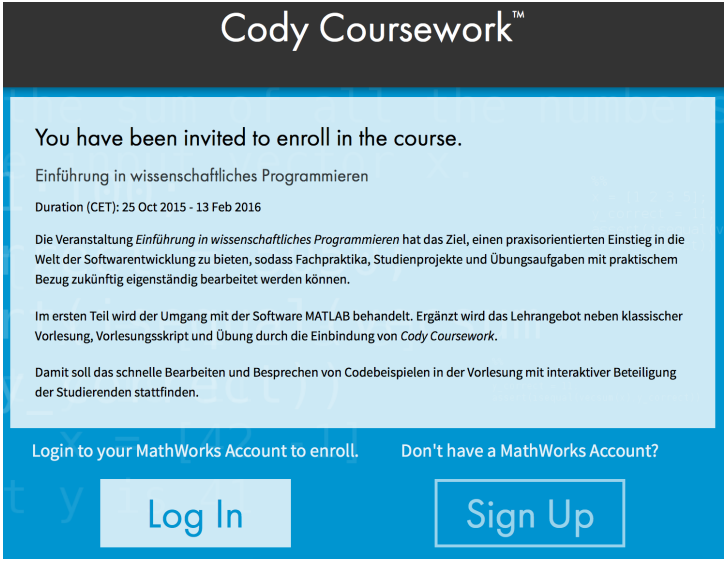
- 50% MATLAB
- 50% C

Scheinerwerb:

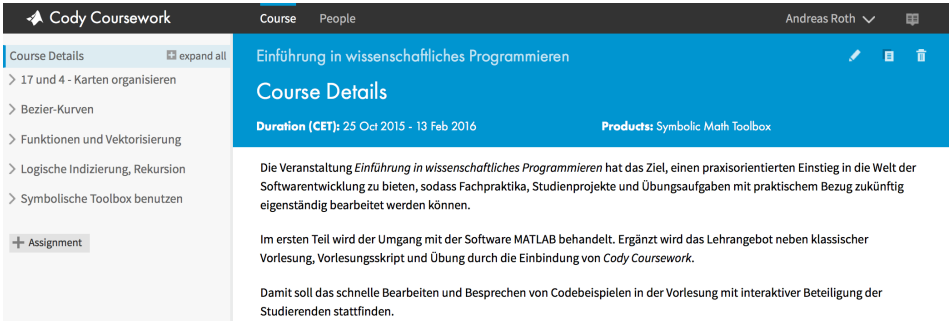
- Übungspunkte: Testatzulassung
- 2 Testate
  $\rightarrow$ Schein

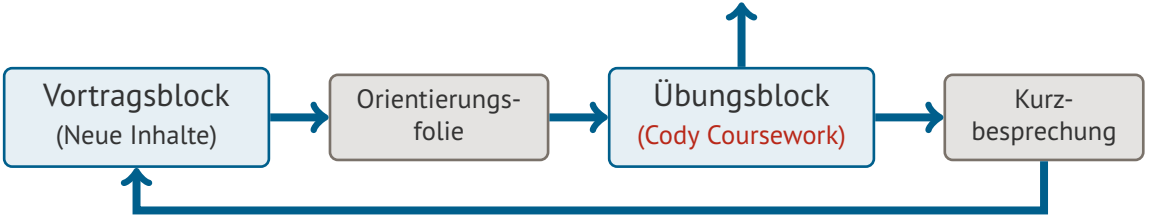
Einbindung Cody Coursework:

- Einladung per Mail
- Einschreibung in Kurs mit MathWorks-Account
- Keine Lizenznummer für Kursteilnehmer notwendig



Vorlesung: Ablauf





Vortragsblock:
Vektorisierung

In MATLAB gilt: Versuche, so viele Schleifen wie möglich durch Vektor-Matrix-Operationen zu ersetzen.

So nicht:

```
f=@(x) (x^2-1)/(x^2+1);
tic;
sum_Slow=0;
for k=1:n
  y(k)=f(x(k));
  sum_Slow=sum_Slow+...
      y(k)*hx;
end
time_Slow=toc;
```

Ergebnis:

```
>>Zeit:0.0080,
  sum_Slow=6.5939
```

Besser so:

```
fv=@(x) (x.^2-1)./...
    (x.^2+1);
tic;
yv=fv(x);
sum_Fast=sum(yv*hx);
time_Fast=toc;
```

Ergebnis:

```
>>Zeit:0.0027,
  sum_Fast=6.5939
```

Orientierungsfolie:

Ressourcen und Übungen

Weiterlesen...

- VL-Script 2.3.1. Komponentenweise Ausführung
- VL-Script 2.5.1. Scripts, Listing 2.5.1
- VL-Script 2.8.1. Zeitmessung

Cody Coursework Kurs zu EWP

- Assignment 3: Funktionen und Vektorisierung
  - Problem 2: Distanzmatrix berechnen

Codebeispiele auf der Vorlesungshomepage:

Abschnitt Downloads, VL03_Code.zip:

- `function_eval.m`
- Lösung der Übung: `distmat.m`

Übungsblock:
Distanzmatrix

Distanzmatrix berechnen

Problem Description

Seien *x_i, i=1,2,...,n* die Positionen von Partikeln in 2D. Berechnen Sie die Matrix *D*, deren Einträge *D_ij* gegeben sind durch den Abstand der Partikel *i* und *j*.

Zu diesem Zweck soll eine Funktion

```
D=distmat(X)
```

geschrieben werden, welche als Input eine *2xn* matrix *X* erhält, welche spaltenweise die Punktkoordinaten *x_i, i=1,2,...,n* enthält. Der Rückgabewert soll *D* sein, die *nxn* Distanzmatrix, welche die Abstände der Spalten *i* und *j* von `X` in ihrem *ij*-ten Eintrag enthält.

Hinweise:

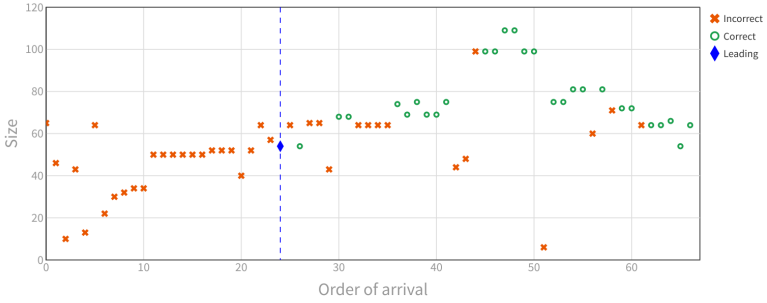
- Als Distanzmaß soll die **Euklidische Distanz** (https://de.wikipedia.org/wiki/Euklidischer_Abstand) benutzt werden.
- Es ist möglich, dieses Problem mit Schleifen zu lösen. Die besondere Herausforderung besteht jedoch darin, auf die Benutzung von Schleifen weitgehend zu verzichten und auf Vektoroperationen zu setzen.
- Die Funktion `repmat` kann unter Umständen hilfreich sein.
- Benutzen Sie für diese Übung nicht die MATLAB-eigene Funktion `pdist()`

Solve

Kurzbesprechung:

Abgaben sofort für Dozenten einsehbar

- Bewertung mit *Size*
- Falsch oder richtig gemäß *Tests*
- Verbesserung direkt vorführbar



Submitted on 18 Nov 2015 by Size 54 Test Suite Results

Solution 1214246

```
function D=distmat(X)

A=repmat(X(1,:),size(X,2),1);
B=repmat(X(2,:),size(X,2),1);

D=sqrt((A-A').^2+(B-B').^2)

end
```

Nützliche Techniken zur Aufgabenerstellung

Benutzerordner:

```
./+reference/solution.p
```

```
./solution.m
```

Test Suite:

```
%% Test 1
... some code
%% Test 2
... some code
```

Testen und Bewerten:

- Aufruf Referenzlösung: `reference.solution(...)`
- Aufruf Lösung: `solution(...)`
- Outputs / Grafische Ausgabe vergleichen (`assert()`)
- Quelltext überprüfen

Beispiel: Distanzmatrix

Referenzlösung:

```
function D=distmat(X)
n=size(X,2); % get nbr of points
% compute distances
dx=repmat(X(1,:),[n,1])...
    -repmat(X(1,:)',[1,n]);
dy=repmat(X(2,:),[n,1])...
    -repmat(X(2,:)',[1,n]);
% Euclidian norm
D=sqrt(dx.^2+dy.^2);
end
```

Lösungsschablone:

```
function D=distmat(X)
% D = ...
end
```

Möglicher Test:

```
%% Test 1
X=rand(2,100);
% Your solution
D=distmat(X);
% Reference solution
D_c=reference.distmat(X);
eps=1e-15;
assert(max(max(abs(D-D_c)))<eps);
```

Beispiel: Bisektionsverfahren

```
function out=bisektion(a,b,f,eps)
% Bisektionsverfahren rekursiv
if f(a)*f(b)>0
    error('Kein Vorzeichenwechsel.');
else
    c=0.5*(a+b);
    if abs(a-b)/2<eps||f(c)==0
        out=c;
    elseif f(c)*f(b)<0
        out=bisektionRek(c,b,f,eps);
    else
        out=bisektionRek(a,c,f,eps);
    end
end
```

Aufgabe:

Finde Lösung der Gleichung

$$0 = 1 - x - sin(x) =: f(x)$$

im Intervall $[a, b]$.

- Nullstellensuche
- Übung zu Rekursion

Beispiel: Bisektionsverfahren

Mögliche Tests (u.a.):

```
%% Fehlermeldung korrekt ?
f=@(x) x.^2;
fehler=false;
try
    erg=bisektion(1,2,f,1e-6);
catch ME
    if strcmp(ME.message,...
        'Kein Vorzeichenwechsel.')
        fehler=true;
    end
end
assert(fehler,...
    'Falsche Meldung!');
```

```
%% Rekursion benutzt ?
name='bisektion.m';
F=fileread(name);
% Es wurde eine Schleife benutzt
wh=strfind(F,'while');
fo=strfind(F,'for');
assert(isempty([wh,fo]),...
    'Schleife benutzt!');
% Selbstaufruf findet nicht statt
anzAufrufe=length(strfind(F,...
    name(1:end-2)));
assert(anzAufrufe>1,...
    'Kein Selbstaufruf!');
```

Referenzlösungen gestalten

Was ich will:

- Code wiederverwenden
- Wenig Quelltext in Tests
- Teillösungen zur Nutzung bereitstellen

Was ich nicht kann:

- Zusätzliche Dateien hochladen
- Teillösungen im Klartext in Solution Template schreiben

Was möglich ist:

1. Variables Verhalten der Referenzlösung: `varargin` und `varargout` mit `inputParser`
2. Handles auf Subfunctions / Nested Functions zurückgeben
3. Teillösungen als String exportieren, `pcode()`

Beispiel: Coulomb-Kräfte in einer Punktewolke

Aufgabe:

Für $x_i \in \mathbb{R}^2, i = 1, 2, \ldots, n$ berechne:

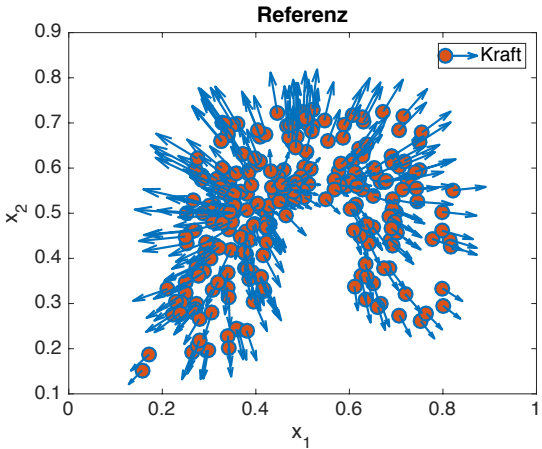
$$F_i = \sum_{j=1}^{n} \nabla_x U(|x_i - x_j|)$$

$$U(r) = -\frac{1}{r}$$

$$\Rightarrow F_i = \sum_{j=1}^{n} \frac{x_i - x_j}{|x_i - x_j|^3}$$

Implementierung in:

```
function F=coulomb(X,minRad)
```



Referenzlösung:

```
function varargout=coulomb(varargin)
p=inputParser();
p.addParameter('returnHandle',[]);
```

mit Subfunktionen:

```
function F=coulomb_c(X,minRad)
function [D,dist]=distmat_c(X)
function plotField(fg,X,F,ti)
function X=generatePoints(test,n)
function write2file(fname)
```

- Erste zwei Argumente mit richtiger Gestalt: Berechnung gemäß Definition
- Restliche Inputs: `p.parse()`
- Wert von `returnHandle` steuert Rückgabe eines *Function Handles* auf Subfunktion
- Handles können in Tests benutzt werden

Funktionen exportieren mit `f2str_mod()`

Vor dem Hochladen auf Referenz anwenden:

```
f2str_mod('fun_file.m',{'fun1','fun2',...},'writefun');
```

fun_file.m

```
function fun1()
% ...

function fun2()
% ...
```

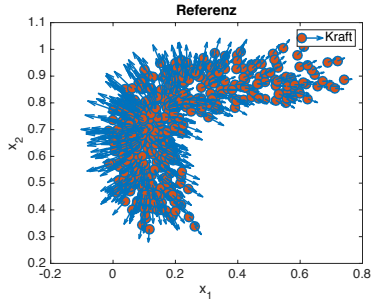
erzeugt

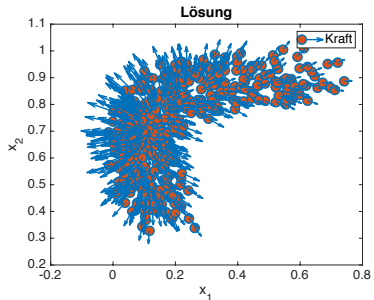
fun_file_mod.m, wie **fun_file.m**, aber mit

```
function writefun(fname)
switch fname
case 'fun1'
    F=fopen('fun1.m','w');
    fprintf(F,'function fun1()\n');
    fprintf(F,'%% ...\n');
    fprintf(F,'\n');
    fclose(F);
    pcode('fun1.m','-inplace');
    delete('fun1.m');
case 'fun2'  % ...
```

Ein möglicher Test:

```
%% Punkte generieren
genPoints=reference.coulomb(...
    'returnHandle','generatePoints');
X=genPoints('band',500);
% Hilfsfunktion exportieren
write2file=reference.coulomb(...
    'returnHandle','write2file');
write2file('distmat_c');
% Auswerten und plotten
[F_c,plotField]=reference.coulomb(...
    X,0.1,'returnHandle','plotField');
F=coulomb(X,0.1);
plotField(1,X,F_c,'Referenz');
plotField(2,X,F,'Loesung');
eps=1e-15;
assert(max(max(abs(F_c-F)))<eps);
```





Kommentare aus der Vorlesungsumfrage

- Die Vorlesung in den Computerräumen zu halten sehr sinnvoll, programmieren lernt man viel besser, wenn man Sachen auch ausprobieren kann.
- Matlab: positiv: Cody Coursework von Matlab und Matlab Academy.
- Daumen hoch! Nutzung von Mathworks-Umgebung (Cody Course- work) sehr gut!
- Neues Konzept viel besser: durch integrative Aufgaben wird Verständnis zu einzelnen Themen erheblich gefördert.

Aufgaben ausprobieren?
Quelltext beziehen?

Mail an roth@mathematik.uni-kl.de