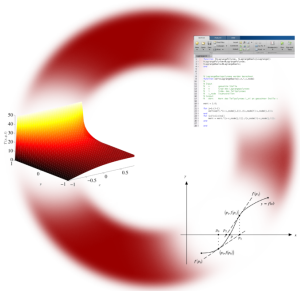
Praktikum Numerische Methoden für Ingenieure

Einführung, Matlab, Grundfunktionen

Christoph Ager

TUM – Lehrstuhl für Numerische Mechanik

October 17, 2016



Kurzer Überblick

- Interpolation und Approximation
- Numerische Differentiation und Integration
- Numerische Lösung von Anfangswertproblemen
- Methode der Finiten Elemente
- Approximation von Eigenwerten
- Lösung von linearen Gleichungssystemen: direkt und iterativ
- Lösung von nichtlinearen Gleichungssystemen

Praktikum

Ausgewählte numerische Methoden eigenständig umsetzen:

- Besseres Verständnis der Methoden
- Erkennen von Vor- & Nachteilen
- Verbesserung der Programmierkenntnisse
- Selbständiges Anwenden auf weitere Problemstellungen
- ...

Termine

- Präsentation der neuen Aufgabenstellung (Anwesenheitspflicht!):
  Donnerstags 17:00 - 17:45, MW 0350 - siehe TUMonline (teilweise nur jede 2. Woche)
  aktuelle Termine siehe TUMonline, Ende der Präsentation
- selbstständige Bearbeitung der Aufgabenstellung in der folgenden Woche
- Tutorensprechstunde:
  Montags 16:00-18:15, 1264 - Computer-Red-Pool
  Mittwochs 15:30-17:45, 1264 - Computer-Red-Pool
- 2x im Semester: Überprüfung der selbstständigen Bearbeitung der Aufgabenstellungen
  Vorraussichtliche Termine:
  (Do 1.12.2016, Mo 05.12.2016) und
  (Mo 23.01.2016, Mi 25.01.2016)

Matlab

Die Bearbeitung der Aufgabenstellungen erfolgt mithilfe von Matlab!

- Matlab Lizenz: Lizenzverteilung MathWorks-Rahmenvertrag: https://matlab.rbg.tum.de/
- Verwendung von Matlab am eigenen Notebook, ansonsten im Red-Pool des LNM

Aufgabenblätter

Es werden wöchentlich neue Aufgabenblätter ausgegeben, die eigenständig bearbeitet werden sollen!

- Beinhalten mehrere Aufgaben die zu Bearbeiten sind!
- Enhalten bereits Lösungen (Zahlenwerte oder Diagramme) um den erstellten Code zu überprüfen!
- Vorgegebene durchnummerierte Funktionen welche genau so erstellt werden sollen! (Um diese in späteren Aufgabenblättern wiederzuverwenden)
  $\Rightarrow$ **Matlab Funktionen**
  Diese sollen automatisch auf das vorgegebne Ergebnis geprüft werden (1. Aufgabenblatt)
- Lösungscodes werden nicht zur Verfügung gestellt!
- Die Aufgabenblätter sind jeweils auf Moodle zu finden!

Hilfe!

- Matlab-Hilfe: http://www.mathworks.de/de/help/matlab/
- Matlab Tutorial
- Befehl 'help' 'functionname' in Matlab Konsole
- Online Suchmaschine
- Tutorsprechstunden - vorbereitet in die Tutorsprechstunden kommen:
  (Fragen des aktuellen Arbeitsblattes!):
  - Arbeitsblatt gelesen
  - Aufgaben in Matlab bearbeitet
  - Falsches Ergebnis
  - selbstständige Fehlersuche (Eingrenzen des Fehlerbereiches)
  - alle Funktionen mit Modultest getestet

Überprüfungen

- 2x im Semester:
  (1.: Arbeitsblatt 1 - 6, 2.: Arbeitsblatt 7 - 12)
- jeweils zwei Termine
- eintragen in Listen am Do., ca. 1 Monat vor der Überprüfung
- Matlab-Code zum lösen der Aufgabenstellungen an den Computern im Red Pool
- Zum lösen der Aufgabenstellung sollten alle Arbeitsblätter selbstständig bearbeitet werden

Überprüfungsablauf - Als Vorinformation

- Bearbeitung an Rechnern des Red Pools
- **Matlab Funktionen** werden je nach Aufgabenstellung ohne Beschreibung zur Verfügung gestellt
- Alle Unterlagen erlaubt, außer:
  keine Matlab Codes (Matlab Hilfe ausgenommen) in jeglicher Form
  keine Pseudo Matlab Codes wie ('Wenn x ist y dann ...')
- keine externen Datenträger, keine e-mails
- Internet zum 'googeln', 'Matlab Hilfe' steht zur Verfügung

Fragen?

Fragen zum Ablauf?

Praktikum Numerische Methoden für Ingenieure

Aufgabenblatt 1

WS 2016/2017

Übungsblatt Matlab, Grundfunktionen für das Praktikum

Aufgabe 1: Vektoren, Matrizen

Erstelle ein Matlab Skript, das folgende Operationen durchführt:

- $\boldsymbol{A} = \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix}$
- $\boldsymbol{v} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$
- $a = \boldsymbol{v} \cdot \boldsymbol{v}$ $(Lsg. : 30.0)$

- $\boldsymbol{B} = \boldsymbol{v}\boldsymbol{v}^T$ $(Lsg. : [1, 2, 3, 4; 2, 4, 6, 8; 3, 6, 9, 12; 4, 8, 12, 16])$

- $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}$ $(Lsg. : [3, 6, 9, 12; 6, 12, 18, 24; 9, 18, 27, 36; 12, 24, 36, 48])$

- Berechne die Eigenwerte $\boldsymbol{\lambda}$ der Matrix $\boldsymbol{C}$ (nutze die vorhandene Matlab Funktion).
  Entspricht diese Lösung der exakten analytischen Lösung? Stelle dazu qualitative Überlegungen an.
  $(Lsg. : [-0.000000000000002; 0.0; 0.000000000000007; 90.0])$

- Löse das Gleichungssystem nach $\boldsymbol{x}$: $(\boldsymbol{C} - \boldsymbol{A})\boldsymbol{x} = a\boldsymbol{v}$ (nutze die vorhandene Matlab Funktion).
  $(Lsg. : [0.344827586206896; 0.689655172413793; 1.034482758620690; 1.379310344827587])$

- Weise der zweiten Spalte der Matrix $\boldsymbol{C}$ den Vektor $\boldsymbol{v}$ zu.
  $(Lsg. : [3, 1, 9, 12; 6, 2, 18, 24; 9, 3, 27, 36; 12, 4, 36, 48])$

- Speichere die dritte Zeile der Matrix $\boldsymbol{C}$ in den Spaltenvektor $\boldsymbol{b}$. $(Lsg. : [9, 3, 27, 36])$

- Erzeuge einen Zeilenvektor $\boldsymbol{c}$ mit Einträgen von 10.0 bis 100.0 und einer Schritteweite von 0.5.

- Erzeuge einen Zeilenvektor $\boldsymbol{f}$ mit Einträgen $c_i(5 + c_i) + 1/c_i + 2^{c_i}$ (c_i sind die Elemente des Vektors $\boldsymbol{c}$).

- Ermittle die Dimension l_f des Zeilenvektors $\boldsymbol{f}$. $(Lsg. : 181)$

Aufgabe 2: Operatoren und Flusskontrolle, Funktionen

Erstelle ein Matlab Skript, das folgende Operationen durchführt:

- Weise $\boldsymbol{a}$ einen Zufallsvektor mit Einträgen zwischen 0 und 1 der Dimension 1000 zu.
- Wenn der erste Eintrag von $a_1 \geq 0.5 \rightarrow$ Matlabausgabe 'a1 >= 0.5', sonst 'a1 $<$ 0.5' (HINWEIS: `if`).
- Ermittle die Anzahl n_{05} der Einträge $a_i \geq 0.5$ im Vektor $\boldsymbol{a}$ (HINWEIS: `for,if`).
- Ermittle den ersten Eintrag (Index i & Wert a_i) im Vektor $\boldsymbol{a}$ für den gilt $0.499 \leq a_i \leq 0.501$. Falls kein Element existiert, welcher das Kriterium erfüllt $\rightarrow$ Matlabausgabe 'Kein Element 0.499 <= a_i <= 0.501' (HINWEIS: `while,if`).
- Berechne den Wert von $n!$ für ganzzahlige n. Erstelle dazu eine eigene Funktion **Fkt. A**(siehe unten). Die Funktion soll in einem eigenen *.m-file abgespeichert werden. Teste die Funktion für $n = 0$ und $n = 5$.

Aufgabe 3: Plots

Erstelle ein Matlab Skript, das folgende Plots erstellt (HINWEIS: Nutze die Matlab Hilfe):

Werte dazu die Funktion an den gewünschten Punkten aus und plotte diese (kein direktes plotten analytischer Funktionen).

- 2d–Plot: Gegeben ist die Funktion $f(x) = \sin(x)$. Plotte die Funktion im Interval $-\pi \leq x \leq \pi$ mit geeigneten Einstellungen.

- 3d–Quadplot: Erstelle ein Programm um beliebige 3d–Flächen im Raum, welche aus einer beliebigen Anzahl an Vierecken besteht, zu plotten. Als erster Schritte soll dazu die allgemeine Funktion **Fkt. 0** zum Plotten von beliebigen Vierecken erstellt werden. Diese soll alle Vierecke in jeweils zwei Dreiecke unterteilen und dann diese mithilfe der Matlab-Plot Funktion `trisurf` plotten. Als zweiter Schritt soll die Funktion verwendet werden um folgende 3d–Fläche zu plotten:

  Gegeben ist ein Gitter bestehend aus 4 Vierecken mit den Eckkoordinaten $x = -1, 0, 1$ und $y = -1, 0, 1$. Die z-Koordinate in den Ecken der Vierecke ist durch die Funktion $f(x, y) = x^2 + y^2$ gegeben.

Beschrifte alle Achsen und lege einen Diagrammtitel fest.

Bearbeitung des Aufgabenblattes

Erstellen eines *.m-files: LAB1.m:

```
%Funktion die alle Aufgaben des ersten Arbeitsblattes berechnet!
function LAB1

    str = strrep(pwd, 'Lab1/', ''); %Benutze die Tatsache das  beide Ordner m-files heißen!
    addpath(str); %Füge den Pfad vordefinierten Funktionen hinzu!

    clc;clear all;close all;
    format long;

    Aufgabe1(); %Berechnungen für Aufgabe 2
    Aufgabe2(); %Berechnungen für Aufgabe 2
    Aufgabe3(); %Berechnungen für Aufgabe 3

    mtest(); %Modultest für Aufgabe 4 durchführen (liegt in file mtest.m)

end

%Berechnet alle Aufgabenstellungen aus Aufgabe 1
function Aufgabe1()
    ...
end
```

Bzw.: Erstellen eines eigenen *.m-files für jede Aufgabe.

Bearbeitung des Aufgabenbl. - Ordnerstruktur Empfehlung

- Erstellen eines Ordners für jedes Aufgabenblatt, darin enthalten sind die *.m-files dieses Arbeitsblattes
  (z.B.: '/NumiP/LAB1/m-files/LAB1.m')
- Erstellen eines Ordners für das gesamte Praktikum, darin enthalten sind die *.m-files der **Matlab Funktionen**
  (z.B.: '/NumiP/m-files/facultaet.m')

Aufgabe 4: Modultests

Erstelle ein Matlab Skript (und speichere dies in einem eigenen *.m-file ab), das den Rückgabewert von Funktionen mit einem vorgegebenen Ergebnis auf einen maximalen absoluten festgelegten Fehler (Toleranz) prüft.

Das Skript sollte ausgeben, welche Funktionen den Test bestehen bzw. welche nicht. Trage als erste Funktion `facultaet(n)` in das Skript ein und teste das Ergebnis für $n = 0$ und $n = 5$ mit einer Toleranz von 10^{-12}.

HINWEIS: Dieses Skript soll im Laufe des Praktikums erweitert werden, sodass ALLE Funktionen mit Rückgabewert, die während des Praktikums erstellt werden, geprüft werden können (Ergebnisse können Skalare, Vektoren oder Matrizen sein). Die genaue Gestaltung des Skripts ist dir selbst überlassen.

Matlab Funktionen:

Folgende Funktionen sollen bei der Bearbeitung dieses Aufgabenblattes erstellt werden, da diese für spätere Aufgabenblätter wiederverwendet werden sollen. Erstellen Sie die Funktionen in Matlab und speichern Sie diese in eigenen *.m-files ab.

- **Fkt. A**: `function nfac = facultaet(n)`

  Rückgabewert: Fakultät der Zahl n.

  Teste die Funktion mit:

Modultest

Command Window

```
##########################################################################################################
############################################## Modultest ##################################################
001:     Testing function <Fkt -I: facultaet_0> ... passed (= 1, tol = 1e-12)!
002:     Testing function <Fkt -I: facultaet_5> ... passed (= 120, tol = 1e-12)!
003: !!! Testing function <Fkt -I: facultaet_0,1,2,3> ... failed ([1 1;2 6] ~= [1.01 1.02;2.0001 6], tol = 1
004:     Testing function <Fkt I: linquadref(0,0)> ... passed (= [0.25;0.25;0.25;0.25], tol = 1e-12)!
005:     Testing function <Fkt I: linquadref(0.577,-0.577)> ... passed (= [0.16676775;0.62173225;0.16676775;
006:     Testing function <Fkt II: linquadderivref(0,0)> ... passed (= [-0.25 -0.25;0.25 -0.25;0.25 0.25;-0.
007:     Testing function <Fkt II: linquadderivref(0.577,-0.577)> ... passed (= [-0.39425 -0.10575;0.39425 -
008:     Testing function <Fkt III: gx(3)> ... passed (= [-0.774596669241483 0 0.774596669241483], tol = 1e-
009:     Testing function <Fkt IV: gw(3)> ... passed (= [0.555555555555556 0.888888888888889 0.5555555555555
010:     Testing function <Fkt V: gx2dref(1)> ... passed (= [0 0], tol = 1e-12)!
011:     Testing function <Fkt V: gx2dref(2)> ... passed (= [-0.577350269189626 -0.577350269189626;-0.57735(
012:     Testing function <Fkt VI: gw2dref(1)> ... passed (= 4, tol = 1e-12)!
013:     Testing function <Fkt VI: gw2dref(2)> ... passed (= [1;1;1;1], tol = 1e-12)!
014:     Testing function <Fkt VII: getxPos([2,1;4,1;4,3;2,2],0.577,-0.577)> ... passed (= [3.577;1.37826775
015:     Testing function <Fkt VIII: getJacobian([2,1;4,1;4,3;2,2],0.577,-0.577), J> ... passed (= [1 0;0.1(
016:     Testing function <Fkt VIII: getJacobian([2,1;4,1;4,3;2,2],0.577,-0.577), detJ> ... passed (= 0.8942
017:     Testing function <Fkt VIII: getJacobian([2,1;4,1;4,3;2,2],0.577,-0.577), testinvJ> ... passed (= [1
018:     Testing function <Fkt IX: OST(0.5,0.2,1.1,[1.4,1.5],[1.7,1.8],[2.0]), LHS> ... passed (= 0.96, tol
019:     Testing function <Fkt IX: OST(0.5,0.2,1.1,[1.4,1.5],[1.7,1.8],[2.0]), RHS> ... passed (= 2.85, tol
020:     Testing function <Fkt X: AB2(0.2,1.1,[1.5,1.6],[1.8,1.9],[2.0,2.1]), LHS> ... passed (= 1.1, tol =
021:     Testing function <Fkt X: AB2(0.2,1.1,[1.5,1.6],[1.8,1.9],[2.0,2.1]), RHS> ... passed (= 3.114, tol
022:     Testing function <Fkt XI: AM3(0.2,1.1,[1.4,1.5,1.6],[1.7,1.8,1.9],[2.0,2.1]), LHS> ... passed (= 0.
023:     Testing function <Fkt XI: AM3(0.2,1.1,[1.4,1.5,1.6],[1.7,1.8,1.9],[2.0,2.1]), RHS> ... passed (= 2.
024:     Testing function <Fkt XII: BDF2(0.2,1.1,1.4,1.7,[2.0,2.1]), LHS> ... passed (= 1.37, tol = 1e-12)!
025:     Testing function <Fkt XII: BDF2(0.2,1.1,1.4,1.7,[2.0,2.1]), RHS> ... passed (= 3.585, tol = 1e-12)!
026:     Testing function <Fkt XIV: solveGauss([10.0,2,1;3,4,4;1,8,4],[1;1;2])> ... passed (= [0.05128205128
027:     Testing function <Fkt XV: solveG([10.0,2.0,10.0;2.0,40.0,8.0;10.0,8.0,60.0],[1.0;1.0;2.0],[0.0;0.0;
028:     Testing function <Fkt XVI: solveCG([10.0,2.0,10.0;2.0,40.0,8.0;10.0,8.0,60.0],[1.0;1.0;2.0],[0.0;0.
----------------------------------------------------------------------------------------------------------
1 / 28 tests failed!
##########################################################################################################
```

- **Fkt. A**: `function  nfac = facultaet(n)`

  Rückgabewert: Fakultät der Zahl n.

  Teste die Funktion mit:

$$(n = 0) \to 1, \quad (n = 5) \to 120$$

- **Fkt. 0**: `function quadplot(nodes,elements,sol)`

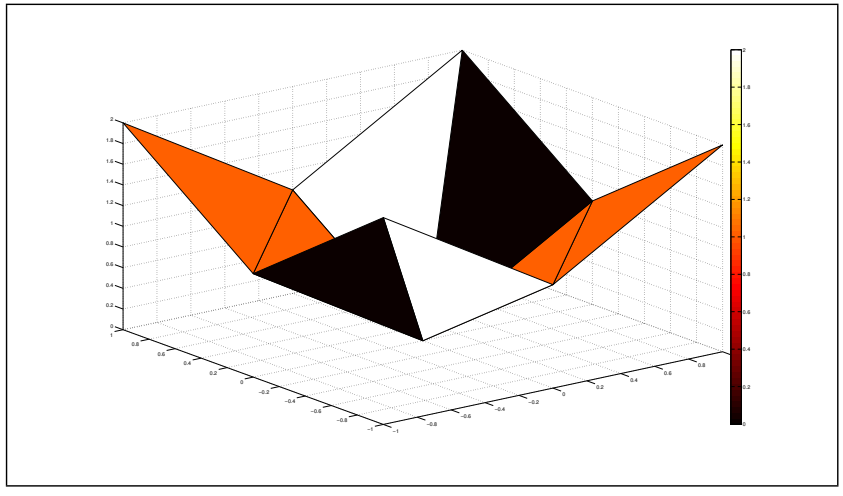
  `nodes`... [Knotenkoordinaten (Knotenid, (x=1,y=2)-Richtung)],

  `elements` ... [Knotenid (Elementid, lokale Knotenid)],

  `sol` ... [Lösungsvektor (Knotenid)]

  Teste die Funktion mit:

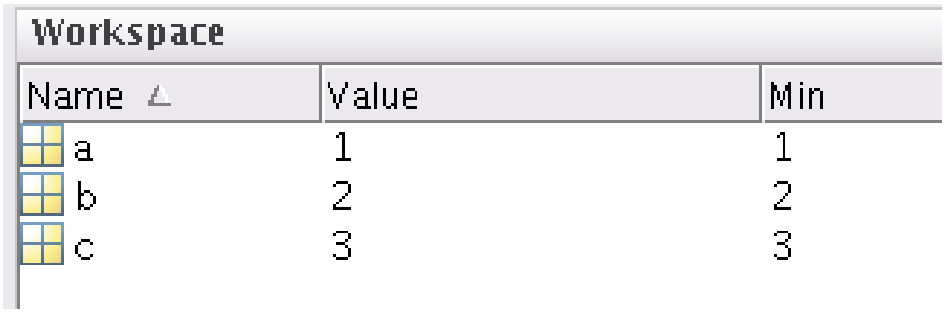
$$(nodes = [-1,-1;0,-1;1,-1;-1,0;0,0;1,0;-1,1;0,1;1,1],$$
$$elements = [1,2,5,4;2,3,6,5;4,5,8,7;5,6,9,8], sol = [2;1;2;1;0;1;2;1;2])$$

$\to$ 

Debugger

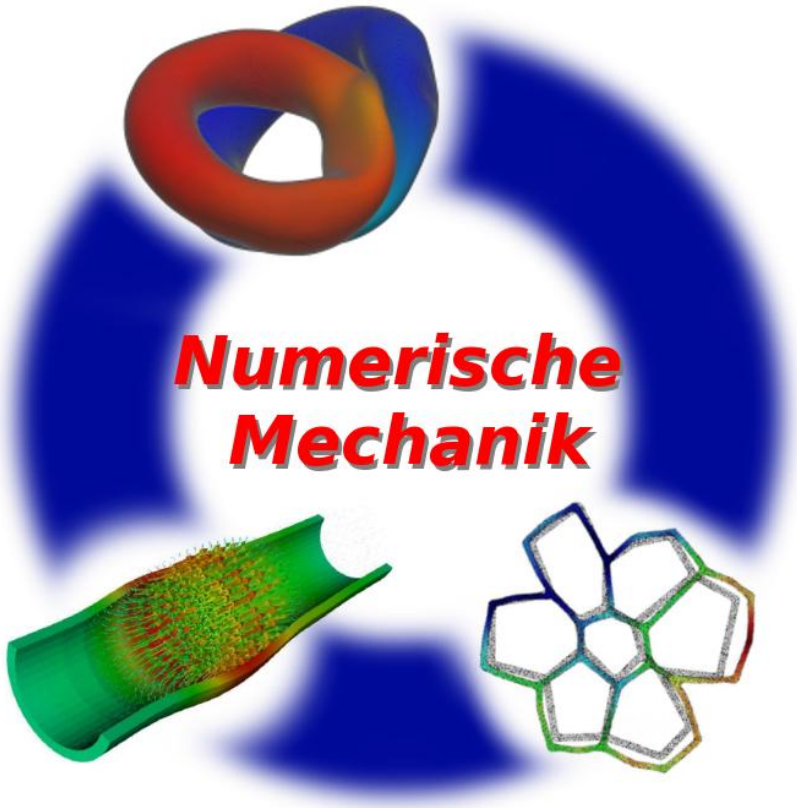
Breakpoints setzen:

```
26        %Berechnet alle Aufgabenstellungen aus Aufgabe 1
27      function Aufgabe1()
28 -        a = 1;
29 -        b = 2;
30 -        c = a + b;
31 ●➡       d = a*b;
32 -    end
```



Allgemeines

Info-Veranstaltung Studienschwerpunkt Numerische Mechanik



- Infos zu allen Lehrveranstaltungen des LNM
- Heute, 18:00 Uhr, MW 1237 (Gebäude 2, 1. Stock)
- weiteres unter http://www.lnm.mw.tum.de/teaching/nummech/
- Alle sind herzlich eingeladen!

Und Los ...

Nächste Tutorsprechstunden:

Montag 24.10. 16:00-18:15, 1264 - Computer-Red-Pool
Mittwoch 26.10. 15:30-17:45, 1264 - Computer-Red-Pool

Nächstes Aufgabenblatt:

Donnerstag 27.10. 17:00-17:45, MW 0350