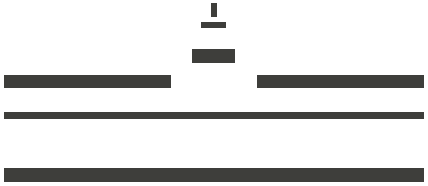
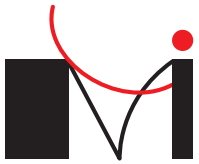


Westfälische Wilhelms-Universität Münster



Angewandte Mathematik Münster

Praktikum zur Modellreduktion

Wintersemester 2015/16

wissen.leben
WWU Münster

Julia Brunken

2. Februar 2016

Organisatorisches

- Anwesenheitsliste
- Anrechnung als Übung zur Vorlesung I im Spezialisierungsmodul Wissenschaftliches Rechnen
- Linux-Kenntnisse?
- Programmier-/ Python-Kenntnisse?
- Funktioniert der Login?

Inhalte

- Python-Grundkenntnisse, Numpy
- Das Modellreduktions-Framework pyMOR
- Standard-Lösungsverfahren (Finite Elemente)
- Reduzierte Basis Methoden

Literatur

Linux-Terminal:

- https://help.ubuntu.com/community/UsingTheTerminal
- http://community.linuxmint.com/tutorial/view/244

Python und Numpy:

- http://wwwmath.uni-muenster.de/num/Vorlesungen/Pythonkurs_SS15/Pythonkurs_komplett.pdf
- https://en.wikibooks.org/wiki/Non-Programmer's_Tutorial_for_Python
- http://docs.python.org/2/reference/
- http://docs.scipy.org/doc/numpy/reference/index.html

pyMOR:

- http://pymor.org
- https://github.com/pymor/pymor
- http://docs.pymor.org/en/0.3.x/

Einstieg in Python

Python-Programme werden üblicherweise mit Endung `.py` gespeichert.
Sie können dann über das Terminal ausgeführt werden.

Ausführung eines Programms im Terminal

- Terminal öffnen: `Strg + Alt + T`
- In das richtige Verzeichnis wechseln:
  `cd pythonordner/unterordner`
- Programm ausführen: `python programm.py`

Einstieg in Python

Beispiel: Hello world!

hello_world.py

```
print("Hello world!") # This is a comment
```

Ausführung der Datei im Terminal:

```
python hello_world.py
```

Ausgabe:

```
Hello world!
```

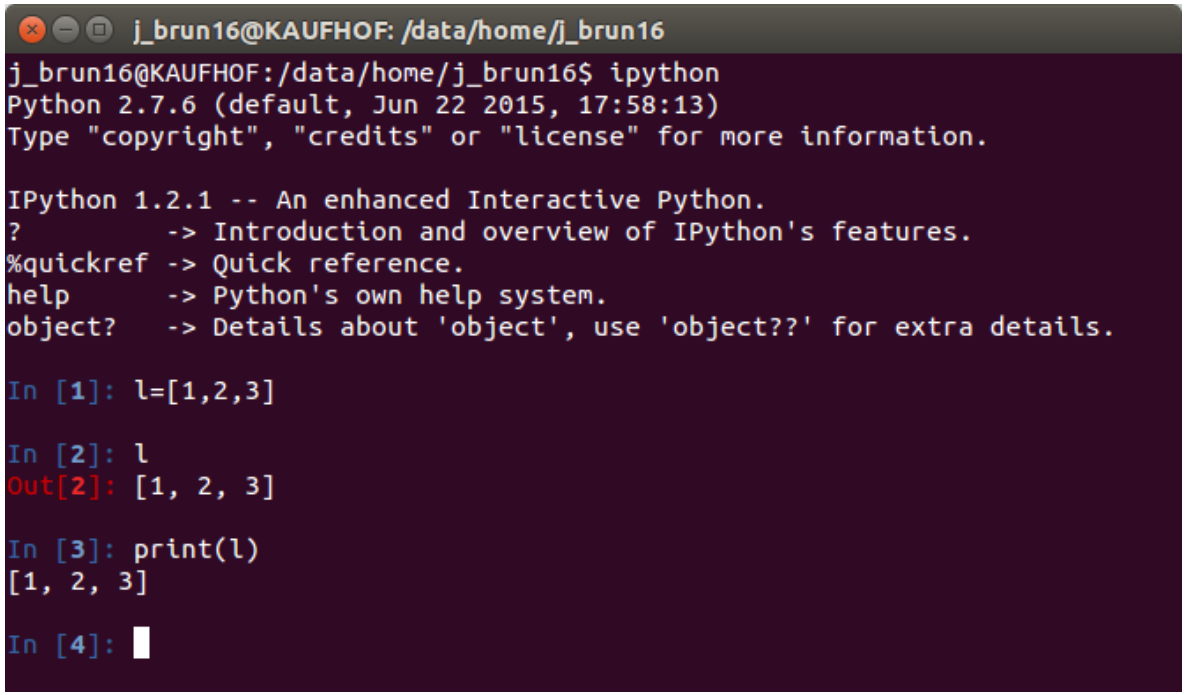
Einstieg in Python

Ein sehr nützliches Werkzeug bei der Entwicklung von Python-Programmen ist die IPython-Shell:

Die IPython-Shell

- Terminal öffnen: `Strg + Alt + T`
- IPython starten: `ipython`
- Python-Kommandos zeilenweise eingeben
- Nützlich: Info über ein Objekt mit `Objekt?`, `Objekt??`, `help(Objekt)`

IPython



NumPy

- Python-Modul für numerische Berechnungen
- Liefert Array-Objekte (verwendet für Vektoren, Matrizen, ...) und vieles mehr
- Einbindung in Python mit:

```
import numpy
```

- Verwendung:

```
A = numpy.array([1,2,3])
```

- Gebräuchliche Abkürzung:

```
import numpy as np
A = np.array([1,2,3])
```

Aufgabe 1

Probieren Sie in einer IPython-Shell wichtige Typen und Funktionen aus Python und NumPy aus, z.B.

- Python-Types: **list, tuple, dictionary, string**
- print(), range(), append(),
- NumPy: Matrizen und Vektoren als **np.array**
- Operationen und Funktionen für NumPy **arrays** wie
  - Aufruf von Komponenten / Teilmatrizen
  - Elementweise Operationen, Matrixmultiplikation
  - Informationsabruf
  - ...

NumPy

Übersicht wichtiger **array** Befehle

- **Erstellung**: array(), ones(), zeros(), diag(), eye(), empty(), arange(), linspace()
- **Manipulation**: transpose(), inv(),reshape(), ravel()
- **Information**: shape, ndim, dtype, itemsize, size, print, sum(), min(), max()
- **Operationen**: dot(), trace(), column_stack(), row_stack(), vstack(), hstack(), hsplit(), vsplit()

Achtung!

Befehl **array([1,2,3,4])** korrekt, array(1,2,3,4) erzeugt Fehler!

Aufgabe 2

Wiederholen/lernen Sie die Python-Syntax für

- **if/else**-Abfragen, **for**-, **while**-Schleifen, Boolean- und Vergleichsoperatoren
- Definition von *Funktionen*
- Konzept, Definition und Initialisierung von *Klassen*

Dazu lösen Sie wahlweise Übungsaufgaben vom Übungsblatt oder andere, selbst ausgesuchte, die die Konzepte beinhalten.

Werbung: Software-Tools-Seminar

Wann und Wo: Mittwoch, 04.11.2015, ab 14:00 Uhr, SRA

Themen:
- Versionskontrolle mit Git
- Wissenswertes zu Python
- Emacs als Entwicklungsumgebung
- Octave, eine freie Matlab-Alternative
- Webframeworks für Softwaretests
- Autotools

Infos: `http://wwwmath.uni-muenster.de/num/Vorlesungen/SoftwareToolSeminar_WS1516/`

PyCharm

Wir können die Python-Skripte im Terminal ausführen und in normalen Texteditoren bearbeiten. Um aber bei größeren Projekten eine Übersicht zu bekommen und auf einfache Art neue Skripte zu schreiben, verwenden wir die Python-IDE (integrierte Entwicklungsumgebung) *PyCharm*.

Ist PyCharm schon installiert??

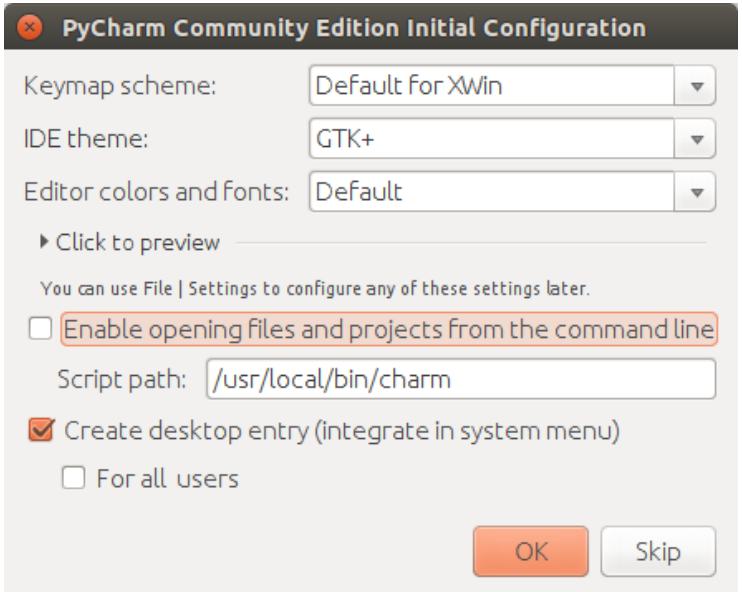
Wenn nicht:

PyCharm installieren

- Download
- Entpacken

PyCharm einrichten

Anfangseinstellungen:



Aufgabe 3

Öffnen Sie in PyCharm den Ordner mit den Python-Programmen vom Übungsblatt 1.
Machen Sie sich mit den Funktionen von PyCharm vertraut:

- Python-Programm starten: **Run**
- Debuggen: **Debug**
- Breakpoints setzen:

pyMOR

pyMOR ist eine Python-Library, die in der AG Ohlberger entwickelt wird.

Hauptfokus: Modellreduktion für parametrisierte PDGL mit der Reduzierten Basis Methode

Funktionen von pyMOR

- Standardlösungsverfahren für PDGL:
  - Finite Elemente Methode
  - Finite Volumen Methode
- Zeitschrittverfahren
- Reduktion der Gleichung
- Verschiedene Basisgenerierungs-Algorithmen

Virtual Environment

Für pyMOR werden einige nicht systemweit installierte Python-Pakete benötigt. An den Uni-PCs können wir neue Python-Pakete aber nicht einfach systemweit installieren. Wir benutzen deshalb eine *Virtual Environment*, eine abgeschlossene lokale Entwicklungsumgebung, in der wir dann frei Pakete lokal installieren können.

virtualenv einrichten

- Terminal öffnen, in gewünschten Ordner wechseln
- Virtual Environment erstellen:

```
virtualenv NameDerVirtualenv --system-site-packages
```

- Virtual Environment aktivieren:

```
source NameDerVirtualenv/bin/activate
```

Virtual Environment

Ist die virtualenv aktiviert, ist ihr Name in Klammern am Anfang der Kommandozeile zu sehen

Werden bei aktivierter virtualenv Python-Programme ausgeführt, werden die in der virtualenv installierten Pakete benutzt.

Neue Pakete werden installiert mit: `pip install Paketname`

pyMOR

pyMOR einrichten

- Terminal öffnen, in gewünschten Ordner wechseln
- pyMOR herunterladen mit:

```
git clone https://github.com/pymor/pymor.git
```

- In den Ordner `pymor` wechseln (`cd pymor`) und Folgendes ausführen:

```
python setup.py build_ext --inplace
```

- Zusätzliche Pakete installieren:

```
pip install pyside
pip install docopt
pip install pyopengl
```

pyMOR-Demo

Ein pyMOR Demo-Script ausführen

- In den Ordner `src` wechseln
- Ein Skript aus dem Ordner `pymordemos` ausführen mit:

  `python pymordemos/NAME SKRIPT-PARAMETER`
- Zum Beispiel: `python pymordemos/elliptic.py 1 0 0 0`

Die verschiedenen ausführbaren Dateien können mithilfe von `ls` oder mithilfe der `Tab`-Taste angezeigt werden

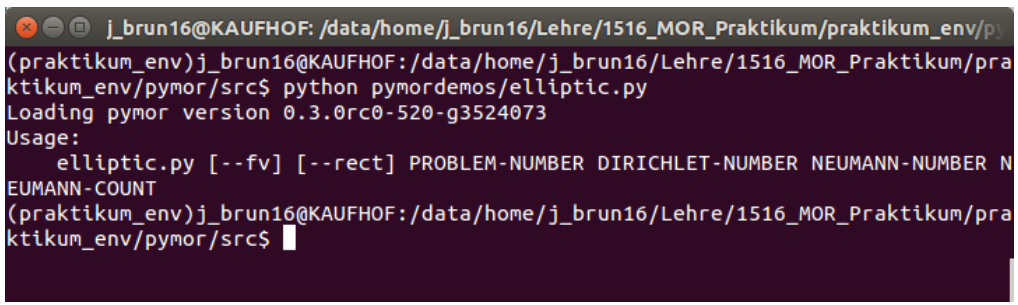
Achtung!

Die pyMOR-Skripte müssen immer aus dem `src`-Ordner heraus gestartet werden! (Alternativ kann `src` als Suchpfad gespeichert werden)

Skript-Parameter

Skript-Parameter werden beim Ausführen des Programms in der Kommandozeile hinter den Dateinamen eingetippt. Sie legen fest, mit welchen Optionen und Daten das Programm ausgeführt werden soll.

Welche Parameter benötigt werden, wird angezeigt, wenn das Programm ohne Parameter gestartet wird:



Parameter ohne Klammern müssen angegeben werden, Parameter in eckigen Klammern sind Optionen, die einfach hinzugefügt werden, z.B.

```
python pymordemos/elliptic.py --fv 0 0 1 0
```

Skript-Parameter

Eine ausführliche Beschreibung der Parameter wird beim Ausführen des Programms mit Option `-h` oder `-help` angezeigt:

```
j_brun16@KAUFHOF: /data/home/j_brun16/Lehre/1516_MOR_Praktikum/praktikum_env/py
(praktikum_env)j_brun16@KAUFHOF:/data/home/j_brun16/Lehre/1516_MOR_Praktikum/pra
ktikum_env/pymor/src$ python pymordemos/elliptic.py -h
Loading pymor version 0.3.0rc0-520-g3524073
Simple demonstration of solving the Poisson equation in 2D using pyMOR's builtin
 discretizations.

Usage:
    elliptic.py [--fv] [--rect] PROBLEM-NUMBER DIRICHLET-NUMBER NEUMANN-NUMBER N
EUMANN-COUNT

Arguments:
    PROBLEM-NUMBER    {0,1}, selects the problem to solve

    DIRICHLET-NUMBER  {0,1,2}, selects the Dirichlet data function

    NEUMANN-NUMBER    {0,1}, selects the Neumann data function

    NEUMANN-COUNT     0: no neumann boundary
                      1: right edge is neumann boundary
                      2: right+top edges are neumann boundary
                      3: right+top+bottom edges are neumann boundary

Options:
    -h, --help    Show this message.

    --fv          Use finite volume discretization instead of finite elements.

    --rect        Use RectGrid instead of TriaGrid.
(praktikum_env)j_brun16@KAUFHOF:/data/home/j_brun16/Lehre/1516_MOR_Praktikum/pra
ktikum_env/pymor/src$
```

Aufgabe 4

Starten Sie verschiedene pyMOR-Demo-Skripte mit unterschiedlichen Skript-Parametern aus dem Terminal.

PyCharm und pyMOR einrichten

Wir richten nun pyMOR in PyCharm ein:

pyMOR-Projekt erstellen

- Create New Project
- Location: `src`-Ordner aus pyMOR
- Interpreter: Einstellungen - Add Local

  `PfadZurVirtualenv/bin/python2.7`

Aufgabe 5: Finite Elemente Demos

Führen Sie die pyMOR-Demo-Skripte mit PyCharm aus.

- Skript-Parameter einstellen: **Run - Edit Configurations...**

Schauen Sie sich die Codes der Finite-Elemente Demos `elliptic.py` und `elliptic2.py` an.

- Welche Gleichungen werden gelöst?
- Welche Objekte werden erzeugt, welchen Zweck haben sie?

Hinweis: Benutzen Sie die Debug-Funktion und Breakpoints, um sich die verschiedenen Objekte genauer anzusehen.

Hilfe: Erste Hälfte von `http://pymor.readthedocs.org/en/0.3.x/getting_started.html#the-thermalblock-demo-explained`

(Fortgeschrittene) Hilfe: `http://pymor.readthedocs.org/en/0.3.x/technical_overview.html#technical-overview`

Aufgabe 6: Parametrisierte PDGL

- Wie wird der Parameter in `elliptic2.py` definiert?
- Schreiben Sie ein neues Skript als Erweiterung von `elliptic2.py`, in dem die Gleichung

$$-\nabla(\mu_l(1-x)+\mu_r x)\nabla u(x,y,\mu) = f(x,y)$$

  gelöst wird.
- Erweitern Sie Ihr Skript, sodass eine rechte Seite $f(x,y,\mu)$ wählbar ist, die ebenfalls von einem Parameter abhängt:

$$f(x,y,\mu) = \mu_f y$$

- Berechnen Sie jeweils Lösungen für verschiedene Kombinationen der zwei bzw. drei Parameter

Hinweis: Zum Bearbeiten der Skripte ist es praktischer, ohne Docopt zu arbeiten

Reduzierte Basis Methode

Wir wollen nun die verschiedenen Implementierungen der Reduzierten Basis Methode kennen lernen. Dazu benutzen wir das ThermalBlock-Problem.
Wir bauen Stück für Stück ein neues Skript auf, indem wir verschiedene Methoden ausprobieren können.

Aufgabe 7: ThermalBlock-Problem

Erstellen Sie ein neues Skript, indem das ThermalBlock-Problem für verschiedene Parameter gelöst und visualisiert wird.

Reduzierte Basis Methode: Themen

- Berechnen von reduzierten Lösungen
- Zusammensetzung der reduzierten Lösung aus den einzelnen Snapshots
- Fehlerabfall
- Vergleich verschiedener Snapshotmengen
- Vergleich verschiedener Fehlernormen

Reduzierte Basis Methode

Aufgabe 8: Reduzierte Basis aus Snapshots

- Berechnen Sie eine Menge aus Snapshots von
  - Von Ihnen gewählten Parametern
  - Uniform gewählten Parametern
  - Zufällig gewählten Parametern
- Reduzieren Sie mithilfe der Funktion `reduce_generic_rb` das volle Problem mit der berechneten reduzierten Basis und berechnen Sie reduzierte Lösungen

Fehlerberechnung

Wir wollen nun verschiedene Basen und verschiedene Basisgrößen vergleichen.
Welche Fehlernormen sind geeignet?

Aufgabe 9: Vergleich von Modellfehlern

- Berechnen Sie für ausgewählte Parameter μ_i die Fehler zwischen RB- und FEM-Lösung in einer geeigneten Norm.
- Vergleichen Sie die Fehler für verschieden große Snapshotmengen (Hinweis: `reduce_to_subbasis`)
- Plotten Sie Fehlerkurven, z.B. mit `pyplot`. Einfaches Beispiel:

```
import matplotlib.pyplot as plt
plt.plot(x_vec, y_vec)
plt.show()
```

Aufgabe 10: Stabilität

- Plotten Sie die Fehler für größere Snapshotmengen logarithmisch (`plt.semilogy`). Fällt Ihnen ein Problem auf?
- Ändern Sie Ihr Programm ab, um das Problem zu beheben.

Fehlernormen

Aufgabe 11: Vergleich verschiedener Fehlernormen

Vergleichen Sie die Energienorm von Fehlern mit einer parameterunabhängigen Norm.

- Welche parameterunabhängige Norm kann gewählt werden?
- Bestimmen Sie die Äquivalenzkonstanten
- Reproduzieren Sie die Grafik aus Bemerkung 3.22 für eigene Beispiele.

pyMOR-Update

pyMOR-Branch ändern und aktualisieren:

- Im Terminal in den Order `pymor` wechseln
- `git checkout 0.3.x`
- Eventuell
  - Eigene Änderungen commiten oder verwerfen (z.B. mit `git gui`)
  - Merge-Konflikte lösen
- `git pull`

Fehlerschätzer

Wir wollen nun Fehlerschätzer verwenden, dazu erstellen wir die reduzierte Diskretisierung mit der Funktion
`reduce_stationary_affine_linear`

Aufgabe 12: Fehlerschätzer

Erstellen Sie eine reduzierte Diskretisierung, die auch einen Fehlerschätzer enthält. Benötigt werden dazu:

- Ein geeignetes Fehlerprodukt. *Achtung:* Produkt mit festen Randwerten wählen, siehe `discretize_elliptic_cg` !!!
- Eine Abschätzung für die Koerzivitätskonstante (benutzen Sie z.B. `GenericParameterFunctional`).

Berechnen Sie den Fehlerschätzer für eine reduzierte Lösung.

Fehlerschätzer

Aufgabe 13: Vergleich von echtem Fehler und Fehlerschätzer

- Plotten Sie Fehler und Fehlerschätzer für verschiedene reduzierte Lösungen in einem Diagramm.
- Überprüfen Sie die Effektivität des Fehlerschätzers, indem Sie auch eine untere Schranke für den Fehler ausgehend vom Fehlerschätzer und der garantierten Effektivität plotten.

Aufgabe 14: „Parameter Sweep“

Vergleichen Sie Fehler und Fehlerschätzer für verschiedene Parameter, indem Sie eine Parameterkomponente variieren und alle anderen fest lassen.
Plotten Sie die Fehlerkurven für verschiedene (nicht allzu große) reduzierte Basen (z.B. uniform, verschiedene Random-Mengen, selbst gewählte Trainingsparameter)

Greedy-Algorithmus

Wir erstellen nun reduzierte Basen nicht mehr mit willkürlich gewählten Snapshots, sondern benutzen den Greedy-Algorithmus zur Basis-Generierung.

Aufgabe 15: Greedy-Algorithmus

- Generieren Sie eine reduzierte Basis mit dem Greedy-Algorithmus, benutzen Sie dazu die Funktion `greedy`.

  Hinweis: Sie benötigen außerdem die Funktion `functools.partial`.
- Schauen Sie sich die Ausgabe des Programms beim Durchführen des Greedy-Algorithmus und die Rückgabewerte der Funktion `greedy` an.

Maximale Fehler

Der Greedy-Algorithmus hat den Zweck, reduzierte Basen zu wählen, die zu einem möglichst geringen maximalen Fehler aller Parameter im Parameterraum führen. Es ist daher sinnvoller, nicht die Fehler einzelner Parameter zu vergleichen, sondern die maximalen Fehler einer Parametermenge.

Aufgabe 16: Maximale Fehler bei Greedy- und Standardbasis

- Berechnen Sie reduzierte Lösungen für verschiedene Parameter und plotten Sie den *maximalen Fehler* abhängig von der Modellordnung.
- Vergleichen Sie die maximalen Fehler von einer Greedy- und einer Standardbasis

Parameterauswahl beim Greedy-Algorithmus

- Laden Sie **hier** die Datei `greedy_chosen_values.py` herunter und speichern Sie das Skript im `pymordemos`-Ordner.
- Führen Sie das Skript aus und schauen Sie sich an, welche Parameterwerte vom Greedy-Algorithmus in welcher Reihenfolge ausgewählt werden. Vergleichen Sie `uniform` und `random`-Mengen.
- Ist die gegebene Reihenfolge der Parameter relevant? Setzen Sie `shuffle_samples = True`. Wie können Sie Ihre Beobachtungen erklären?

Overfitting

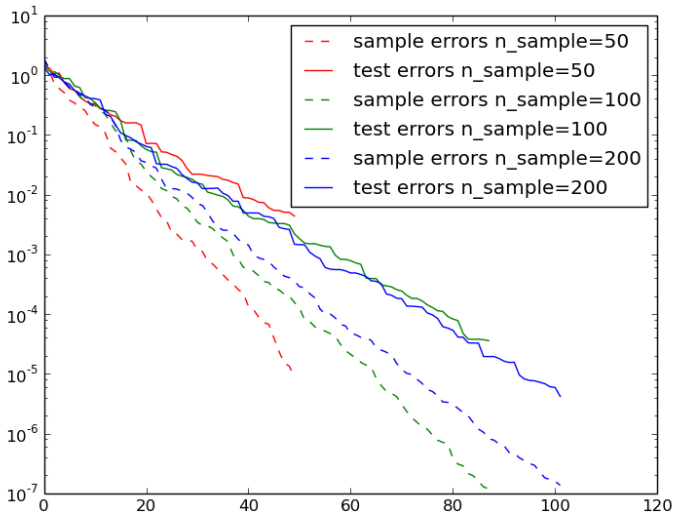
Wie verlässlich ist der maximale Fehler, der als Output beim Greedy-Algorithmus angegeben wird?

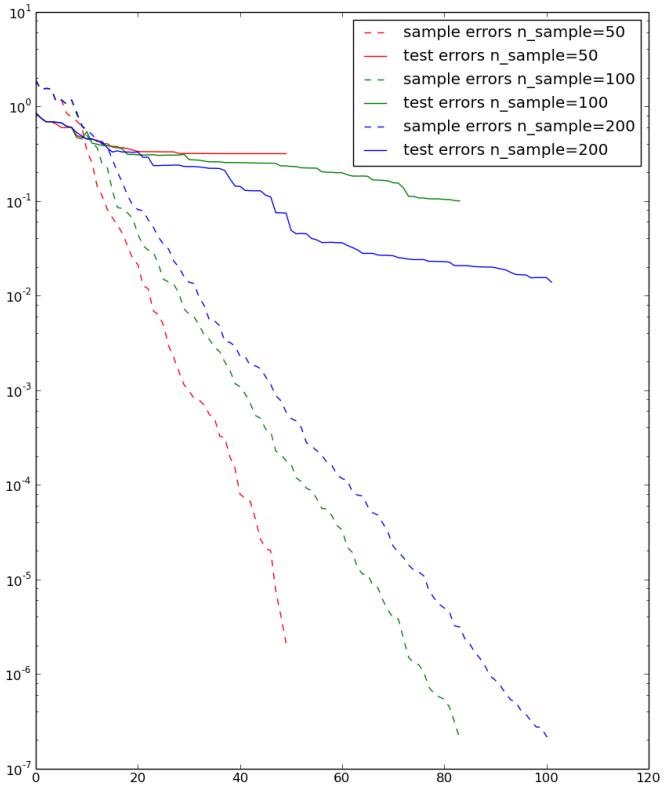
Aufgabe 17. : Overfitting

- Vergleichen Sie für verschiedene Samplemengen und Modellordnungen den maximalen Fehlerschätzer auf der Samplemenge mit dem maximalen Fehlerschätzer für eine zusätzliche fixe (genügend große) Testmenge.
- Wählen Sie als extremes Beispiel die Samplemenge absichtlich „schlecht“.
- Betrachten Sie insbesondere höhere Modellordnungen.

Overfitting

Vergleich von maximalem Fehler auf der Greedy-Sample-Menge und maximalem Fehler auf einer zusätzlichen Test-Sample-Menge von 5000 Parametern. Links: Random-Sample-Mengen, Rechts: Abgeschnittene Uniform-Sample-Mengen





POD

Wir beschäftigen uns nun mit der POD (Proper Orthogonal Decomposition), die für eine Menge an Vektoren hierarchische Bestapproximationens-Unterräume liefert.
Beispiel:

$$\begin{pmatrix}0\\0\\0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1/2\\1/2\\1/2\\0\end{pmatrix}, \begin{pmatrix}0\\1/2\\1\\1/2\\0\end{pmatrix}, \begin{pmatrix}0\\1/2\\1/2\\1/2\\0\end{pmatrix}, \begin{pmatrix}0\\0\\0\\0\\0\end{pmatrix} \longrightarrow \begin{pmatrix}0.\\1/2\\\sqrt{1/2}\\1/2\\0.\end{pmatrix}, \begin{pmatrix}0.\\1/2\\-\sqrt{1/2}\\1/2\\0.\end{pmatrix}$$

POD

Aufgabe 18: POD programmieren

Programmieren Sie (unabhängig von pyMOR) in Python eine Funktion, die die POD von in einer Matrix zusammengefassten Vektoren berechnet.

Verwenden Sie dazu die „Method of Snapshots“ (Satz 4.35) für den Spezialfall $X_{\mathcal{N}} = \mathbb{R}^{\mathcal{N}}$.

Das heißt, die Funktion berechnet aus einer Snapshot-Matrix $\in \mathbb{R}^{\mathcal{N} \times n}$ oder $\mathbb{R}^{n \times \mathcal{N}}$ die POD-Moden $\phi_i \in \mathbb{R}^{\mathcal{N}}$ und die Eigenwerte λ_i (oder die Singulärwerte σ_i) für $i = 1, \ldots, r \leq n$.

POD

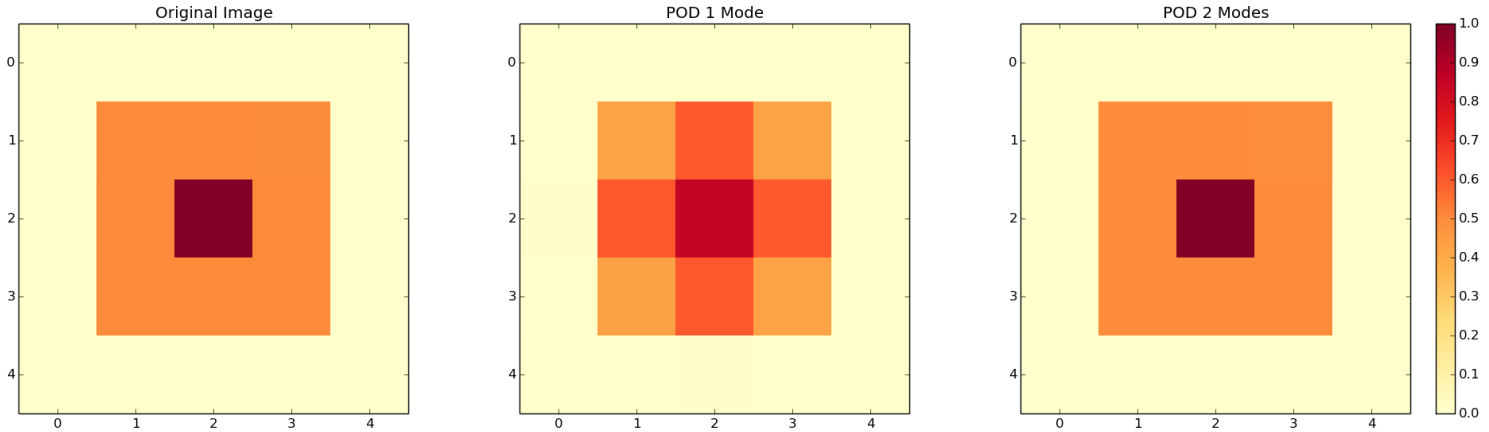
Hinweise:

- Das Eigenwertproblem kann mit der Funktion `eigh` aus dem Paket `scipy.linalg` (Eigenwertlöser für symmetrische/hermitesche Matrizen) gelöst werden.
- Entscheiden Sie, ob die Snapshots als Zeilen- oder Spaltenvektoren angegeben werden sollen.
- Achten Sie darauf, nur echt positive Eigenwerte zu berücksichtigen, geben sie dazu eine untere Schranke größer als die Maschinengenauigkeit vor.
- Achten Sie auf die Reihenfolge der Eigenvektoren. Die POD-Vektoren sollen mit absteigender Relevanz ausgegeben werden.

Approximationen von Matrizen und Bildern

Aufgabe 19: Projektion auf POD-Raum

Berechnen Sie Approximationen von Matrizen oder Bildern, indem Sie die Vektoren mit einer Orthogonalprojektion auf in den Raum der ersten r POD-Moden projizieren.



Aufgabe 20: Qualität der POD-Approximation

- Vergleichen Sie für verschiedene Matrizen (siehe Seite 49) und Fotos (siehe z.B. Seite 50, oder eigene Fotos) die Qualität der Approximationen
- Plotten Sie dazu auch die Eigenwerte der POD-Berechnung
- Schauen Sie sich die ersten POD-Moden an

Import und Visualisierung von Fotos

Import von PNG-Fotos (Schwarz-weiß, sonst eine Farbkomponente auswählen)

```
import matplotlib.image as mpimg
img = mpimg.imread('foto.png')
```

`img = img.astype(float)` (vorher beim importieren `float32`, macht Ärger bei den Berechnungen später)

Visualisierung von Matrizen und Bildern:

```
import matplotlib.pyplot as plt
plt.imshow(img)
```

- Optionen für `imshow`:
  - `cmap='gray'`: SW-Bild wirklich SW. Andere Colormaps siehe **hier**.
  - `vmin=0, vmax=1`: Wertebereich der Farbskala
  - `interpolation='none'`: Für Matrizen (geringerer Größe)

Beispielmatrizen

1. $A \in \mathbb{R}^{400\times 400}, A_{ij} = \begin{cases} 110, & 50 \le i \le 150, 30 \le j \le 130 \\ 150, & 50 \le i \le 150, 230, \le j \le 330 \\ 180, & 250 \le i \le 350, 70 \le j \le 170 \\ 220, & 250 \le i \le 350, 270 \le j \le 370 \\ 80, & \text{sonst} \end{cases}$

2. $B = \begin{pmatrix} 1 & \dots & \dots & 1 \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 1 \end{pmatrix}$

3. Gitterauswertung der Funktion
   $f(x,y) = (1-x)\cos(3\pi y(x+1))e^{-(1+x)y}$ auf $x \in [-1,1], y \in [0,\pi]$.

Beispielfotos

POD in pyMOR

Aufgabe 21: Implementierung in pyMOR

- Überlegen Sie sich kurz, was an Ihrer Implementierung geändert werden muss, damit sie mit pyMOR kompatibel ist.
- Vergleichen Sie dann Ihre Implementierung mit der POD-Funktion in `pymor.la.pod`.

Hausaufgabe bis zum 26.01.16

Vorlesungs-Kapitel zur **Empirischen Interpolation** detailliert wiederholen, insbesondere *Definition* des Algorithmus und die genaue *Implementierung* !!!

Reduzierte Basis mit POD

Aufgabe 22: ThermalBlock-Problem mit POD

- Benutzen Sie in Ihrer ThermalBlock-Demo eine POD-Basis.
- Schauen Sie sich die POD-Moden an
- Vergleichen Sie POD und Greedy-Basis im Hinblick auf Fehler und die Dauer der Offline-Phase

Empirische Interpolation

Es soll nun die Empirische Interpolation implementiert werden.

Aufgabe 23: EIM (Offline-Phase)

- Laden Sie sich **hier** die Vorlage für die EIM-Implementierung herunter und machen Sie sich mit der Struktur vertraut
- Implementieren Sie die Offline-Phase der EIM, z.B. indem Sie die Funktion `eim` in der Vorlage vervollständigen.
- Schauen Sie sich für die definierte Funktion $g((x_1, x_2), (\mu_1, \mu_2))$ die Verteilung der Interpolationspunkte und ausgewählten Trainingsparameter an. Vergleichen Sie Ihre Ergebnisse mit den Ergebnissen aus dem EIM-Artikel **[Grepl, Maday, Nguyen, Patera, 2006]**.

Hinweise:

- Die Projektionsmatrix $\mathbb{P}$ muss nicht zwingend implementiert werden, denn $\mathbb{P}^t\mathbb{Z}$ entspricht dem Aufruf `Z[J,:]` für die Liste der Interpolationsindizes `J`.
- Die meisten Werte können einfach in Listen gespeichert werden. Die Matrix $\mathbb{Z}$ sollte als `np.array()` gespeichert werden.
  - Entweder kann sie als genügend große leere Matrix angelegt und dann aufgefüllt werden (dann muss in den Teilschritten auf den entsprechend gefüllten Teil der Matrix zugegriffen werden).
  - Eine andere Möglichkeit ist, eine leere Matrix der Größe $N_h \times 0$ iterativ mit `np.append( ,axis=1)` oder `np.hstack()` zu vergrößern. Achtung bei den Shapes!

RB Verfahren für nichtaffine Probleme

Anwendungsbeispiel: Massentransport mit parametrisiertem Quellterm

(aus [Quarteroni, Manzoni, Negri: Reduced Basis Methods for Partial Differential Equations, 2015])

$$\begin{cases} -\mu_1 \Delta u + \boldsymbol{b}(\mu_2) \cdot \nabla u + a_0 u &= s(\boldsymbol{\mu}) \quad \text{in } \Omega = [0,1] \times [0,0.5] \\ \mu_1 \nabla u \cdot \boldsymbol{n} &= 0 \quad \text{auf } \Gamma_N = \partial\Omega \end{cases}$$

mit $\boldsymbol{b}(\mu_2) = \begin{pmatrix} \cos(\mu_2) \\ \sin(\mu_2) \end{pmatrix}$ und $s(\boldsymbol{x}, \boldsymbol{\mu}) = \exp\left(-\frac{(x_1 - \mu_3)^2 + (x_2 - \mu_4)^2}{\mu_5^2} \right)$.

Schwache Formulierung:

$$a(u, v; \boldsymbol{\mu}) = \int_\Omega \left(\mu_1 \nabla u \cdot \nabla v + \cos(\mu_2) \frac{\partial u}{\partial x_1} v + \sin(\mu_2) \frac{\partial u}{\partial x_2} v \right) d\Omega$$

$$f(v; \boldsymbol{\mu}) = \int_\Omega s(\boldsymbol{x}; \boldsymbol{\mu}) v d\Omega$$

In der schwachen Formulierung Bilinearform affin zerlegbar, aber rechte Seite nicht (Quellterm!!) $\longrightarrow$ Empirische Interpolation

Empirische Interpolation für die rechte Seite

$$s(\boldsymbol{x}, \boldsymbol{\mu}) = \exp\left(-\frac{(x_1 - \mu_3)^2 + (x_2 - \mu_4)^2}{\mu_5^2}\right) \tag{1}$$

- Wähle $\mu_5 = 0.25$ fix, $\mu_3 \in [0.2, 0.8]$ und $\mu_4 \in [0.15, 0.35]$.
- Quadraturpunkte in $\boldsymbol{x}$: Dreiecksgitter mit 4. Ordnung Quadratur ergibt $N_q = 62208$.
- Trainingsmenge mit $n_{train}^{EI} = 1000$
- $\varepsilon = 10^{-3}$

Empirische Interpolation ergibt kollaterale Basis der Größe $Q_f = 30$.

RB-Verfahren

$$\begin{cases} -\mu_1 \Delta u + \boldsymbol{b}(\mu_2) \cdot \nabla u + a_0 u &= s(\boldsymbol{\mu}) \quad \text{in } \Omega = [0,1] \times [0,0.5] \\ \mu_1 \nabla u \cdot \boldsymbol{n} &= 0 \quad \text{auf } \Gamma_N = \partial\Omega \end{cases} \tag{2}$$

- Wähle $\mu_1 = 0.03$ fix, $\mu_2 \in [0, 2\pi]$
- Benutze affine Zerlegung für die rechte Seite aus der EI (für verschiedene Werte von Q_f)
- Berechne $n^{RB}_{train} = 150$ Snapshots und eine POD-Basis der Größe 85.

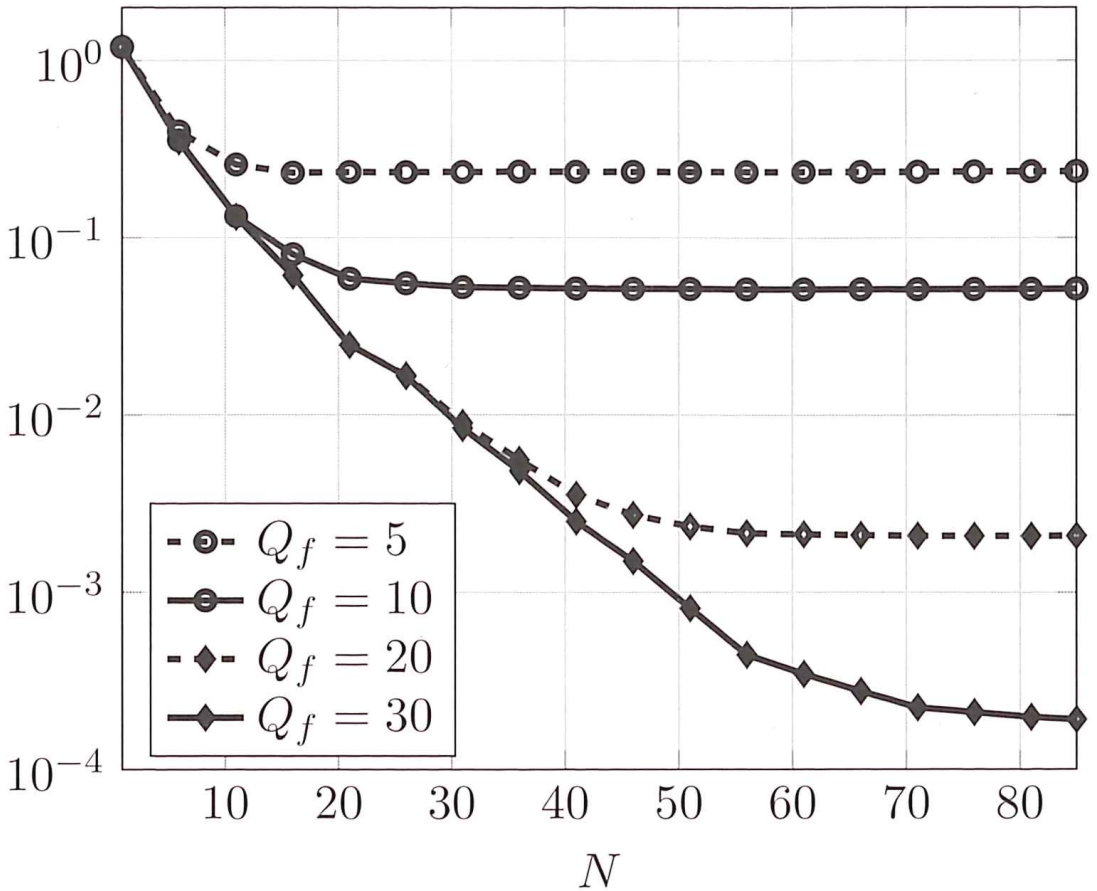


Abbildung: Fehler für die reduzierten Lösungen für verschiedene Modellordnungen N und verschiedene Größen der kollateralen Basis Q_f.