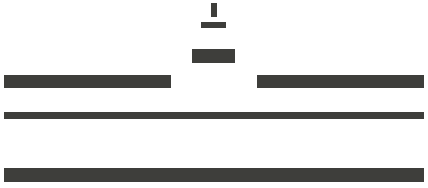


Westfälische
Wilhelms-Universität
Münster

Praktikum Modellreduktion

für partielle Differentialgleichungen

wissen.leben
WWU Münster

René Milk

Sommersemester 2014

Organisatorisches

- Anwesenheit (Kursbuchungssystem, 936)
- Linuxerfahrung?
- Programmiererfahrung?

Material

- https://en.wikibooks.org/wiki/Non-Programmer's_Tutorial_for_Python
- http://docs.python.org/2/reference/
- http://wwwmath.uni-muenster.de/num/Vorlesungen/Pythonkurs_WS1314/
- http://docs.scipy.org/doc/numpy/reference/index.html
- http://wiki.scipy.org/Tentative_NumPy_Tutorial
- https://help.ubuntu.com/community/UsingTheTerminal
- my_code_isnt_working.pdf
- http://pymor.readthedocs.org/en/0.2.x/
- Git Primer

PyCharm3 Setup

1. Runterladen
2. Entpacken
3. Nachvollziehen

PyCharm3 Benutzung

- Ausführen/Debuggen
- Git
- Hilfe

Aufgaben

1. https://github.com/wwu-numerik/ss14-modellreduktion-gitintro
2. https://github.com/wwu-numerik/ss14-modellreduktion-pythonintro

Numpy Crashcourse

```
import numpy as np

a = np.array([1, 2, 3, 4])
a.shape == (4,)
a.dtype == numpy.int64

a = np.array([[1., 2.],
              [3., 4.],
              [5., 6.]])
a.shape == (3,2)
a.dtype == numpy.float64

a[1,:]   # komplette ZWEITE zeile
a[:,0] *= 2  # erste spalte elementweise multiplizieren
print(a) # werte von a geaendert
a[:,0] - a[:,1]
a[1:3,0] # [3,5]

a.dot(a) # fehler
a.dot(a.transpose())
```

Numpy Crashcourse

Anlegen, reshaping, slicing, broadcasting, masking

Numpy Crashcourse

```
# python schleifen vs. Numpy Vektorisierung
import numpy as np

# neuer Vektor durch elementweise Addition
end = 10000000
A = np.arange(end)
B = np.arange(end)
%%timeit
C = []
for a,b in zip(A,B):
    C.append(a+b)

%timeit _ = [a+b for a,b in zip(A,B)]
%timeit _ = A+B

# zaehlen der durch 13 teilbaren zahlen

%%timeit
count = 0
for i in A:
    if i % 13 == 0:
        count += 1

%timeit _ = sum((1 for i in A if i % 13 == 0))
%timeit _ = np.sum(np.mod(A, 13) == 0)
```

numpy.einsum

http://docs.scipy.org/doc/numpy/reference/generated/numpy.einsum.html

```
import numpy as np

a = np.arange(25).reshape(5,5)
b = np.arange(5)
c = np.arange(6).reshape(2,3)

np.einsum('ii', a) # 60
np.trace(a)

np.einsum('ii->i', a) # array([ 0,  6, 12, 18, 24])
np.diag(a)

np.einsum('ij,j', a, b) # array([ 30,  80, 130, 180, 230])
np.dot(a, b)

np.einsum('ji', c)
c.T
```

Aufgaben

1. Nachvollziehen der Vektorisierungsfolie in einer iPython-Sitzung
2. Schnellere Varianten der python/numpy Implementierungen?
3. einsum Variante von a.dot(a).dot(b)

Nächste Sitzung

Eigener Rechner? pyMOR 0.2.0 installieren!

Nachtrag zur zweiten Sitzung

- Lösung der Python Aufgaben
- GitHub Interaktion vereinfachen:
  1. Access Token erstellen
  2. in Pycharm: “File” -› “Settings” -› “Version Control” -› “GitHub” umstellen von “AuthType: password” auf “Token” und Access Token da eintragen.

pyMOR intro

- Paketbaum
- Dokumentation!

Neues Pymor Projekt

1. `http://github.com/pymor/pymor` forken
2. in pyCharm: VCS -› Checkout from Version Control -› GitHub
3. pymor fork auswählen
4. VCS -› Git -› branches -› “origin/0.2.x” -› Checkout as new local branch
5. File -› Settings -› Project Structure -› Rechtsklick “src” Unterorder -› als “source” markieren

Aufgabe

Berechnung des *EOC* mit *N* Stufen in `src/pymordemos/cg.py`. Plotten der Fehlerkurve gegen Dreiecksanzahl. Dazu ein festes Settings (Randart/-wert, rhs) wählen und exakte Lösung bestimmen.

```
exact = GenericFunction(lambda X: ..., 2)
U_ex = NumpyVectorArray(exact.evaluate(grid.centers(2)))
```

Definition (Experimental Order of Convergence)

Es sei eine Funktion f und eine Folge f_{h_i} von Approximationen von f auf der Gitterebene i gegeben. Sei außerdem e_i der Approximationsfehler in einer geeigneten Norm und h_i die Gitterweite. Dann definieren wir für $i > 0$ die Experimental Order of Convergence (*eoc*) von f_{h_i} des Levels i durch

$$eoc_f(i) := \frac{\ln\left(\frac{e_i}{e_{i-1}}\right)}{\ln\left(\frac{h_i}{h_{i-1}}\right)} \tag{1}$$

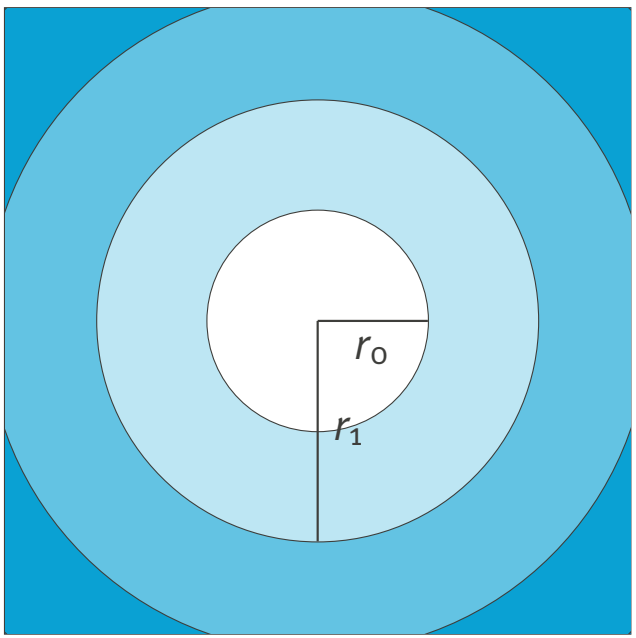
Hausaufgabe

the thermalblock demo explained

Aufgaben

1. Neues Demoscript anlegen: `src/pymordemos/radial_thermalblock.py`
2. neues analytical Problem mit radialen Diffusionsblöcken (nächste Folie) anlegen und einbinden
3. Scriptparameter von `X,Y` Blöcke auf `N` Ringe ändern
4. Ausführen !

Diffusionsringe



Ω

$$i = 0 \ldots N-1$$
$$r_i = (i+1)r(\Omega)/N$$
mit $\quad \Omega = [0,1]^2, N = 4$
$$r_i = (i+1)2^{-\frac{1}{2}}/4$$

Beispiellösung

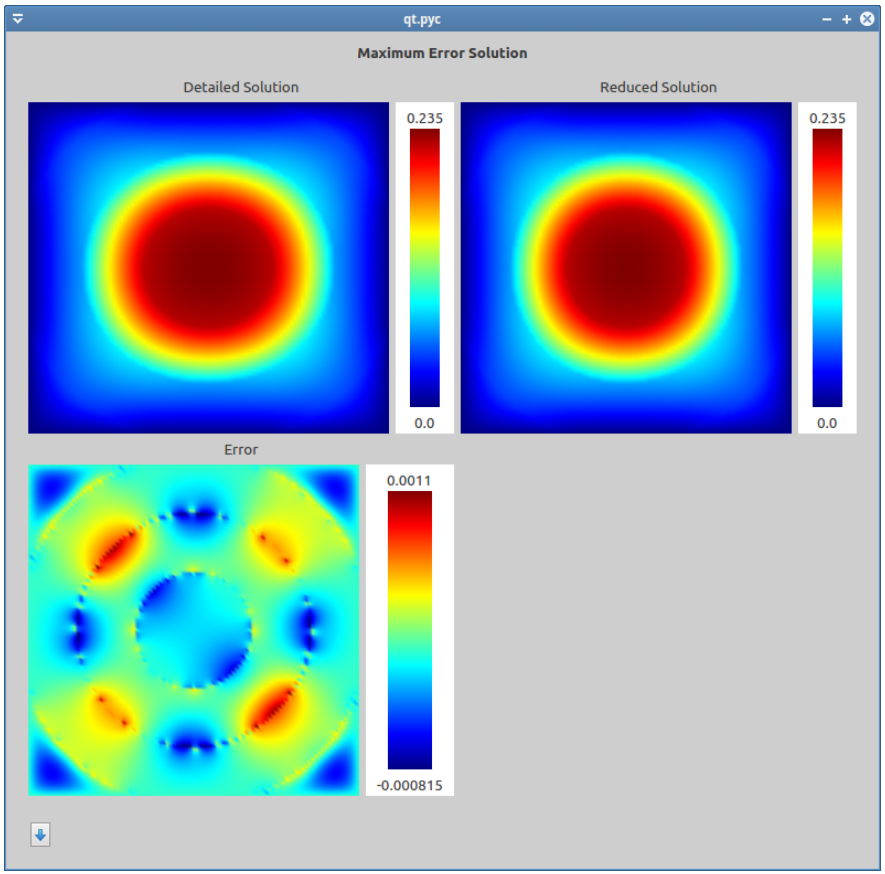


Abbildung : RINGS=4, SNAPSHOTS=8, RBSIZE=8

Aufgabe 1

Neumann-0 Randwerte am oberen und unteren Rand für RadialThermalblock Demo: RectDomain mit entsprechender BoundaryInfo.

Aufgabe 2

Ziel: LaTeX Dokument mit einer figure die 2x2 subfigures enthält. Die X-Achse soll jeweils Basisgrösse darstellen, die Y-Achse maximalen H^1, H^1_{est}, sowie Kondition des reduzierten Operators und Effizienz des Fehlerschätzers. Jeder Plot soll 4 Datenreihen für unterschiedliche Blockkonfigurationen (etwa 1x4, 2x2) enthalten. Es soll nur eine gemeinsame Legende gezeigt werden.
Ändern Sie dazu die `thermalblock` demo so ab, dass die berechneten Daten in eine CSV-Datei geschrieben werden. Anschliessend sollen die Plots dann mit `pgfplots` daraus generiert werden (Bsp. nächste Folie).

pgfplotsexample.tex

```
\documentclass[a4paper,10pt]{article}
\usepackage[utf8]{inputenc}
\usepackage{pgfplots}
\begin{document}
\begin{figure}[h!]
    \begin{tikzpicture}
      \begin{axis}[width=0.5\linewidth, xlabel=RBSIZE, ylabel=Err]
        \addplot table[x=links,y=hoch,col sep=comma] {data.csv};
        \legend{Config A}
      \end{axis}
    \end{tikzpicture}
    \caption{Bildunterschrift}
\end{figure}
\end{document}
```

data.csv

```
links, hoch
1, 2.2
2, 2
4, 4
8, 2.9
```

Resultat

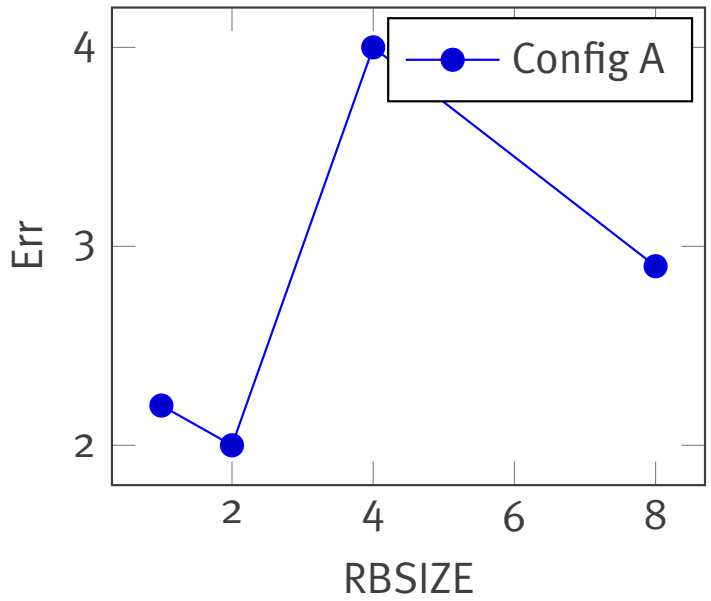


Abbildung : Bildunterschrift

Caching

- Global abschalten: `PYMOR_CACHE_DISABLE=1` in Prozessumgebung
- Aktivieren: von `pymor.core.cache.CacheableInterface` ableiten, dekorieren mit `pymor.core.cache.cached`
- wahlweise in-memory oder on-disk -> flüchtig / persistent zwischen pyMOR Instanzen
- implementiert als `DogpileCacheRegion`
- default ist in-memory
- cache key ist generiert aus state-ID des Objekts, der state-IDs der Funktionsargumente wenn verfügbar, sonst deren `pickle` Darstellung

Beispiel

```
# pymor/discretizations/interfaces.py:64
@cached
def solve(self, mu=None, **kwargs):
    return self._solve(mu, **kwargs)
```

Aufgabe: elliptisches CG

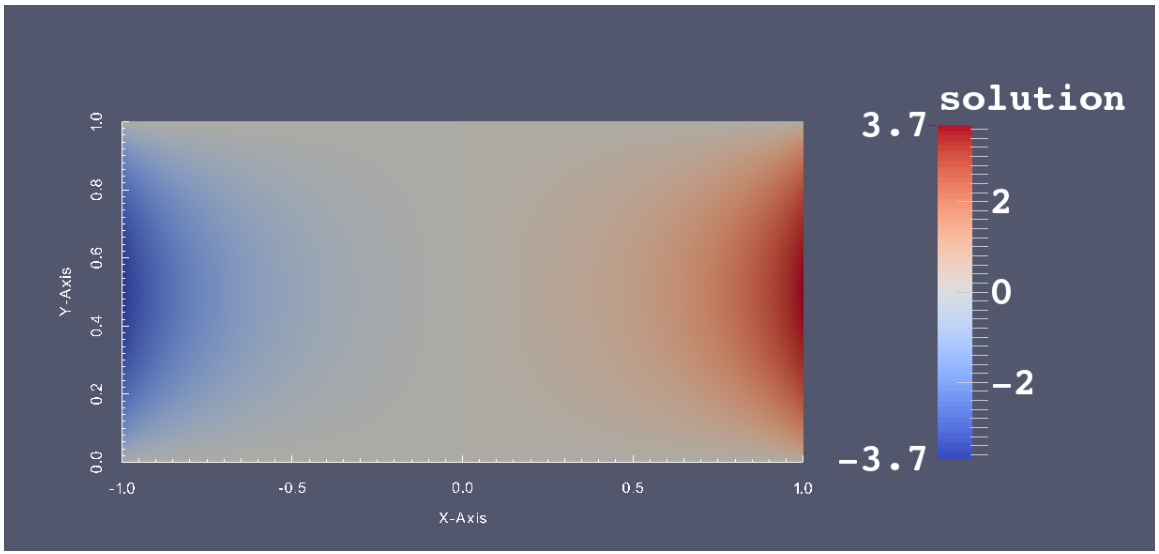
Randwerte

- links: $\frac{\partial}{\partial n} u(x) = -1$
- rechts: $\frac{\partial}{\partial n} u(x) = 1$
- oben, unten: $u(x) = 0$

sonst.

- $\Omega = [-1, 1] \times [0, 1]$
- $f(x) = 0$
- Diffusion: $\mu(x) = 0.1$

Aufgabe: elliptic. CG



Live Demo

`screen`

Zum nachlesen: `http://aperiodic.net/screen/quick_reference`

Aufgabe 1

Experiment aus Sitzung 5, Aufgabe 2 für grosse Datenreihen ($>$ 20 RB-Size) in `screen` ausführen (`nice`) und alle Ausgaben+Pythoncode gezipped per Email an mich schicken.

Visualizer Tour

Aufgabe 2

1. Runterladen und in neuem Verzeichnis entpacken: `pymor-elasticity.tgz`
2. Git-Repository initialisieren und Dateien hinzufügen.
3. neue virtualenv im Verzeichnis anlegen: `$ virtualenv -system-site-packages env`
4. Unterverzeichnis `env` zu `.gitignore` hinzufügen
5. Neues PyCharm Projekt anlegen. Dabei neuen Interpreter anlegen mit `env/bin/python`
6. Visualisierung via matplotlib in `gui.py` implementieren.
7. gui.py per Email an mich.

Visualisierungsbeispiel

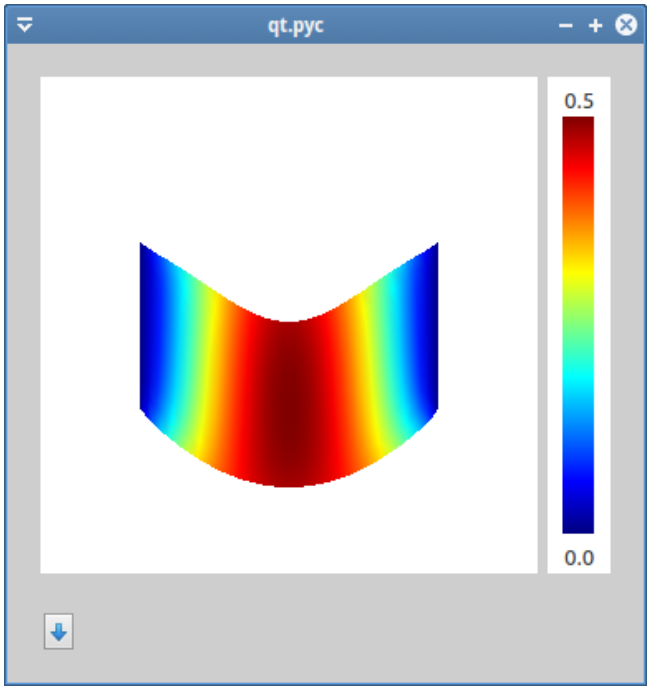


Abbildung : demo.py 1010 10 4

py.test

- warum unittests / py.test ??
- Alle Tests ausführen `$ cd pymor/src/ ; py.test pymortests`
  oder nach aktivieren der virtualenv `$ cd pymor/ ; make-full-test`
- Test nach Namen auswählen `$ cd pymor/src/ ; py.test pymortests/ -k test_centers_wrong_arguments`
- Alles Tests einer Datei ausführen: wie gewohnt in pycharm
- np.testing
- Plugins: coverage, pep8, flakes
- Travis
- Referenz: `http://pytest.org/latest/`

Aufgaben

1. Test für induced_norm (`src/pymortests/la.py`) erweitern um ausgewählte konstante, polynomiale, transzendete Funktionen. Hinweis: `src/pymortests/tools.py`
2. Fügen Sie neue, möglichst umfassende, Tests für den Gram-Schmidt Algorithmus hinzu.
3. Finden Sie eine Möglichkeit die Namen aller von py.test gefundenen Tests auszugeben.

- VectorArrays
- Fixtures in `vectorarray.py`

Aufgabe 1

Kreieren Sie eine Ableitung `MatlabListVectorArray` von `la.listvectorarray.ListVectorArray` mit einer passenden Implementierung von `la.listvectorarray.VectorInterface`. Das `MatlabListVectorArray` soll als `__init__` Argument einen Dateinamen erhalten und dann aus dieser, im Matlab v7.3 Format angenommenen, Datei Vektoren auslesen und ensprechend in die Liste einfügen. Sorgen Sie dafür, dass möglichst viele der vorhandenen ListVectorArray Tests für ihre neue Implementierung ausgeführt werden.

Hinweise

- matlab (2014) -› preferences -› general -› MAT-Files oder `save('filename','var','-v7.3')`
- v7.3 Dateien sind hdf5 Datensets -› einlesen mit h5py Modul

Aufgabe 2

Vektoren nur bei Bedarf aus der Datei lesen, nach Operationen wieder zurückschreiben und Speicher in Python freigeben.