Introduction to Numerical Linear Algebra

Dr. Christian Vollmann

Winter Term 2022/2023

At Trier University variants of this course serve the modules:
– Elements of Mathematics
– Elemente der Linearen Algebra
– Numerical Methods for Geoscientists

A Short Note on Programming

...and specifically on Python.

Programming Languages in Numerical Mathematics (selection)

- **Fortran** (FORmula TRANslation) (1957):
  - proprietary (e.g. from IBM) and free compilers
  - intended for numerical calculations (matrix and vector operations)
  - extensive libraries
  - LAPACK (**L**inear **A**lgebra **Pack**age) standard library for numerical linear algebra
- **C** (1972)/ **C++** (1985):
  - universal programming language
  - Standard libraries for numerics: Armadillo, LAPACK++ (based on LAPACK)
- **MATLAB** (MATrix LABoratory) (1984):
  - proprietary software from MathWorks
  - designed for numerical mathematics (matrix and vector operations)
- **Mathematica** (1988):
  - proprietary software from Wolfram Research
  - visualization of 2d/3d objects
  - symbolic processing of equations
  - see also: https://www.wolframalpha.com/
- **Python** (1990): open source
  - universal programming language (several application areas)
  - for numerical calculations: SciPy (2001), NumPy (1995,2006), matplotlib (2003).
- **Julia** (2012): open source
  - developed mainly for scientific computing
  - syntax looks like MATLAB
  - execution speed is in the range of C and Fortran

→ All programming related parts of this lecture will be presented and implemented using **Python 3**

Why Python?

- universal, multi-purpose programming language
  - many packages for scientific computing, web development, ...
- open source and free (Python Software Foundation License (PSFL), OSI/FSF approved)
- multi-platform (runs on all operating systems)
- design philosophy: easy syntax, readable code (almost looks like pseudocode)
- Also see: https://www.youtube.com/watch?v=M0vBoBqqjr0

Background

- developed in 1990 by Guido van Rossum (Netherlands)
  - name is homage to Monty Python
- interpreter (scripting) programming language
  ($\neq$ compiled language such as C or Fortran)
- used by: Google Mail, Google Maps, YouTube, Dropbox, sphinx, and many more
- for scientific computing we use from the Scipy Stack:
  - **NumPy** (1995,2006)
  - **SciPy** (2001)
  - **matplotlib** (2003)

Programming Workflow

CLI [ex:demo]

- Any text editor can be used (emacs, vi, vim, nano, geany, gedit,...)
  - many editors provide syntax highlighting
- Install Python and then interpret the source code

IDE [ex:demo]

- For software development it is often more convenient to use an **integrated development environment (IDE)**
- Specifically for Python:  PyCharm[1],  Spyder

[1] sign up to jetbrains with your university account and you can get the PyCharm professional edition!

For exercise submission/presentation:

Jupyter-Notebook [ex:demo]

- open source, *web based* interactive environment
  $\rightarrow$ thus multi-platform
- developed by **Project Jupyter** (NPO)
  $\rightarrow$ name refers to: **Ju**lia, **Pyt**hon, **R**
- the whole process can be documented:
  Coding $\rightarrow$ Documentation $\rightarrow$ Run $\rightarrow$ Communication and Presentation
- in fact, a jupyter notebook contains all the input **and** output of an interactive session plus additional text
  $\rightarrow$ complete record!
- client VS server
  - client (local lightweight machine): browser-based workflow
  - server (remote, number cruncher): does the actual computation

Get Started

- We recommend to download the distribution ***Anaconda***:

  https://www.anaconda.com/distribution/

  → available for Linux, Windows, and MacOS

- Comes along with:
  - graphical user interface (*Anaconda Navigator*)
  - Spyder, Jupyter Notebook, RStudio (IDE for R)
  - installs all important packages (NumPy, SciPy, matplotlib, TensorFlow, scikit-learn,$\ldots$)
  - package manager (*Conda*) (standard is *pip*)

Tutorials

- Scientific computing with Python:
  https://scipy-lectures.org/
- Official Online-Documentation:
  https://docs.python.org/3/
- Official Python Tutorial:
  https://docs.python.org/3/tutorial/index.html
- Quickstart to Jupyter Notebook:
  https://jupyter.readthedocs.io/en/latest/content-quickstart.html

Final remark:

- For Software development I would always go with an IDE due to the many additional tools: debugging, variable explorer, version control, file manager, etc.
- However, Jupyter Notebooks are very well suited for presentations and thus teaching. In particular for mandatory submissions, since the tutor can see your output, even if the program does not run on his/her machine (for whatever reasons).

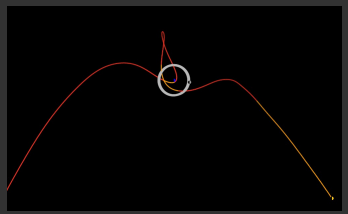
A Short Introduction to the Topic

From the Module Handbook:
*"After completing the module, the students know the mathematical foundations in the areas of linear algebra and **numerical mathematics**. As part of the course, they acquire or deepen knowledge in the programming language **Python**."*

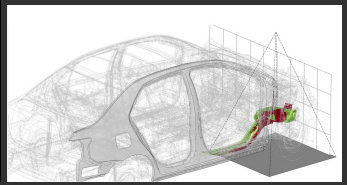
Numerical mathematics?
From Wikipedia: Field of mathematics that deals with the construction and analysis of algorithms to approximately (but accurately) compute solutions to (hard) continuous problems – typically using computers.

Why is this important?

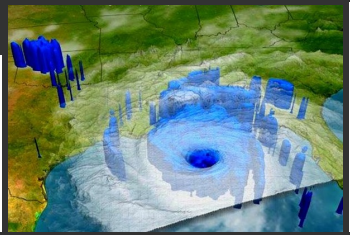
- most (all?) **application**-oriented problems cannot be solved exactly $\rightarrow$ ***hard*** problems



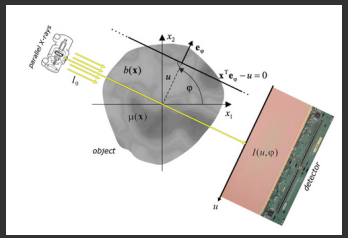
trajectories of objects in space
2020SO



car crash simulation



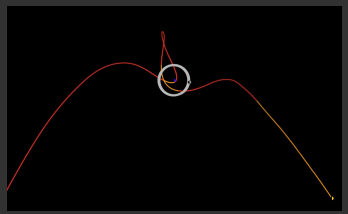
weather prediction



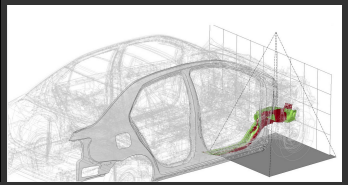
CT Scan

Relation to Data Science?

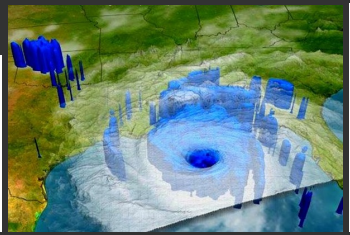
- due to the high amount of data available **data**-driven models are more important than ever
- data can be considered as a **mathematical object** (e.g., as a matrix/vector)
- with **numerical algorithms** we can manipulate data:
  - solve systems involving the data (fitting data, prediction,...)
  - extract the most important features (singular values, PCA, data compression,...)
  - calibrate models against data (machine learning, neural networks,...)



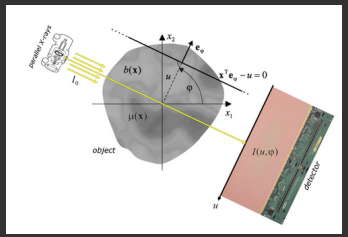
trajectories of objects in space
2020SO



car crash simulation



weather prediction



CT Scan

Preview

Let us assume we have m data points

$$(z_i, y_i), \quad i = 1, \dots, m,$$

where

- z_i are n-dimensional vectors of *explanatory features*
- y_i are k-dimensional vectors representing the *response/prediction/classification*

$\rightarrow$ *The term "vector" already indicates that Linear Algebra comes naturally into the game.*

Examples

- You ask m persons about

$$z_i = (\text{age, sex, weight, height, years of experience}) \qquad (n = 5 \text{ dimensional vector})$$
$$y_i = \text{salary} \qquad (k = 1 \text{ dimensional vector})$$

- Consider m years where

$$z_i = \text{year} \qquad (n = 1 \text{ dimensional vector})$$
$$y_i = \text{global mean temperature} \qquad (k = 1 \text{ dimensional vector})$$

- Consider m images that you want to classify

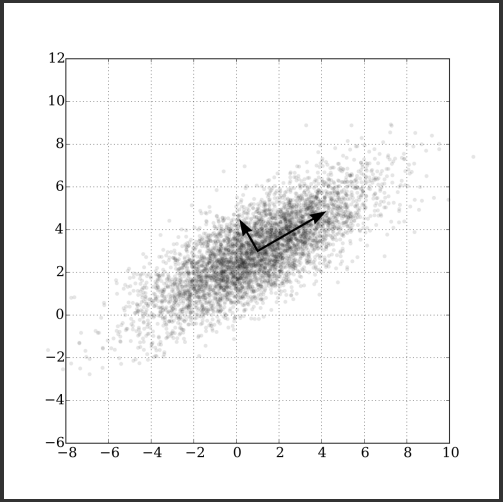
$$z_i = (p_{lj})_{lj} \qquad (\text{image stored as matrix/vector})$$
$$y_i = (\text{dog, cat, elephant}) \qquad (k = 3 \text{ dimensional vector})$$

Dealing solely with the features z

Applications of the **Singular Value Decomposition** are:

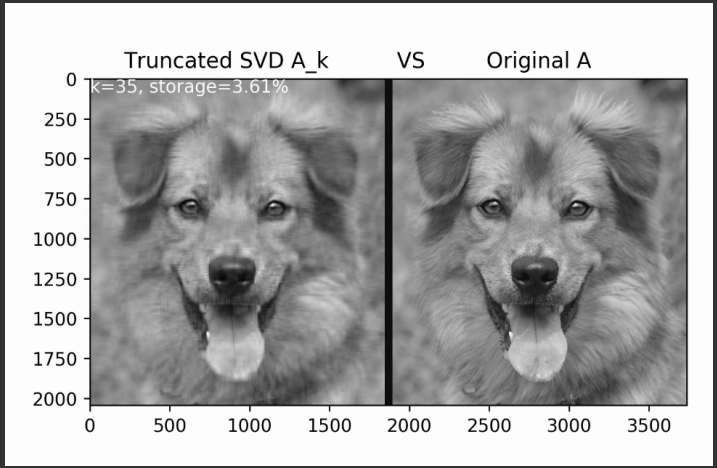
Principal Component Analysis (PCA)

$\rightarrow$ Aim: dimension reduction



Data compression

$\rightarrow$ Aim: compression without dimension reduction



Relating features z to response y

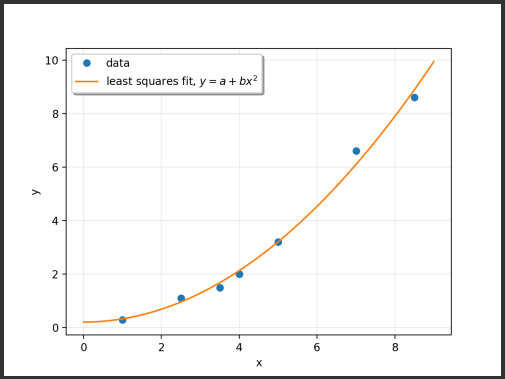
One central goal in many scientific fields is to find a **model** f_x depending on some parameters $x = (x_i)_i$, which "best" explains the relation between z_i and y_i in the sense that

$$f_x(z_i) \approx y_i, \quad \text{for all} \quad i = 1, \dots, m$$

- **The task:** Find those parameters x for which the "distance" between our prediction $f_x(z_i)$ and the measured response y_i is "as small as possible".
- Math gives us the tools to rigorously define this task, to analyze it systematically and to provide numerical solutions!

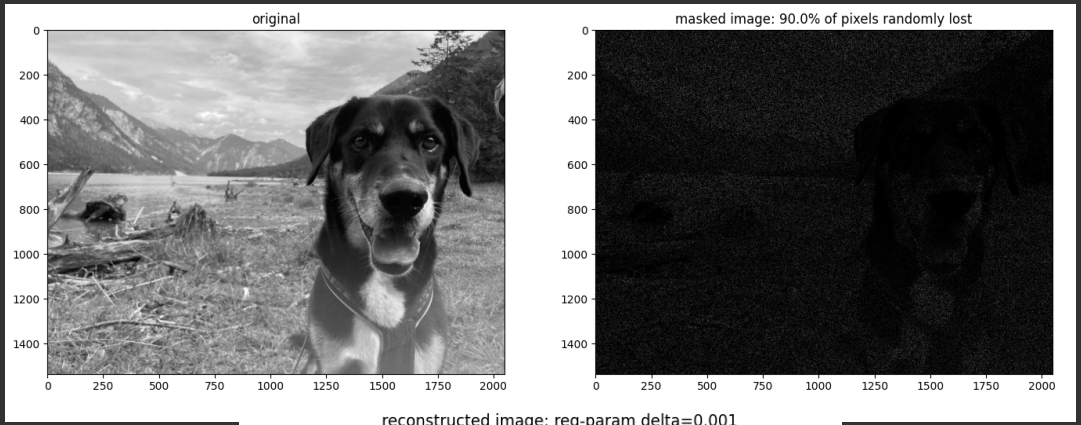
Curve Fitting

$$f_x(z) := x_0 + x_1 z + x_2 z^2$$



Simple image inpainting

$$\min_x \|Ax - b\|^2 + R(x)$$



Take away message

Math provides a rigorous framework of abstract structures (Definition) within which their properties and relations (Theorem) can be systematically analyzed (Proof).

The same mathematical concept can be exploited in multiple applications; see, e.g., SVD above.

Given a real-world problem the user has to provide an interface (a mathematical model) to these concepts to make use of the mathematical theory.