Installation von Python

In der Vorlesung setzen wir die Programmiersprache Python zusammen mit der Umgebung Jupyter ein. Skripte sowie „Mitschnitte“ der Vorlesung werden im entsprechenden Emil-Lernraum als sogenannte *Notebooks* zum Download angeboten werden. Das sind Dateien mit der Endung `.ipynb`. Damit Sie diese Notebooks lesen, drucken, konvertieren und insbesondere ausführen können, brauchen Sie Jupyter.

Sie können Jupyter zwar auch online über einen Service wie „Try Jupyter“ benutzen, es empfiehlt sich aber, die Software lokal zu installieren. Falls Sie schon Erfahrung mit Python haben, können Sie Jupyter wahrscheinlich selbst installieren oder haben das sogar schon getan. Anderenfalls gibt es verschiedene Möglichkeiten, von denen hier eine besonders einfache vorgestellt wird, die sowohl für Windows als auch für OS X funktionieren sollte[1] und die auch gleich Python sowie diverse Bibliotheken und Entwicklungstools installiert:

[1] Es funktioniert wohl auch für Linux, aber wenn Sie Linux benutzen, sollten Sie Jupyter evtl. über Ihren Paketmanager installieren.

(i) Laden Sie Anaconda von `https://www.continuum.io/downloads` herunter. Wählen Sie die Version für Python 3 aus (und typischerweise die 64-Bit-Variante, wenn Sie nicht einen sehr alten Rechner haben).

(ii) Installieren Sie Anaconda und akzeptieren Sie dabei die Standardeinstellungen. (Insbesondere sollten Sie Anaconda *nicht* für alle Benutzer, sondern nur für sich installieren.)

(iii) Unter Windows öffnen Sie danach eine Konsole; und zwar möglichst die, die über das Startmenü als „`Anaconda Prompt`“ erreichbar ist. Geben Sie nacheinander die beiden folgenden Befehle ein, um die Distribution auf den neuesten Stand zu bringen:

```
conda update conda
conda update anaconda
```

Beantworten Sie Fragen, die Ihnen evtl. gestellt werden, mit `y(es)`.

(iv) Unter OS X verfahren Sie ebenso, allerdings mit diesen Befehlen:

```
anaconda update conda
anaconda update anaconda
```

(v) Führen Sie nun den Befehl

```
ipython profile create
```

aus. Es werden zwei Dateien angelegt, von denen Sie die Datei `ipython_config.py` mit einem Texteditor bearbeiten sollen. Fügen Sie die folgende Zeile hinzu:

```
c.InteractiveShellApp.matplotlib = "inline"
```

(Alternativ können Sie die mit „`# c.InteractiveShellApp.matplotlib`“ beginnende Zeile editieren. Achten Sie in diesem Fall darauf, das Kommentarzeichen „#“ am Anfang der Zeile zu entfernen.)

(vi) Jetzt sind Sie schon fertig und sollten JUPYTER starten und benutzen können.

(vii) Optional können Sie noch ein paar Anpassungen vornehmen, wenn Sie wollen. Geben Sie dazu in der Konsole den folgenden Befehl ein:

```
jupyter notebook --generate-config
```

Als Antwort wird Ihnen angezeigt werden, dass eine Datei angelegt wurde. Diese Konfigurationsdatei können Sie nun mit einem Texteditor bearbeiten, um das Verhalten von JUPYTER nach dem nächsten Start zu modifizieren.

Suchen Sie in der Datei z.B. die Zeile

```
# c.NotebookApp.notebook_dir = ''
```

und ändern Sie sie folgendermaßen um:

```
c.NotebookApp.notebook_dir = '/Users/meinName/Notebooks'
```

Beachten Sie dabei, dass das Kommentarzeichen „#“ am Anfang der Zeile entfernt wurde. Außerdem sollten Sie `meinName` natürlich durch den Namen Ihres Windows-Kontos ersetzen. Beim nächsten Start wird JUPYTER nun den Ordner `Notebooks` in Ihrem Benutzerverzeichnis benutzen. (Dafür muss dieser Ordner allerdings existieren!)

Lesen Sie sich die Kommentare in der Konfigurationsdatei durch, um herauszufinden, was man ansonsten noch ändern kann. Zum Beispiel:

```
c.NotebookApp.port = 4242
c.NotebookApp.open_browser = False
```

Mit diesen beiden Einstellungen öffnet JUPYTER immer Port 4242, und es wird nicht automatisch ein Browser gestartet, wenn man JUPYTER startet. Stattdessen sollten Sie sich nun ein Lesezeichen für die URL `http://localhost:4242/` speichern.

Entsprechend kann man bei neueren Versionen von JUPYTER auch die Sicherheitseinstellung umgehen, durch die man gezwungen wird, am Anfang einer Sitzung einen sogenannten *Token* einzugeben:

```
c.NotebookApp.token = ''
```

Bitte beachten Sie, dass Sie es hier mit Software zu tun haben, die regelmäßig aktualisiert wird. Es kann also durchaus vorkommen, dass sich die Syntax von Befehlen oder von Konfigurationsoptionen gegenüber der Darstellung hier im Buch geändert hat. Im Zweifelsfall müssen Sie die Dokumentation von JUPYTER oder von ANACONDA konsultieren oder Ihre Kommilitonen um Hilfe bitten.

Bibliotheken für dieses Buch

Ich habe ein paar einfache PYTHON-Bibliotheken, die in verschiedenen Kapiteln erwähnt werden, speziell für dieses Buch geschrieben. Um diese zu verwenden, laden Sie sich das Archiv herunter, das Sie unter der URL `http://weitz.de/files/PythonLibs.zip` finden. In diesem Archiv befinden sich mehrere Dateien mit der Endung `.py`. Es gibt zwei Möglichkeiten, diese Dateien zu verwenden:

(i) Legen Sie die Dateien in den Ordner, in dem sich das JUPYTER-Notebook befindet, mit dem Sie gerade arbeiten.

Um diesen Ordner zu finden, geben Sie den folgenden Befehl in Jupyter ein:

```
!cd
```

(Wenn Sie Linux oder einen Mac verwenden, geben Sie stattdessen `!pwd` ein.)

(ii) Der Nachteil der obigen Methode ist, dass sie nur für Notebooks im selben Ordner funktioniert.

Wenn Sie den Code von *allen* Notebooks aus verwenden wollen, dann müssen Sie zur Datei `ipython_config.py` (siehe Seite 634) den folgenden Code hinzufügen:

```
c.InteractiveShellApp.exec_lines = [
  'import sys; sys.path.append("C:\\Pfad\\zum\\Ordner")'
]
```

Dabei muss `C:\Pfad\zum\Ordner` durch einen Ort ersetzt werden, an dem auf Ihrer Festplatte die Dateien aus dem obigen Archiv zu finden sind.

Unabhängig von der gewählten Methode müssen Sie zur Verwendung des besagten Codes im jeweiligen Notebook die benötigten Bibliotheken noch importieren. Darauf wird im Buch ggf. nicht immer wieder explizit hingewiesen.

Achtung: Sämtlicher Code von mir, der sich im oben beschriebenen Archiv befindet, ist relativ einfach gehalten und nur für Demonstrationszwecke im Rahmen des Buches gedacht. Er eignet sich *nicht* für den produktiven Einsatz in „ernsthaften“ Anwendungen.

Die Canvas-Bibliothek

```
from canvas import Canvas
```

Diese Bibliothek stellt ein paar primitive Funktionen zum Zeichnen von geometrischen Objekten wie Punkten, Vektoren, Strecken, Geraden und Zirkeln dar. Weitere Informationen finden Sie im Buchtext ab Kapitel 30.

Die Vektor-Bibliothek

```
from vectors import Vector
```

Mit einer Anweisung wie `Vector(3,-4)` oder `Vector(1,0,3)` können Sie einen *Vektor* (technisch ein Python-*Objekt* der *Klasse* `Vector`) erzeugen. Solche Vektoren können mit `+` und `-` ganz normal (wie Zahlen in Python) addiert und subtrahiert werden. Ebenso kann man sie mit Zahlen multiplizieren oder durch diese dividieren (skalare Multiplikation).

Multipliziert man zwei Vektoren mit `*`, so erhält man das Skalarprodukt. „Multipliziert" man sie mit `%`, so erhält man das Vektorprodukt. (Letzteres funktioniert natürlich nur mit Vektoren, die drei Komponenten haben.) Ist `v` ein Vektor, so berechnet `v.norm()` dessen Norm.

Vektoren können außerdem in vielen Situationen wir Python-Tupel behandelt werden. So kann man z.B. mit `v[0]` die erste Komponente des Vektors `v` auslesen; wie bei einem Tupel kann man die Komponenten eines Vektors allerdings nicht ändern.

Weitere Informationen zu dieser Bibliothek finden Sie am Anfang der Datei `vectors.py`.

Die Matrix-Bibliothek

```
from matrices import Matrix
```

Ergänzend zu `vectors.py` gibt es auch eine Bibliothek `matrices.py`, mit der Sie Objekte der Klasse `Matrix` erzeugen und mit ihnen rechnen können. Diese Objekte sollen natürlich Matrizen darstellen und sie können mit den eben beschriebenen Vektoren interagieren. Auch hier finden Sie ausführlichere Informationen in der Datei selbst, aber die wichtigsten Funktionalitäten sollen hier kurz vorgestellt werden.

Zum Erzeugen einer Matrix können Sie zum Beispiel einen Befehl wie `Matrix([[1,2],[3,4]])` verwenden. Die einzelnen Listen werden dabei als *Zeilen* der Matrix interpretiert. Alternativ können Sie eine Matrix durch z.B. `Matrix([v1,v2,v3])` erzeugen, wobei `v1` bis `v3` Vektoren sind, die dann die *Spalten* der Matrix werden.

Man kann auf einzelne Komponenten einer Matrix `M` mit der Syntax `M[1,2]` zugreifen, aber auch auf ganze Zeilen oder Spalten (als Listen) mit `M.row(1)` oder `M.col(2)`.

Wie bei Vektoren können auch Matrizen mit anderen Matrizen bzw. mit Zahlen verknüpft werden, sofern die Verknüpfung Sinn ergibt, indem man die PYTHON-Operatoren für die vier Grundrechenarten verwendet. Auch die Multiplikation von Matrizen mit Vektoren ist selbstverständlich möglich.

Schließlich haben Matrizen Methoden wie `transpose`, `invert` und `determinant`, die jeweils das tun, was der Name erwarten lässt.

Arbeiten mit homogenen Koordinaten

```
from homog import *
```

Eine kleine Bibliothek `homog.py` enthält ein paar Hilfsfunktionen für homogene Koordinaten.[2] Wir werden diese Funktionen in Kapitel 35 verwenden.

[2] Siehe dazu Kapitel 34.

Plotten von Funktionen

```
from plot import *
```

Diese Bibliothek stellt verschiedene Methoden zum Zeichnen von Funktionen zur Verfügung. Sie wird in Kapitel 40 näher erläutert.