Prof. Dr. Ulrich Rüde, Dominik Thönnes

Algorithmik kontinuierlicher Systeme
Aufgabenblatt 1 — Einführung in Python und NumPy

Allgemeines:

- Die Abgabe der Programmieraufgaben erfolgt über StudOn, es handelt sich um **Einzelabgaben**.
- Bitte ändern Sie nicht die Namen der hochzuladenen Dateien!
- Sie können während der Bearbeitungszeit Ihre Abgaben im StudOn beliebig oft aktualisieren. Nur die aktuellste Abgabe, die in der Bearbeitungszeit hochgeladen wurde, wird gewertet.
- Auf der Übungswebsite finden Sie zu jedem Übungsblatt eine Vorlage, sowie zu jeder Aufgabe eine Datei `*_test.py` mit der Sie ihre Lösungen jederzeit selbst überprüfen können. Die Tests lassen sich pro Teilaufgabe separat ausführen. Um nur die Tests für die Teilaufgaben x), y), z) auszuführen, rufen Sie `python3 *_test.py x y z` auf.
- Damit Sie auf eine Teilaufgabe Punkte bekommen, muss sie mit Python 3.11 im CIP Pool funktionieren. Das bestehen der mitgelieferten Tests ist notwendig, aber nicht hinreichend dafür, dass Sie Punkte bekommen. Auf die Rechner des CIP Pools können Sie mittels SSH zugreifen (siehe `https://wwwcip.informatik.uni-erlangen.de/documentation/services.de.html`).

Auf diesem Aufgabenblatt lernen Sie Eigenschaften der Sprache Python und das Paket NumPy kennen. Sollten Sie noch keine Erfahrung mit Python haben, empfehlen wir Ihnen, sich parallel zu diesem Blatt die Aufzeichnung der Python-Einführungsvorlesung aus dem Jahr 2022 anzusehen (`https://www.fau.tv/clip/id/41721`, Zugriff per Idm-Login).

Aufgaben 1 bis 5 befassen sich jeweils mit einem grundlegenden Aspekt von Python. Sie werden feststellen, dass Python viele Dinge bereits von Grund auf beherrscht, ohne dabei `import`-Statements zu benötigen. Sie brauchen daher keinerlei Bibliotheken, um diese Aufgaben zu lösen. In Aufgaben 6 und 7 beschäftigen Sie sich mit grundlegenden Funktionen der Bibliothek NumPy für numerisches Rechnen. NumPy bildet die Grundlage für alle weiteren Programmieraufgaben, daher raten wir Ihnen, sich diesen Aufgaben gewissenhaft zu widmen.

Achtung: In Python's Syntax spielt Einrückung eine wichtige Rolle, und Python reagiert empfindlich auf das Mischen von Leerzeichen und Tabulatoren bei der Einrückung. Achten Sie daher darauf, Code *stets mit vier Leerzeichen* einzurücken! Richten sie ggf. ihren Editor so ein, dass dieser beim Drücken von `TAB` automatisch vier Leerzeichen erzeugt.

Zum Programmieren im CIP empfehlen wir den Editor *VS Code*. Sie können die Dateien dieser Übung in VS Code öffnen, indem Sie in einer Konsole in den extrahierten Order `Blatt01` navigieren und dort folgende Befehle ausführen (beachten Sie den Punkt am Ende):

```
module load vscode
code .
```

Ihren Code, sowie die Testfälle, können Sie auf der Konsole mittels des Befehls `python3` ausführen; zum Beispiel für die Tests zu Aufgabe 2:

```
python3 integers_test.py
```

Aufgabe 1 — Hallo, Python! (1 Punkt) `hallo_welt.py`

Bei dieser Aufgabe geht es um das erste Standardbeispiel, wenn Sie eine neue Sprache lernen: „Hallo, Welt!“.

a) Hallo, Welt! Schreiben Sie eine Funktion `hallo_welt`, welche den obigen String `Hallo, Welt!` auf der Kommandozeile ausgibt.

b) Hallo, Lisa! Schreiben Sie nun eine weitere Funktion `hallo_name`, welche einen Parameter `name` erwartet, und dann den String `Hallo, <name>!` auf der Kommandozeile ausgibt.

c) Main-Funktion Schreiben Sie nun eine „Main“-Funktion, welche ausgeführt wird, wann immer das Programm/die Datei von der Kommandozeile direkt aufgerufen wird. In dieser Funktion sollen beide Funktionen `hallo_welt` und `hallo_name` aufgerufen werden. Die Übergabeparameter sind an dieser Stelle egal.
Wichtig: Wird eine Python-Datei als Modul geladen, so wird der darin enthaltene, nicht in Funktionen gekapselte, Code sofort von Oben nach Unten ausgeführt. Sorgen Sie also dafür, dass Ihre `main`-Funktion nicht beim Importieren der Datei aufgerufen wird, sondern nur, wenn Sie auf der Kommandozeile `python3 hallo_welt.py` schreiben!

Aufgabe 2 — Zahlen 101: Natürliche Zahlen (1 Punkt) `integers.py`

a) Primzahlen Schreiben Sie eine Funktion `is_prime`, die entscheidet, ob eine gegebene Zahl eine Primzahl ist.

b) Zahlendarstellung Schreiben Sie eine Funktion `int2str`, die eine gegebene natürliche Zahl in eine Zeichenkette umwandelt. Das zweite Argument beschreibt dabei die Basis, in der die Zahl dargestellt werden soll.

c) Mirpzahlen Schreiben Sie eine Funktion `is_emirp`, die entscheidet, ob eine gegebene Zahl eine Mirpzahl ist. Eine Mirpzahl ist eine Primzahl, deren Spiegelbild (in Dezimaldarstellung) eine **andere** Primzahl ist, z.B. die Zahlen 13 und 31. Sie können dazu natürlich die Funktionen der beiden vorherigen Aufgaben verwenden.

Aufgabe 3 — Zahlen 102: Gleitkommazahlen (1 Punkt) `float.py`

In dieser Aufgabe werden Sie mit Gleitkommazahlen arbeiten. Im Vergleich zu ganzen Zahlen werden Gleitkkommazahlen durch eine feste Anzahl Bits repräsentiert. Python verwendet standardmäßig Gleitkommazahlen vom Typ *double* gemäß dem IEEE 754 Standard. Computer können sehr effizient mit solchen Zahlen arbeiten. Der Preis dafür ist, dass bei jeder Rechenoperation Genauigkeit verloren geht. Sie werden in den einzelnen Aufgaben merken, wie stark der Genauigkeitsverlust ist und wie man diesem entgegenwirkt.
Wichtig: Importieren Sie **keine** Bibliotheken, auch nicht aus der Standardbibliothek!

a) Exponentialfunktion Die Exponentialfunktion $\exp(x) = e^x$ wird vielfältig in der Mathematik und in der Elektrotechnik eingesetzt. Eine Möglichkeit deren Funktionswerte auszurechnen ist über die Taylor-Reihe der Funktion:

$$\exp(x) \;=\; e^x \;=\; \sum_{k=0}^{\infty} \frac{x^k}{k!} = \frac{1}{1} + \frac{x}{1} + \frac{x^2}{2} + \frac{x^3}{6} + \cdots \tag{1}$$

Schreiben Sie eine Funktion `my_exp`, welche die Exponentialfunktion gemäß der obigen Formel approximiert. Brechen Sie die Iteration ab, sobald die obige Reihe konvergiert, sprich sich der Wert in einem Iterationsschritt nicht mehr ändert.
Zusatzfrage: Was passiert, wenn Sie `my_exp(-25)` ausführen? Gibt es hier ein Problem? Wie könnte man das eventuell auftretende Problem lösen?

b) Numerische Differentiation Die Ableitung einer Funktion f an einer Stelle x ist definiert durch den Grenzwert

$$f'(x) \;:=\; \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}. \tag{2}$$

Analytisch ist es nun meistens „einfach“ diesen Grenzwert über die geltenden Ableitungsregeln auszurechnen. In der Praxis haben Sie allerdings oftmals nicht eine analytische Darstellung ihrer Funktion, sondern nur Schätzwerte und brauchen trotzdem einen Wert der Ableitung. In solchen Fällen hilft die numerische Differentation. Man kann den Wert der Ableitung approximieren, indem man in Formel (2) nun ein fixes, allerdings sehr kleines h einsetzt, das heißt

$$f'(x) \;\approx\; \frac{f(x+h) - f(x)}{h} \tag{3}$$

Schreiben Sie eine Funktion `diff1`, die eine Funktion f, einen Wert x und den Parameter h übergeben bekommt, und mit dieser Methode die Ableitung einer Funktion abschätzt. Das Verfahren heißt das Verfahren der Vorwärtsdifferenzen und gehört zu den Methoden der finiten Differenzen zum Abschätzen von Ableitungen.
Eine genauere Methode zur numerischen Abschätzung der Ableitung ist es, Formel (3) sowohl mit einem Parameter h, als auch mit dem Parameter $-h$ auszuwerten, und daraus den Mittelwert zu bilden. Der resultierende Ausdruck lässt sich wie folgt vereinfachen:

$$f'(x) \;\approx\; \frac{\frac{f(x+h)-f(x)}{h} + \frac{f(x-h)-f(x)}{-h}}{2} = \frac{f(x+h) - f(x-h)}{2h} \tag{4}$$

Schreiben Sie nun auch eine Funktion `diff2`, die die Ableitung mithilfe von Formel (4) berechnet. Dieses Verfahren heißt das Verfahren der zentralen Differenzen und gehört ebenfalls zu den Methoden der finiten Differenzen.
Sie bekommen an dieser Stelle bereits einen Standardwert für h übergeben, müssten diesen also nicht wirklich selbst angeben. Probieren Sie trotzdem verschiedene Werte für h in beiden Funktionen aus und berechnen Sie die Fehler der beiden Methoden für verschiedene h bei Funktionen, deren Ableitungswerte Sie bereits kennen! Was stellen Sie fest? Ist es wirklich immer besser ein $h \approx 0$ zu wählen, oder kann das h auch „zu klein“ sein?

c) Quadratwurzeln (*schwer*) Die letzte Teilaufgabe ist die schwierigste von den bisherigen Aufgaben, gleichzeitig allerdings auch die belohnenste, wenngleich sie auf den ersten Blick nicht so aussieht. Ziel ist es die Quadratwurzel einer positiven Gleitkommazahl `x` mittels der Newton-Iteration zu berechnen. Generell gilt, dass sich die Quadratwurzel $\sqrt{x}$ einer positiven reellen Zahl x durch den Grenzwert der Folge

$$\sqrt{x} \;=\; \lim_{n \to \infty} \frac{1}{2} \cdot \left(x_n + \frac{x}{x_n} \right)$$

mit $x_1 = x$ bestimmen lässt.
Schreiben Sie nun eine Funktion `sqrt`, welche eine positive Gleitkommazahl `x` erwartet und die Quadratwurzel dieser Zahl über die Iteration

$$x_1 = x$$
$$x_{n+1} = \frac{1}{2} \cdot \left(x_n + \frac{x}{x_n} \right)$$

ausrechnet, wobei man so lange iteriert, wie `abs(x_n**2 - x) > x * epsilon`. Das ε sollte dabei klein genug gewählt werden, beispielsweise $\approx 10^{-15}$. Terminiert Ihre Funktion für alle positive Gleitkommazahlen?

Aufgabe 4 — Zeichenketten in Python (1 Punkt) `strings.py`

a) Enthält meine Zeichenkette ein bestimmtes Zeichen? Schreiben Sie eine Funktion `contains_char`, welche einen String `string` und einen String `c` der Länge 1 übergeben bekommt und mittels `True` resp. `False` entscheidet, ob das Zeichen `c` in der Zeichenkette `string` enthalten ist.
Was müssten Sie ändern, wenn die Längenbeschränkung von `c` nicht mehr gölte?

b) „Eine güldne, gute Tugend: Lüge nie!“ Schreiben Sie eine Funktion `is_palindrom`, welche einen String `string` übergeben bekommt und entscheidet, ob der String `string` ein Palindrom ist, sprich von vorne und hinten gelesen identisch ist. Wir betrachten in unserem Fall für Palindrome **keine** Leerzeichen, sprich diese Zeichen sind davor aus der Zeichenkette zu entfernen.

c) Zeichenfrequenzen Schreiben Sie eine Funktion `count_char_frequency`, welche einen String `string` übergeben bekommt und ein Wörterbuch (`dict`) von den Zeichen im String und ihrer Häufigkeit zurückgibt.

d) Das erste sich nicht wiederholende Zeichen Schreiben Sie eine Funktion `first_non_repeating_char`, welche einen String `string` und einen optionalen Wahrheitswert `repeating` übergeben bekommt und das erste sich in der restlichen Zeichenkette nicht wiederholende (bei `repeating == False`) oder wiederholende (bei `repeating == True`) Zeichen zurückgibt.

e) Rotierende Zeichenketten Schreiben Sie eine Funktion `rotate_string`, welche einen String `string`, sowie zwei Zahlen `left_rot` und `right_rot`, und anschließend den String um `left_rot` Zeichen nach links und `right_rot` Zeichen nach rechts rotiert und zurückgibt.

f) Rotationsäquivalenz Schreiben Sie eine Funktion `rotationally_equivalent`, welche zwei Zeichenketten `string1` und `string2` übergeben bekommt und überprüft, ob diese rotationsäquivalent sind. Zwei Zeichenketten `a` und `b` heißen *rotationsäquivalent*, wenn der String `b` aus einer beliebigen Rotation von `a` entsteht.

Aufgabe 5 — Listen in Python (1 Punkt) `lists.py`

In dieser Aufgabe werden Sie mit einigen Eigenheiten von Listen konfrontiert. Sollten Sie Probleme mit der Darstellung von Listen im Speicher haben, so empfehlen wir Ihnen einen Blick auf die Seite `http://www.pythontutor.com/`, auf welcher Sie Python-Code ausführen und sich dabei die Strukturen im Speicher anzeigen lassen können. Passen Sie allerdings bitte auf, dass die dortige Python-Version nur 3.6 ist, wir allerdings mit einer neueren Version arbeiten.
Das Ziel dieser Aufgabe ist es Ihnen sogenannte *for comprehensions* ans Herz zu legen. So ist jede einzelne dieser Teilaufgaben mit nur **einer** Zeile lösbar!

a) Erstes und letztes Element Schreiben Sie eine Funktion `first_and_last_element`, welche eine Liste `l` übergeben bekommt und überprüft, ob das erste und letzte Element der Liste übereinstimmen.

b) Alle geraden Elemente Schreiben Sie eine Funktion `get_all_even_elements`, welche eine Liste `l` und einen Index `start` übergeben bekommt, und eine Liste aller Elemente ab dem Index `start` mit geradem Index zurückgibt.

c) Letzten zwei Elemente Schreiben Sie eine Funktion `get_last_two_elements`, welche die letzten beiden Elemente einer übergebenen Liste zurückgibt. Sie können davon ausgehen, dass die übergebene Liste mindestens zwei Elemente beinhaltet.

d) Quadriere alle Elemente Schreiben Sie eine Funktion `square_elements`, welche eine Liste von Zahlen `number_list` übergeben bekommt, und eine Liste zurückgibt mit den quadrierten Zahlen.

e) Filtere Elemente Schreiben Sie eine Funktion `filter_elements`, welche zwei Listen `list_one` und `list_two` übergeben bekommt, und eine Liste von den Elementen in `list_one` zurückgibt, welche ebenso in `list_two` enthalten sind.

f) Elemente an gewissen Indizes Schreiben Sie eine Funktion `select_elements`, welche eine Liste `l` und beliebig viele Indizes `indices` übergeben bekommt, und alle Elemente aus `l` in einer Liste zurückgibt, wenn ihr Index in `indices` enthalten ist.

Aufgabe 6 — Lineare Algebra mit NumPy (6 Punkte) `matrices.py`

Viele bedeutsame mathematische Transformationen auf Vektorräumen sind linear, und lineare Operatoren auf endlichdimesionalen Vektorräumen lassen sich stets eindeutig durch eine Matrix beschreiben. In dieser Aufgabe geht es daher um einfache Manipulationen von Matrizen, dargestellt als NumPy-Arrays.
NumPy-Arrays (`numpy.ndarray`) sind mehrdimensionale Arrays fester Größe. Bevor Sie mit der Bearbeitung der Aufgaben beginnen, werfen Sie einen Blick auf die NumPy-Dokumentationsseiten `https://numpy.org/doc/stable/user/basics.creation.html` und `https://numpy.org/doc/stable/reference/routines.html`. Für viele Probleme bietet NumPy bereits eine Lösung, und mit einer kurzen Suche in der Dokumentation können Sie sich häufig viel Programmieraufwand ersparen.

a) Rotationsmatrix Schreiben Sie eine Funktion `rotation_matrix`, die zu einem gegebenen Winkel ω eine 2×2 Rotationsmatrix R zurückgibt, so dass $R \cdot x$ einen zweidimensionalen Vektor x um ω gegen den Uhrzeigersinn dreht.

b) Spiegelungsmatrix Schreiben Sie eine Funktion `reflection_matrix`, die zu einem gegebenen Winkel ω eine 2×2 Spiegelungsmatrix S zurückgibt, so dass $S \cdot x$ einen zweidimensionalen Vektor x an der Gerade durch den Ursprung mit Winkel ω zur Ordinate spiegelt.

c) Einheitsmatrix — Rechteckig Schreiben Sie eine Funktion `eye`, welche zwei Parameter n und m erwartet und eine $n \times m$ Matrix mit 1en auf der Diagonale zurückgibt.

d) Komposition Schreiben Sie eine Funktion `compose`, die beliebig viele Matrizen $A_1, \ldots, A_n$ mit den Dimensionen $d_1 \times d_2,\ d_2 \times d_3,\ \ldots,\ d_n \times d_{n+1}$ übergeben bekommt und diese zu einer einzelnen Matrix B mit der Dimension $d_1 \times d_{n+1}$ zusammenfügt, so dass für alle d_{n+1}-dimensionalen Vektoren x gilt $A_1 \cdot A_2 \cdot \ldots \cdot A_n\, x = Bx$.

e) Antidiagonalmatrix Eine Matrix $A \in \mathbb{R}^{n \times n}$ heißt *antidiagonal*, wenn für ihre Einträge gilt:

$$A_{ij} = 0 \quad \text{für } i + j \neq n + 1$$

Damit wird eine antidiagonale $n \times n$-Matrix durch genau n Einträge bestimmt. Schreiben Sie eine Funktion `antidiag`, die aus einer Liste an d Werten eine antidiagonale $d \times d$-Matrix mit diesen Werten erstellt.

f) Vandermondematrix Eine wichtige Matrix im Bereich der Polynominterpolationen ist die sogenannte Vandermondematrix V, welche Sie in einer späteren Übung noch näher kennenlernen werden. Für gegebene

Stützstellen $x_1, \ldots, x_m$ mit $m \in \mathbb{N}$ ist die Vandermondematrix V durch

$$V = \begin{pmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{m-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{m-1} \\ 1 & x_3 & x_3^2 & \ldots & x_3^{m-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \ldots & x_m^{m-1} \end{pmatrix}$$

gegeben. Schreiben Sie eine Funktion `vandermonde_matrix`, welche eine Liste an unterschiedlichen Stützstellen $x_1, \ldots, x_m$ erwartet und die zugehörige Vandermondematrix V zurückgibt.

g) Kronecker-Matrix-Produkt Sei $A \in \mathbb{R}^{m \times n}$ und $B \in \mathbb{R}^{p \times q}$, so bezeichnet das Kronecker-Matrix-Produkt $A \otimes B$ die folgende $pm \times qn$-Blockmatrix:

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix}$$

Schreiben Sie eine Funktion `kronecker_product`, welche zu zwei Matrizen A und B ihr Kronecker-Matrix-Produkt $A \otimes B$ zurückgibt.

h) Walsh Matrix Schreiben Sie eine Funktion `walsh_matrix`, die zu einem gegebenen $n \in \mathbb{N}$ eine $2^n \times 2^n$-Walsh-Matrix W_n zurückgibt, welche nach folgendem Schema aufgebaut sind:

$$W_0 = \begin{pmatrix} 1 \end{pmatrix} \qquad \text{und} \qquad W_n = \begin{pmatrix} W_{n-1} & W_{n-1} \\ W_{n-1} & -W_{n-1} \end{pmatrix} = W_1 \otimes W_{n-1},$$

wobei $\otimes$ das Kronecker-Matrix-Produkt darstellt.

Aufgabe 7 — Game of Life (4 Punkte) gameoflife.py

Ziel dieser Aufgabe ist es, Conway's Game of Life mithilfe von NumPy-Arrays zu implementieren. Dabei handelt es sich um einen einfachen Zellulären Automaten mit zwei Zuständen pro Zelle. Trotz seiner Einfachheit erzeugt der Automat erstaunliche und kaum vorhersehbare Effekte. Eine detaillierte Beschreibung des Game of Life finden Sie hier: `https://en.wikipedia.org/wiki/Conway's_Game_of_Life`.

a) Initialisierung Das Gitter auf dem das Game of Life ausgeführt wird, stellen wir als NumPy Arrays aus Nullen und Einsen dar (Verwenden sie als Elementtyp `dtype=bool`).
Hinweis: Passen Sie in Python mit Bool'schen Werten auf, so ist hier `True + True != 1`, sondern `2`.
Implementieren Sie nun die Funktion `add_entity`, die ein Objekt (Numpy Array aus Nullen und Einsen) an die spezifizierte Stelle des Gitters schreibt, sodass der Wert des Objekts an der Stelle $[0, 0]$ auf der Gitterkoordinate $[x, y]$ liegt. Sie dürfen dabei annehmen, dass an der spezifizierten Stelle im Gitter genug Platz für das Objekt ist.

b) Zeitschritt Nun geht es an die eigentlichen Spielregeln. Implementieren Sie dazu die Funktion `next_step`, die für ein gegebenes Gitter das zugehörige Gitter des nächsten Zuges berechnet. Dafür gelten folgende Regeln:

1. Eine tote Zelle mit genau drei lebendigen Nachbarn erwacht zum Leben (d.h., wird auf den Wert 1 gesetzt).
2. Eine lebendige Zelle mit weniger als zwei lebendigen Nachbarn stirbt (d.h., wird auf den Wert 0 gesetzt).
3. Eine lebendige Zelle mit zwei oder drei lebendigen Nachbarn lebt weiter.

4. Eine lebendige Zelle mit mehr als drei lebenden Nachbarn stirbt.

Die Randbedingungen seien dabei periodisch, d.h., eine Zelle am oberen Rand hat als oberen Nachbarn die zugehörige Zelle am unteren Rand. Die anderen drei Randfälle seien analog. Wenn Sie dies richtig implementieren, sollte ein Gleiter in der Lage sein über Ränder zu fliegen.