Python Übungen

I. Listen und list-comprehension

1. Initialisieren Sie eine leere Liste `my_list`. Nutzen Sie eine `for`-Schleife und die `append()` Methode, um die Elemente i^2 für $i = 1, \dots, 9$ der Liste hinzuzufügen. Geben Sie die fertige Liste mittels `print()` auf dem Bildschirm aus.
2. Nutzen Sie anstatt der obigen Vorgehensweise list comprehension, um die gleiche Liste zu erzeugen.
3. Nutzen Sie `print()` und Indizierung, um nur die ersten zwei Listenelemente auszugeben.
4. Nutzen Sie `print()` und Indizierung, um nur die letzten zwei Listenelemente auszugeben. Nehmen Sie an, dass die Anzahl der Listenelemente unbekannt ist, sodass `my_list[8:10]` nicht möglich ist.
5. Nutzen Sie `print()` und Indizierung, um jedes zweite Listenelemente auszugeben.
6. Nutzen Sie list-comprehension um eine neue Liste `new_list` aus `my_list` zu konstruieren. Jeder Eintrag in `new_list` soll um eins größer sein als in `my_list`.

II. Dictionaries

1. Erzeugen Sie ein leeres dictionary `params`. Erzeugen Sie dann drei Einträge. Nutzen Sie als Keys die Strings `'a0'`, `'a1'` und `'a2'` und als Values jeweils 1.5, -1.0 und 1.0.
2. Nutzen Sie nun stattdessen dict-comprehension, um das gleiche dictionary zu konstruieren. Definieren Sie hierfür zunächst zwei Listen, welche jeweils Keys und Values beinhalten.
3. Iterieren Sie nun mit einer `for`-Schleife durch `params` und lassen Sie sich für jeden Eintrag Key und korrespondierenden Value anzeigen, sodass die folgende Ausgabe auf dem Bildschirm zu sehen ist:

```
a0 :  1.5
a1 :  -1
a2 :  1.0
```

III. Funktionen

1. Schreiben Sie die Funktion `eval_quadr_poly`. Diese soll das quadratische Polynom $y = a_0x^0 + a_1x^1 + a_2x^2$ an gegebenen Stellen x auswerten und die korrespondierenden y-Werte in einer Liste zurückgeben. Die Funktion erwartet als Argumente eine Liste x_values mit x-Werten sowie Floats `a0`, `a1`, `a2` für die Parameter a_0, a_1, a_2.

```
def eval_quadr_poly(x_values,a0, a1, a2):

return y
```

2. Werten Sie die Funktion an den Stellen $x = 1, \dots, 9$ aus. Wählen Sie $a_0 = 1.5,\ a_1 = -1.0,\ a_2 = 1.0$ als Funktionsparameter.
3. Schreiben Sie eine Lambda-Funktion, welche die gleiche Operation ausführt und evaluieren Sie diese ebenfalls an den Stellen $x = 1, \dots, 9$, um deren Richtigkeit zu überprüfen.
4. Definieren Sie nun eine weitere Funktion `eval_n_poly`, welche ein Polynom beliebigen Grades n evaluiert, d.h. $y = a_0x^0 + a_1x^1 + \cdots + a_nx^n$. Nutzen Sie hierfür den `*args` Parameter

```
def eval_n_poly(x_values,*args):

…

return y
```

5. Ändern Sie die Funktion `eval_n_poly` so ab, dass sie nun Keyword Arguments `**kwargs` erwartet. Nutzen Sie die eindeutige Zuordnung von Key und Value, sodass `eval_n_poly` invariant gegenüber der Reihenfolge ist, in welcher die Funktionsparameter in **kwargs auftauchen.

```
def eval_n_poly(x_values,*kwargs):

    …

    return y
```

IV. Klassen und Methoden

1. Definieren Sie eine Klasse `poly_model`. Die Klasse sollte ein Dictionary `params` als Attribut haben, welches die Funktionsparameter eines quadratischen Polynoms $a_0, \dots, a_n$ als Key-Value Paare beinhaltet. Die Klasse sollte außerdem über eine Methode verfügen, welcher x-Werte in einer Liste als Argument übergeben werden und welche die korrespondierenden y-Werte als Liste zurückgibt.

```
class poly_model():

def __init__(self,params):

…

def evaluate(self,x_values):

…
return y
```

2. Erzeugen Sie mehrere Instanzen der Klasse `poly_model` mit unterschiedlichen Funktionsparametern und werten Sie diese für $x = 1, \dots, 9$ aus.

3. Versuchen Sie auch die Funktionsparameter nach der Erzeugung einer Instanz zu ändern, und untersuchen Sie, ob sich die Ausgabe der `evaluate()`-Methode wie erwartet ändert.

V. Plotten mit Matplotlib

1. Importieren Sie das Modul `pyplot` der Bibliobliothek `matplotlib` wie folgt:

```
import matplotlib.pyplot as plt
```

2. Nehmen Sie an, dass folgende Messungen eines realen Prozesses mit Eingangsgröße x and Ausgangsgröße y vorliegen:

```
x_meas = [0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5]
```

```
y_meas = [-1.71, 1.87, 1.70, 0.71, 0.67, 0.34, -0.55, 2.42, 5.40, 7.65]
```

3. Nutzen Sie die `plot()`-Methode des Moduls `pyplot`, um die Messdaten in einem x-y-Diagramm zu plotten.
4. Erzeugen Sie eine Instanz der Klasse `poly_model` mit Funktionsparametern `{'a0':-0.5,'a1':4.0,'a2':-2.0,'a3':0.4}` und werten Sie deren `evaluate()`-Methode an den Messpunkten `x_meas` aus.
5. Plotten Sie die Ausgabe des Modells im gleichen Diagramm wie die Messdaten und beurteilen Sie visuell, ob das Modell die Messdaten gut widergibt.

VI. Arrays and matrices with numpy

1. Importieren Sie die Bibliothek `numpy` wie folgt:

```
import numpy as np
```

2. Arrangieren Sie die gemessenen Werte der Ausgangsgröße in einem 10×1 `numpy` array `Y`

```
Y = np.array(…)
```

Nutzen Sie `Y.shape`, um zu prüfen, ob das Array die gewünschten Dimensionen hat.

3. Konstruieren Sie ein 10×4 `numpy` array `X`, dessen Spalten die Eingangsgröße `x_meas` in nullter bis dritter Potenz enthält, d.h. $X = [x^0, x^1, x^2, x^3]$. Hinweis: Nutzen Sie den `**` Operator und `np.hstack()`. Geben Sie das resultierende Array mittels `print()` aus und prüfen Sie das Ergebnis.
4. Definieren Sie die Funktion `linear_least_squares`. Diese sollte als Argumente eine Instanz der Klasse poly_model, eine Liste mit gemessenen Werten der Eingangsgröße sowie eine Liste mit gemessenen Werten der Ausgangsgröße erwarten.

```
def linear_least_squares(model,x,y):

…

return model
```

Innerhalb der Funktion sollten die Arrays `X` und `Y` wie in Aufgabe 2 und 3 konstruiert werden. Anschließend soll die folgende Rechenoperation ausgeführt werden, deren Ergebnis die optimalen Funktionsparameter a_0, a_1, a_2, a_3 sind.

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = (X^T X)^{-1} X^T Y.$$

Hinweise:

- Die Transponierte eines `numpy` Arrays ist `X.T`
- Das Skalarprodukt zweier `numpy` Arrays `A`, `B` ist `A.dot(B)`
- Die Inverse eines `numpy` Arrays `A` kann berechnet werden via `np.linalg.inv(A)`

Schließlich sollen die berechneten Parameter dem Attribut `params` der Instanz zugewiesen und die veränderte Instanz zurückgegeben werden.

6. Rufen Sie die Funktion `linear_least_squares` nun mit der im vorigen Abschnitt erzeugten Instanz der Klasse `poly_model` sowie den gegebenen Messdaten `x_meas` und `y_meas` auf.
7. Evaluieren Sie nun das optimierte Modell an den Messpunkten `x_meas` und inspizieren Sie visuell, ob das optimierte Modell die Messdaten nun besser widergibt.