Institut für Mechanik und Adaptronik

Prof. Dr.-Ing. M. Sinapius

Simulation technischer Systeme mit Python

Dr. Ing. H. Schlums
M. Sc. H. Traub
Dr. Ing, C- Pommer
M. Sc. P. Scholle

Abschlussklausur – SoSe 21

Nachname :

Vorname :

Hinweise zur Klausur

Die Klausur ist bestanden, wenn mindestens 50% der maximal erreichbaren Punkte erzielt werden. Zur Bearbeitung der Klausur stehen 90 Minuten Zeit zur Verfügung. In jeder Abbildung müssen alle Achsenbeschriftungen sinnvoll gewählt werden. Bitte denken Sie daran die „Erklärung Online-Prüfung" auszufüllen und nach der Klausur hochzuladen. Die Abgabe erfolgt über den StudIP Ordner „Abgabe Take Home Exam".

Hilfsmittel: ***Die Klausur muss selbstständig angefertigt werden. Es dürfen keine großen Code-Blöcke per copy/paste eingefügt werden. Alle weiteren Hilfsmittel sind erlaubt***

Aufgabe:	**1**	**2**	**3**	**4**	$\sum$	**Bestanden**
Punkte:						**Ja** ☐
Korrektor:						**Nein** ☐
						Note:

Aufgabe 1: Curve Fitting (10 Punkte)

Gegeben sind einige Messdaten von einer Kennlinienbestimmung einer mechanischen Feder in der Datei „federkennlinie_klausur.csv“. In der ersten Spalte ist darin der Weg in meter, in der zweiten Spalte die gemessene Kraft F_c in Newton gegeben.

1. Laden Sie die gegebenen Messwerte in 2 numpy-arrays und plotten Sie Kraft über Weg mit matplotlib mit grünen Kreuzen.
2. Die dargestellten Messwerte können mit folgender Funktion angenähert werden

$$f(x, a, b) = \frac{x}{a} \cdot |\tanh\left(\frac{x}{b}\right)|$$

   Definieren sie diese Formel als Python Funktion. Ermitteln Sie optimale Werte für die Parameter a und b mit scipy. Stellen Sie die optimale Lösung in der figure aus Aufgabenteil 1 mit einer Strichpunktlinie dar. Erstellen sie unten rechts im Bild eine geeignete legend um beide Datenreihen auseinander halten zu können. Speichern Sie die optimalen Parameter mithilfe von Python in einer Textdatei.
3. Erstellen Sie mithilfe des „matplotlib.pyplot.text“ Befehls eine Textbox in der die optimalen Parameter stehen. Beide Parameter sollen als Fließkommazahl mit einer Nachkommastelle angezeigt werden. Die Textbox soll oben links im Bild angezeigt werden. Das Ergebnis soll so aussehen, wobei anstelle von „xyz“ die Parameter angezeigt werden sollen:

```
a=xyz
b=xyz
```

Aufgabe 2: Gewöhnliche Differentialgleichung (20 Punkte)

Die folgende Aufgabe beschäftigt sich mit diesem schwingungsfähigen System:

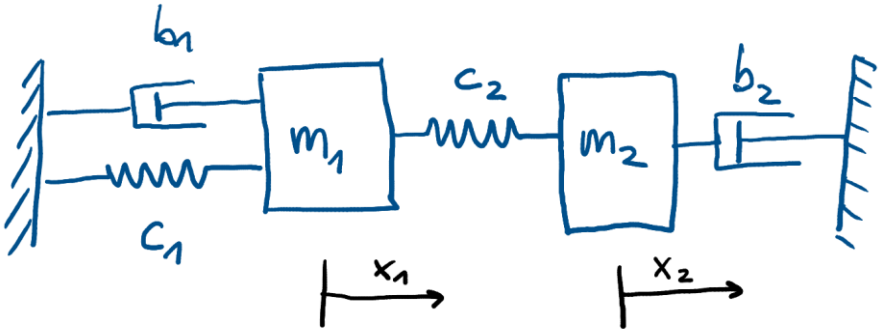

Abbildung 1

Ein Kollege hat Ihnen ein paar Rechenschritte abgenommen und Ihnen folgende Lösung gegeben:

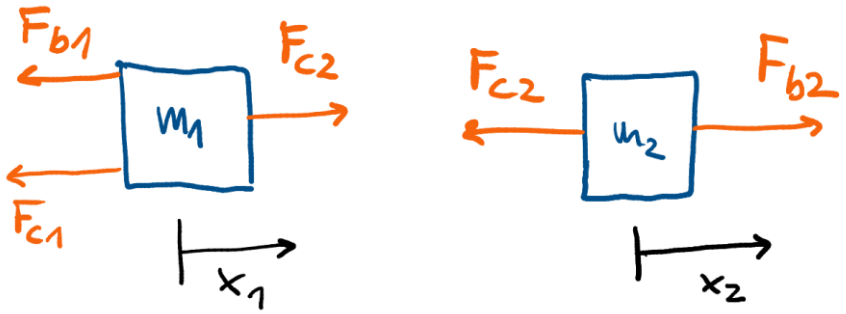

Abbildung 2

$$F_{c1} = c_1 x_1 \qquad F_{c2} = c_2(x_2 - x_1)$$
$$F_{b1} = b_1 \dot{x}_1 \qquad F_{b2} = -b_2 \dot{x}_1$$

$$\sum F_{x1} = m_1 \ddot{x}_1 = c_2 x_2 - c_2 x_1 - c_1 x_1 - b_1 \dot{x}_1$$
$$\sum F_{x2} = m_2 \ddot{x}_2 = -b_2 \dot{x}_2 - c_2 x_2 + c_2 x_1$$

1. Lösen Sie das System gekoppelter Differentialgleichungen mithilfe von scipy. Die Masse m_2 hat dabei eine Anfangsauslenkung von 5m. Alle weiteren

Anfangsbedingungen sind Null. Plotten Sie die Geschwindigkeit beider Massen in den ersten 250 Sekunden über der Zeit. Erstellen Sie eine figure mit 2 nebeneinanderliegenden subplots. Im linken plot soll die Streckung der Feder über der Zeit 2 dargestellt werden. Im rechten plot soll die Federkraft über der Zeit der Feder 2 dargestellt werden.
Nutzen Sie dazu die folgenden Parameter:
c_1 = 1 N/m; c_2 = 0.1N/m; b_1 = 0.01 Ns/m; b_2 = 0.05 Ns/m; m_1 = 3 kg; m_2 = 1 kg

2. Die Feder c_2 wird nun durch die Feder aus Aufgabe 1 ersetzt. Alle anderen Parameter dieser Aufgabe bleiben gleich. Importieren Sie die Funktionsdefinition aus Aufgabenteil 1.2 mit einem geeigneten *import* Befehl. Erstellen Sie dazu eine neue *.py Datei und fügen Sie die Funktion darin ein. Laden Sie die optimalen Werte aus der in Aufgabe 1 gespeicherten txt-Datei. Sollten Sie Aufgabe 1 nicht gelöst haben können sie die Werte $[a = 5, b = 14]$ verwenden. Erstellen Sie erneut alle in Aufgabenteil 2.1 geforderten plots.

Aufgabe 3: Objekt-Orientierte Programmierung (15 Punkte)

Als Softwareentwickler einer kleinen Firma wurden Sie beauftragt einen Digitalen Impfpass zu erstellen. Der digitale Impfpass soll Betreibern von Restaurants, Clubs und Geschäften auf einfache Weise ermöglichen, den Gesundheitsstatus des Inhabers zu bewerten. Dabei sollen geimpfte, genesene und getestete Personen gleich behandelt werden. Sie haben bereits ein UML Klassendiagramm (Abbildung 3) für Ihre Softwarelösung erstellt. In den Teilaufgaben 1-5 können Sie nun den Impfpass strukturiert entwickeln

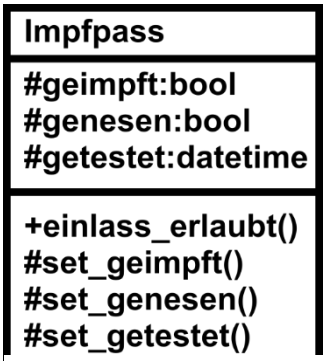

Abbildung 3

1. Definieren Sie die Klasse **Impfpass**. In der magischen Methode **__init__()** sollen die geschützten Attribute (protected attributes) **_geimpft** und **_genesen** mit dem Wert **False** initialisiert werden. Das geschützte Attribut **_getestet** soll mit dem Wert **datetime.datetime.now() - datetime.timedelta(hours=25)** initialisiert werden. Importieren Sie dafür zunächst das **datetime** Modul aus der Python Standard Library.
2. Schreiben sie die geschützten Methoden **_set_geimpft()** und **_set_genesen().** Die Setter-Methoden sollen die jeweils dazugehörigen Attribute **_geimpft** und **_genesen** auf den Wert **True** setzten.
3. Schreiben Sie jetzt die geschützte Methode **_set_getestet**. Setzen Sie in der Methode das Attribut **_getestet** auf den Wert **datetime.datetime.now()**.
4. Schreiben Sie nun die öffentliche Methode (public method) **einlass_erlaubt().** Die Methode soll als Rückgabewert **True** ausgeben, wenn der Impfpassinhaber entweder geimpft oder genesen ist oder der letzte Test nicht länger als 24 Stunden zurück liegt. Eine Zeitspanne von 24h können Sie über

datetime.timedelta(hours=24) definieren. Andernfalls soll die Methode **False** zurückgeben.

5. Instanziieren sie nun außerhalb der Klassendefinition ein Objekt der Klasse **Impfpass**. Verwenden Sie die Methode **einlass_erlaubt()** und geben Sie das Ergebnis als ganzen Satz über eine formatierten String mit dem **print** Befehl aus. Verwenden Sie jetzt die Methode _**set_getestet()** und geben sie das Ergebnis erneut als Satz aus.

Aufgabe 4: Debugging (5 Punkte)

Gegeben ist folgende fehlerhafte Funktion:

```
def crazy_function(number='7', new_list=()):
“””
Ausgabe:        bool_value: Überprüft ob number_sum gleich 41 ist
                number_sum: Summe der Listeneinträge
                Monty: String Objekt mit dem Inhalt Monty
“””

    new_list.extend([number, number])
    new_list = new_list * 3
    number_sum = np.sum(new_list)
    bool_value = number_sum != 42
    return [bool_value, number_sum, Monty]
```

Die Funktion **crazy_function()** soll für den Inputparameter **number = 7** das Tupel **(True, 42, ‘Monty‘)** zurückgeben. Leider haben sich mehrere Fehler eingeschlichen.

1. Überprüfen sie zunächst die Standardargumente (default arguments) der Funktion und berichtigen sie diese gegebenenfalls.
2. Korrigieren Sie die einzelnen Rechenschritte so, dass keine Fehler mehr erzeugt werden und das Ergebnis **(True, 42, ‘Monty‘)** als Tupel ausgegeben wird.
3. Rufen Sie die Funktion auf und geben Sie das Ergebnis in der Konsole aus.