Institut für Echtzeitsysteme
und Softwaretechnik
Prof. Dr. Derk Rembold

Hochschule
Albstadt-Sigmaringen
Albstadt-Sigmaringen University

2019

Praktikum Betriebssicherheit

Aufgabenblatt 3

Simulation einer 1oo2 Architektur mit Hilfe eines Markov-Modells

Anforderungen:

- Die Aufgabe wird in Python programmiert.
- Die Aufgabe wird von jedem Teilnehmer einzeln erstellt!
- Der Teilnehmer kommt zur Abnahme auf den Dozenten zu. Die Abnahme erfolgt für jeden Teilnehmer einzeln. Die Kenntnis des Quellcodes wird erwartet.
- Der Teilnehmer bringt ein Bild des Markov-Modells mit zur Abnahme und erklärt dem Dozenten die Herleitung. Handgeschriebenes wird nicht akzeptiert.
- Programmcode und ein Bild des Markov-Modells wird auf Ilias hochgeladen. Die Lokation wird im Praktikum bekanntgegeben. Beide Files haben folgendes Format:
  - <Name>_<Vorname>_<Matrikelnummer>_Aufgabe_3_Programmcode.py

- **Bedingung zur Abnahme ist die Fähigkeit zur Matrixmultiplikation, da dies für die Aufgabe notwendig ist! Es gibt zwei Versuche bei der Abnahme.**
- **Es gibt eine Frist für die Abnahme, Abgabe des Berichts und Hochladen der Files. Diese wird im Praktikum bekanntgegeben.**

Einleitung

Ein Regelungssystem für einen chemischen Prozess mit einfacher Eingangs- und einfacher Ausgangskomponente wird als 1oo2 Architektur entwickelt, Bild 1. Die Rechnerkomponente ist eine SPS. Diese ist redundant ausgelegt. Es existiert ein Diagnosesystem, das Ausfälle beim Eingang, Ausgang und SPS erkennt. Das Diagnosesystem führt dann den chemischen Prozess in einen sicheren Zustand, um die Wartung zu ermöglichen.

Aufgabe

Es soll die 1oo2 Architektur mit Hilfe eines Markov-Modells modelliert werden. Ziel ist es, das System zu simulieren und dann die Wahrscheinlichkeit des Systems zu ermitteln, dass es in 40 Tagen, in einem halben Jahr und in 12 Jahren noch in Betrieb ist. Das Zeitinkrement dt für die Simulation soll eine Stunde betragen.

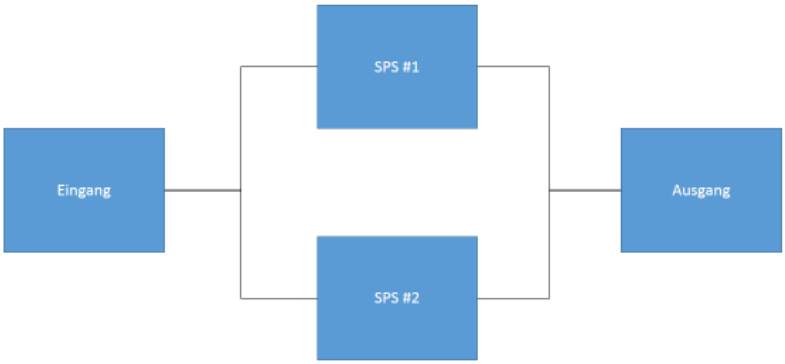

Bild 1: 1oo2 Architektur

Das Markov-Modell wird sechs Zustände erhalten. Es gibt den ersten Zustand, bei dem das System voll funktionsfähig ist. Weiter gibt es einen zweiten Zustand bei dem einer der redundanten SPS-Komponenten ausgefallen sind. Die Eingang und Ausgangskomponenten sind aber noch funktionsfähig. Dann gibt es einen dritten Zustand bei dem die Eingängen bzw. Ausgängen ausgefallen sind. Beim vierten Zustand sind beide SPS ausgefallen.
Im fünften Zustand befindet sich das System in der Wartung. Das defekte System kann innerhalb einer Diagnosezeit $t_{234\text{-}5}$ (6 Stunden) zur Wartung gebracht werden und im Wartezustand braucht es ca. 8 Stunden für die Reparatur. Im letzten Zustand ist das System nicht mehr funktionsfähig und nicht mehr zu warten (der chemische Prozess ist außer Kontrolle). Es gibt keine Möglichkeit mehr das System in den Wartungszustand zu führen.

a.) Bestimmen Sie das Markov-Modell. Zeichnen Sie die Zustände und die Transitionen. Zeitinkrement ist eine Stunde. Es gibt die folgenden Ausfallraten und Übergangsraten:

$\boldsymbol{Ausfallrate}: \boldsymbol{\lambda_{12} = 1500Fit}$
$\boldsymbol{Ausfallrate}: \boldsymbol{\lambda_{13} = 1500Fit}$
$\boldsymbol{Ausfallrate}: \boldsymbol{\lambda_{14} = 2500Fit}$
$\boldsymbol{Ausfallrate}: \boldsymbol{\lambda_{23} = 7000Fit}$
$\boldsymbol{Ausfallrate}: \boldsymbol{\lambda_{24} = 7000Fit}$
Ü$\boldsymbol{bergangsrate}: \boldsymbol{\mu_{16} = \mu_{26} = \mu_{36} = \mu_{46} = 20FIT}$
$\boldsymbol{Reparaturrate}: \boldsymbol{\mu_{r51}}$ (läß$\boldsymbol{t\ sich\ aus\ Diagnosezeit\ ermitteln}$)
$\boldsymbol{Diagnoserate}: \boldsymbol{\mu_{234-5}}$ (läß$\boldsymbol{t\ sich\ aus\ durchschnittliche\ Reparaturzeit\ ermitteln}$)

Die Indizes der Raten bezeichnen die Zustände.

b.) Programmieren Sie das Markov Modell in Python. Verwenden Sie dafür drei Klassen: STATE, TRANSITION und MARKOV, siehe Bild 2, 3 u. 4. Die Klassen können wir folgt aufgebaut sein (Klassen sind nicht vollständig):

```
class STATE:
    def __init__(self,name, num):
        self.name = name
        self.num=num
        return
```

Bild 2: Zustand

```
class TRANSITION:
    def __init__(self,source, destination, name, rate):
        self.source = source
        self.destination = destination
        self.name = name
        self.rate = rate
        return
```

Bild 3: Transition

```
class MARKOV:
    def __init__(self, name, dt=1.0):
        self.nodes=[]
        self.transitions=[]
        self.name=name
```

Bild 4: Markov

Das Markov Modell soll wie unten dargestellt aufgebaut werden. Es werden für die Klasse Markov die Methoden state() und transition() benötigt:

M = MARKOV("Beispiel")

S1 = STATE('S1',0)
S2 = STATE('S2',1)
M.state(S1)
M.state(S2)
T12 = TRANSITION(S1, S2, 'l12', 1000)
M.transition(T12)

c.) Programmieren Sie eine Simulation, um die Wahrscheinlichkeiten zu ermitteln, dass das System in 40 Tagen, in einem halben Jahr und in 12 Jahren noch in Betrieb ist. Es soll dafür die Methode probability() der Klasse Markov programmiert werden. Sie können hierfür die library numpy verwenden. Setzen Sie für die Simulation die folgende Formel ein:

$$\underline{\boldsymbol{p}}(\boldsymbol{t}+\boldsymbol{dt}) = \underline{\boldsymbol{p}}(\boldsymbol{t}) \cdot \boldsymbol{P}$$

mit

$$\underline{\boldsymbol{p}}(\boldsymbol{0}) = (\boldsymbol{1} \quad \boldsymbol{0} \quad \boldsymbol{0} \quad \boldsymbol{0} \quad \boldsymbol{0} \quad \boldsymbol{0})$$

d.) Programmieren Sie den Graphen des Markov Modells mit Graphviz/Matplot und drucken Sie diesen aus.

e.) Führen Sie die Simulation bei der Abnahme mit einem vom Dozenten vorgegebenen Modell aus. Drucken Sie die Übergangsmatrix P und den Graphen des Markov-Modells mit dem Programm aus.