MIE 1613 - Assignment 1 - Problem 2

- Armando Ordorica
- 1005592164

- Jan 2023
- Prof. Vahid Sarhangian

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

We're interested in finding the first system failure or its long-run system availability.

- The state of the TTF system at any point in time is the number of functional components, 2,1 or 0.
- The terminal state is when the state is zero.

```
def generate_random_samples_of_repairs(num_days_min=1, num_days_max = 6, N=10):

  samples_list = []
  for i in range(0,N):
    random_sample = int(np.ceil(np.random.random()*num_days_max))
    samples_list.append(random_sample)

  # sns.distplot(samples_list)

  # for x in list(set(samples_list)):
    # print(x, samples_list.count(x))

  return samples_list



# for x in list(set(samples_list)):
#   print(x, samples_list.count(x))


class Machine:
  def __init__(self, clock=0, num_functional_components = 2, repair_time = 2.5, broken_times_sample = [5,3,6,1,3]):
    self.clock = clock
    self.state = num_functional_components
    self.repair_time = repair_time
    self.broken_times_sample = broken_times_sample

    self.clock_adjusted_failures = np.cumsum(self.broken_times_sample)

    self.num_functional_components = num_functional_components
    self.system_alive = self.get_system_status()

    self.next_failure_time = self.get_next_failure_time()
    self.next_repair_time = self.get_next_repair_time()

    self.log = self.initialize_log()


  def get_clock_value(self):

    # print("Get next clock value...")

    # print(f"Increasing clock from:{self.clock} to...")
    self.clock = np.floor(self.clock)
    self.clock = self.clock+1
    self.clock = min(self.clock, self.next_repair_time)

    # print(f"clock:{self.clock}")
    # print("\n")
    return self.clock
```

```
def get_system_status(self):
  if self.state>0 and self.state <=2:
    return True
  else:
    return False


def get_next_failure_time(self):
  next_failures_list = [x for x in self.clock_adjusted_failures if x>self.clock]

  if len(next_failures_list)>0:
    return np.min(next_failures_list)
  else:
    return np.inf

def get_next_repair_time(self):
  # print("Next Repair")
  if self.state == 2: #nothing to repare
    # print("Nothing to repair")
    return np.inf

  elif self.state >0 and self.state<2 and self.clock == self.next_failure_time:
    # print(f"self.clock:{self.clock} + self.repair_time:{self.repair_time}")
    return self.clock + self.repair_time

  else:
    # print("Repair time is the same")
    return self.next_repair_time

def initialize_log(self):
  if self.state>0 and self.state <=2:
    self.system_alive = True

    next_failure_time = self.get_next_failure_time()
    next_repair_time = self.get_next_repair_time()


  return pd.DataFrame({'Clock':[self.clock], 'State':[self.state], \
                       'Next Failure':[next_failure_time], 'Next Repair':[next_repair_time], \
                       'system_alive':[self.system_alive]})

def update_state(self):
  if self.clock == self.next_failure_time:
    # print("Update State ...")
    # print("here")
    # print("self.state = np.minimum(0,self.state -1)")
    # print(f"self.state:{self.state} = np.minimum(0,self.state:{self.state} -1)")
    self.state = np.maximum(0,self.state -1)

  if self.clock == self.next_repair_time:
    self.state = np.minimum(2, self.state+1)


def increase_clock(self):
  # print(f"Increasing Clock from clock: {self.clock} to {self.clock+1}")
  self.clock = self.get_clock_value()

  self.update_state()

  self.next_repair_time = self.get_next_repair_time()
  self.next_failure_time = self.get_next_failure_time()
  self.system_alive =self.get_system_status()



def update_log(self):

  self.log = self.log.append(pd.DataFrame({'Clock':[self.clock], 'State':[self.state], \
                       'Next Failure':[self.next_failure_time], 'Next Repair':[self.next_repair_time], \
                       'system_alive':[self.system_alive]}))

  # self.log.drop_duplicates(subset= ['State', 'Next Failure', 'Next Repair', 'system_alive'], keep='first', inplace=True)
  return self.log
```

```
def print_log():
  print(f"samples_list:{samples_list}")
```

Problem 2. (25 Pts.)

In the original TTF example we simulated the system until the time of first failure.

- Modify the simulation model to simulate the system for a given fixed number of days denoted by T. Assume that all other inputs and assumptions are the same as in the original example.

Comentary on the answer As you can see in the code below, the following code prints out a log for the first T=30 using the `broken_times_sample` given in the example in class for easier illustration.

```
ttfs = []

machine1 = Machine(broken_times_sample=[5,3,6,1,5,3,6,1])
T = 30
while machine1.clock <T:
  machine1.increase_clock()
  machine1.update_log()

machine1.log
```

	Clock	State	Next Failure	Next Repair	system_alive
0	0.0	2	5.0	inf	True
0	1.0	2	5.0	inf	True
0	2.0	2	5.0	inf	True
0	3.0	2	5.0	inf	True
0	4.0	2	5.0	inf	True
0	5.0	1	8.0	7.5	True
0	6.0	1	8.0	7.5	True
0	7.0	1	8.0	7.5	True

Problem 2 part a)

a) We say that the system is fully functional provided that both components (active and spare) are functional. Denote by $A(t)$ a process that takes value 1 if the system is fully functional at time t and 0 otherwise. Then,

$$\bar{A}(T) = \frac{1}{T}\int_0^T A(t)dt$$

is the fraction of the the system is fully functional between 0 and T. Modify your simulation model to estimate $\bar{A}(T)$ until $T = 1000$ on one replication of the simulation.

b) Estimate $\bar{A}(T)$ for $T = 2000$ and $T = 4000$ again using a single replication and compare the values with the estimate from part (a).

Answer

As we can see on the graph below, the average fraction of functional time of the system converges as the number of random samples or simulated time horizon Tincreases. However, note that these are across-replication outputs, which are independent because we roll the die anew on each replication, and identically distributed because we apply the same initial contidions and model logic to those rolls of the die.

```
Ts_input = [10, 50, 100, 200, 500, 1000, 2000, 3000, 4000]
A_Ts = []


for T in Ts_input:

  samples_list = generate_random_samples_of_repairs(num_days_min=1, num_days_max = 6, N=T)
  machine1 = Machine(broken_times_sample=samples_list)

  while machine1.clock <T:
    machine1.increase_clock()
    machine1.update_log()


  machine1.log['Fully_functional'] = np.where(machine1.log['State']==2, 1,0)
  machine1.log['fractional_time'] = np.where(machine1.log['Clock']%1 !=0, 1,0)

  temp_df = machine1.log[machine1.log['fractional_time'] ==0].copy(deep=True)
  numerator = temp_df.groupby(['Fully_functional']).count().loc[1].iloc[0]
  denominator = len(temp_df)

  A_t = numerator/denominator
  A_Ts.append(A_t)

  print(f"A_t:{A_t}")


machine1.log
```

```
A_t:0.45454545454545453
A_t:0.4117647058823529
A_t:0.40594059405940597
A_t:0.34328358208955223
A_t:0.3592814371257485
A_t:0.37362637362637363
A_t:0.37031484257871067
A_t:0.37187604131956015
A_t:0.3791552111972007
```

	Clock	State	Next Failure	Next Repair	system_alive	Fully_functic
0	0.0	2	1	inf	True	
0	1.0	1	5	3.5	True	
0	2.0	1	5	3.5	True	
0	3.0	1	5	3.5	True	

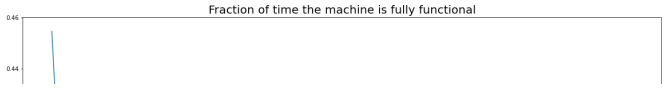
```
summary_df = pd.DataFrame({'T':Ts_input, 'A(t)':A_Ts})
summary_df
```

	T	A(t)
0	10	0.454545
1	50	0.411765
2	100	0.405941
3	200	0.343284
4	500	0.359281
5	1000	0.373626
6	2000	0.370315

```
plt.figure(figsize=(20,10))
plt.plot(summary_df['T'], summary_df['A(t)'])
plt.title("Fraction of time the machine is fully functional", fontsize=20)

plt.xlabel("Number of replications (T)")
plt.ylabel("Fraction of time fully functional (A(t))")
```

```
Text(0, 0.5, 'Fraction of time fully functional (A(t))')
```



Appendix

```
ttfs = []

machine1 = Machine(broken_times_sample=[5,3,6,1])
T = 30
while machine1.clock <T and machine1.state>0:
  machine1.increase_clock()
  machine1.update_log()

machine1.log
```

	Clock	State	Next Failure	Next Repair	system_alive
0	0.0	2	5.0	inf	True
0	1.0	2	5.0	inf	True
0	2.0	2	5.0	inf	True
0	3.0	2	5.0	inf	True
0	4.0	2	5.0	inf	True
0	5.0	1	8.0	7.5	True
0	6.0	1	8.0	7.5	True
0	7.0	1	8.0	7.5	True
0	7.5	2	8.0	inf	True
0	8.0	1	14.0	10.5	True
0	9.0	1	14.0	10.5	True
0	10.0	1	14.0	10.5	True
0	10.5	2	14.0	inf	True
0	11.0	2	14.0	inf	True
0	12.0	2	14.0	inf	True

```
machine1.log['Fully_functional'] = np.where(machine1.log['State']==2, 1,0)
machine1.log['fractional_time'] = np.where(machine1.log['Clock']%1 !=0, 1,0)
machine1.log
```

	Clock	State	Next Failure	Next Repair	system_alive	Fully_functio
0	0.0	2	5.0	inf	True	
0	1.0	2	5.0	inf	True	
0	2.0	2	5.0	inf	True	
0	3.0	2	5.0	inf	True	
0	4.0	2	5.0	inf	True	
0	5.0	1	8.0	7.5	True	

```
temp_df = machine1.log[machine1.log['fractional_time'] ==0].copy(deep=True)
numerator = temp_df.groupby(['Fully_functional']).count().loc[1].iloc[0]
denominator = len(temp_df)

A_t = numerator/denominator
A_t
```

```
0.5
```

0	10.0	1	14.0	10.5	True	
0	11.0	2	14.0	inf	True	