POLITECNICO
MILANO 1863

lasar^3

Laboratory of analysis of systems for the assessment of reliability, risk and resilience

Monte Carlo Simulations: Exercise Session

Luca Pinciroli

Politecnico di Milano, Energy Department

luca.pinciroli@polimi.it

November 18th 2024

EXERCISE 1

Exercise 1

Consider the Weibull distribution:

$$F_T(t) = 1 - e^{-\beta t^{\alpha}}, \quad f_T(t) = \alpha\beta t^{\alpha-1} e^{-\beta t^{\alpha}}$$

with $\alpha = 1.5, \beta = 1$

1. Sample *N*=400 values from $f_T(t)$
2. Verify whether the obtained distribution provides a good approximation of the Weibull distribution. To this aim, you are required to:
   A. find the empirical probability density function (pdf) of the sampled value in 1
   B. compare the empirical pdf found in 2A. with the analytical Weibull distribution.

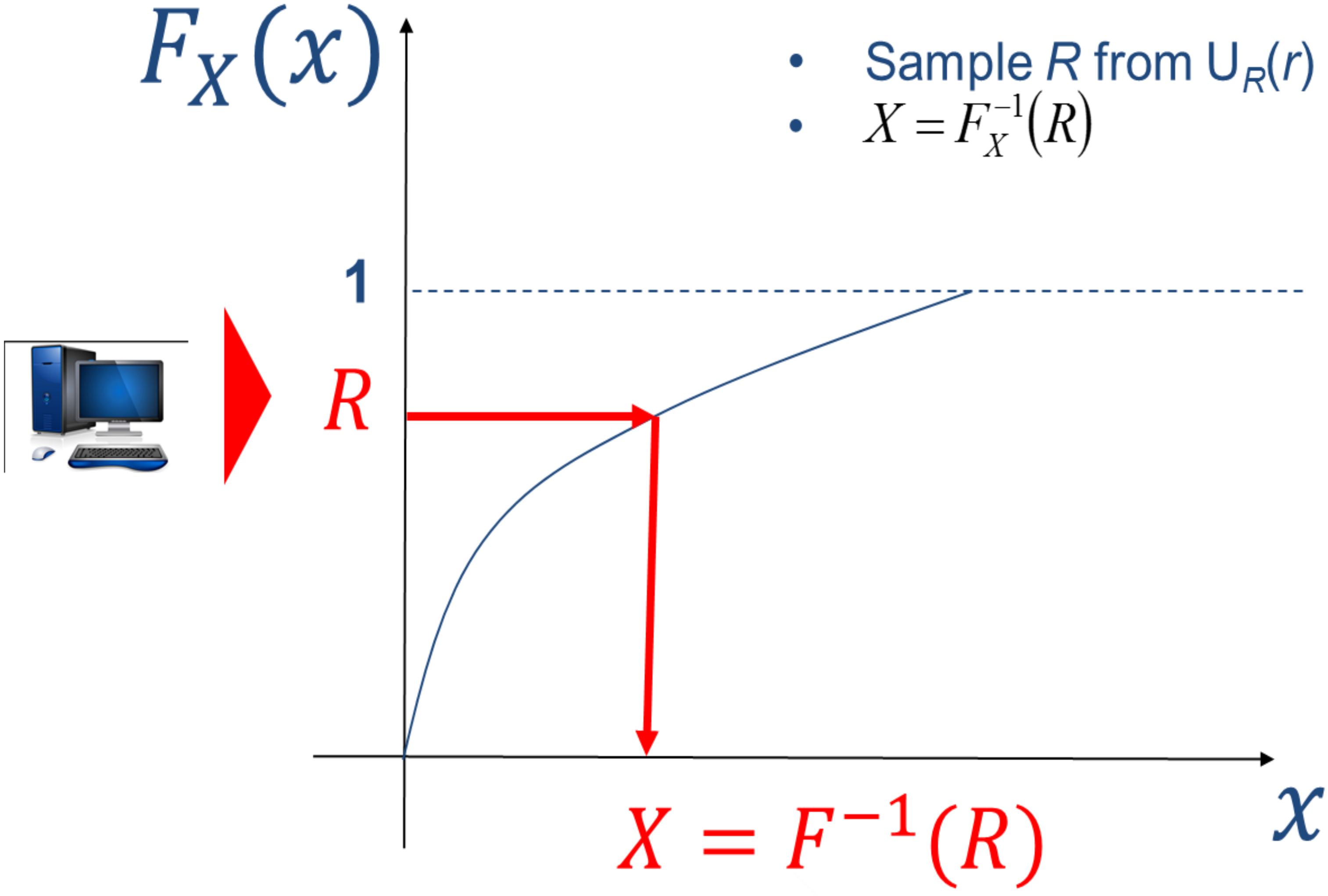
Useful Commands

- np.random.rand(N): provides N random numbers sampled from a uniform distribution in the range [0,1)

- num_samples = matplotlib.pyplot.hist(Y, bins) bins the elements of Y into the bins defined by bins and returns the number of elements in each counter.

Sampling Random Numbers from FX(x)

- Sample R from $U_R(r)$
- $X = F_X^{-1}(R)$



Example: Weibull Distribution

- Time-dependent hazard rate $\lambda(t) = \beta\alpha t^{\alpha-1}$

cdf: $F_T(t) = P\{T \le t\} = 1 - e^{-\beta t^{\alpha}}$

pdf: $f_T(t) \cdot dt = P\{t \le T < t + dt\} = \alpha\beta t^{\alpha-1} e^{-\beta t^{\alpha}} \cdot dt$

- Sampling a failure time *T* (by the inverse transform)

$$R \equiv F_R(r) = F_T(t) = 1 - e^{-\beta t^{\alpha}}$$

$$T = F_T^{-1}(R) = \left(-\frac{1}{\beta}\ln(1-R)\right)^{\frac{1}{\alpha}}$$

Exercise 1 – Initialization

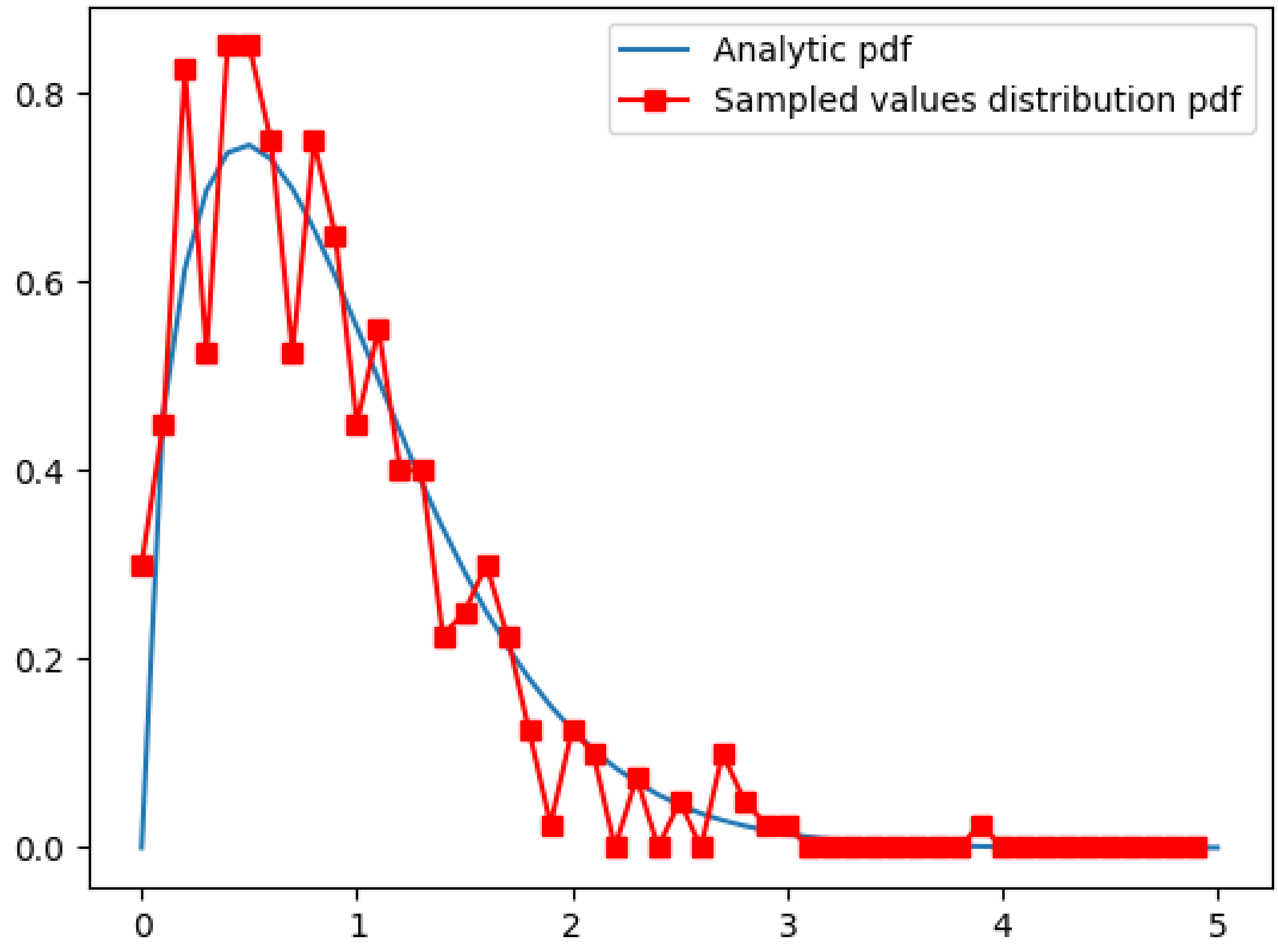

```
import numpy as np
import matplotlib.pyplot as plt
beta = 1
alpha = 1.5
# Sample N values from the Weibull distribution
N = 400 #number of samples
r = np.random.rand(N)
t = (-np.log(1-r)/beta)**(1/alpha) # inverse transform method
```

```
# Verify the distribution
# Estimated pdf
delta_channel = 0.1 # histogram bins width
channels = np.arange(0, 5.1, delta_channel) # histogram bins
num_samples = plt.hist(t,channels);
pdf_est = num_samples[0]/(N*delta_channel);

# Compute the analytic pdf
analytic_weibull = alpha*beta*channels**(alpha-1)*np.exp(-beta*channels**alpha)
```

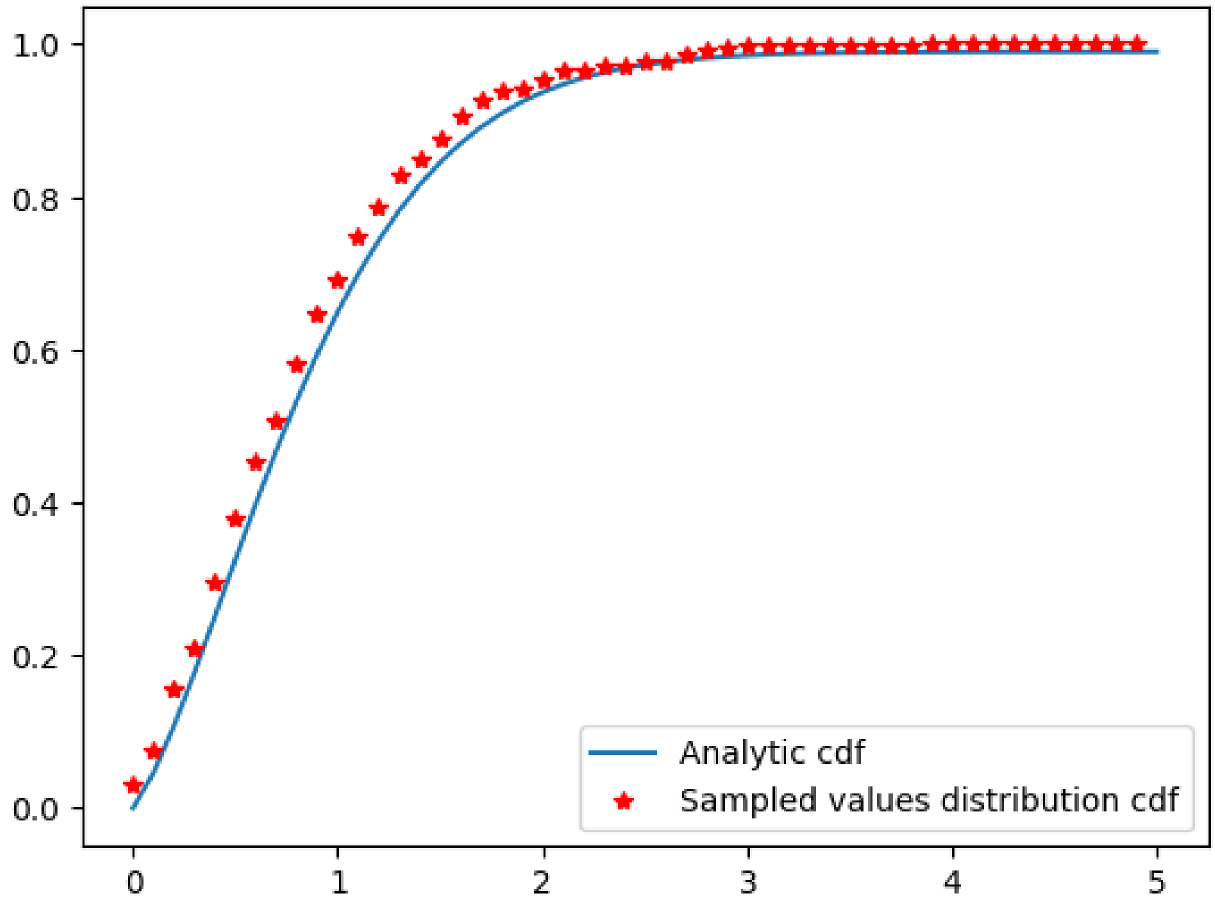
Exercise 1 – Results


```
# plot the pdf
plt.figure() # pdf
plt.plot(channels,analytic_weibull, label='Analytic pdf’)
plt.plot(channels[:-1],pdf_est,'-sr', label='Sampled
 values distribution pdf’)
plt.legend()

# plot the cdf
plt.figure()
cdf_est = np.cumsum(pdf_est)*delta_channel;
plt.plot(channels, np.cumsum(analytic_weibull)*delta_channel, label='Analytic cdf’)
plt.plot(channels[:-1], cdf_est,'*r', label='Sampled values distribution cdf’)
plt.legend()
```

Probability density function



Cumulative distribution

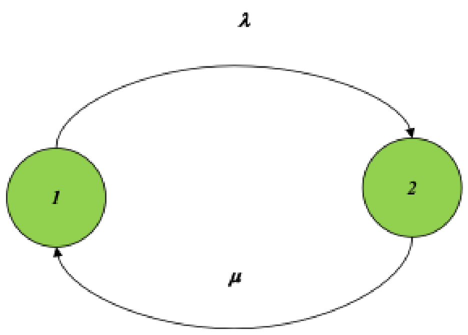


EXERCISE 2

Exercise 2

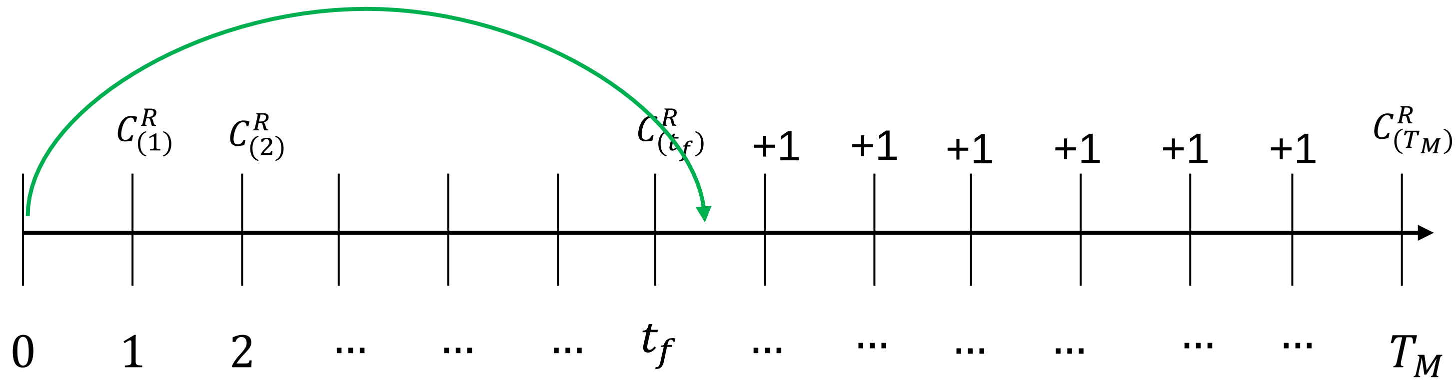
Write the MC code for the estimation of the **time dependent reliability** and **instantaneous availability** of a continuously monitored component with constant failure (λ) and repair (μ) rates

	values
λ	$3\cdot 10^{-3}\ h^{-1}$
μ	$25\cdot 10^{-3}\ h^{-1}$

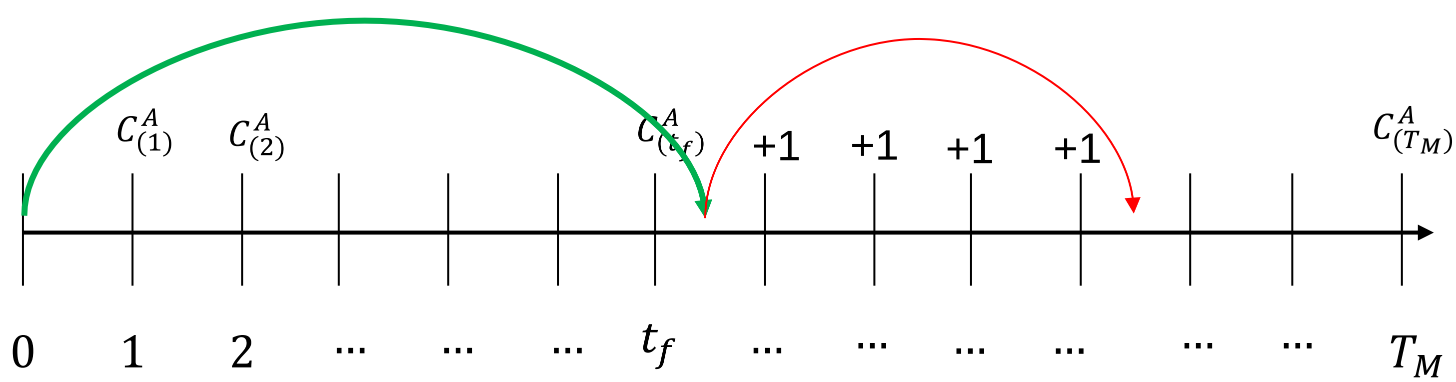


- You can assume a mission time of 10^3 time units
- You can compute the time dependent reliability and the instantaneous availability at all times: 0,1,2,3,…10^3

Estimation of the System Reliability



Estimation of the System Availability



Exercise 2 – Initialization


```
import numpy as np
import matplotlib.pyplot as plt
# Initialize parameters
Tm = 1000  # mission time
M = 10000  # number of trials
l = 3e-3  # failure rate
mu = 25e-3  # repair rate
Dt = 1  # bin length
Time_axis = np.arange(0, Tm + 1, Dt)
unrel_counter = np.zeros(M)  # unreliability at mission time
counter_q = np.zeros(len(Time_axis))  # instantaneous unavailability
unrel = np.zeros([M,len(Time_axis)]) #initialize time dependent reliability
```

Exercise 2 – Monte Carlo

```
# Start simulation of M trials
for i in range(M):
    t = 0
    state = 0  # 0 = working, 1 = failed
    unrel_flag = 0
     while t < Tm:
        if state == 0:  # Working -> sample failure time
            failure_samples = -np.log(1 - np.random.rand(1)) / l
            t += failure_samples
            if t < Tm and unrel_flag == 0:
                unrel_flag = 1
                unrel_counter[i] = 1
                unrel_time = int(np.ceil(t)[0])
                unrel_time = min(unrel_time, Tm)  # Ensure it does not exceed mission time
                unrel[i, unrel_time:] = 1

      state = 1
      failure_time = t
      lower_b = int(np.searchsorted(Time_axis, failure_time)[0])
```

Exercise 2 – MC & Results

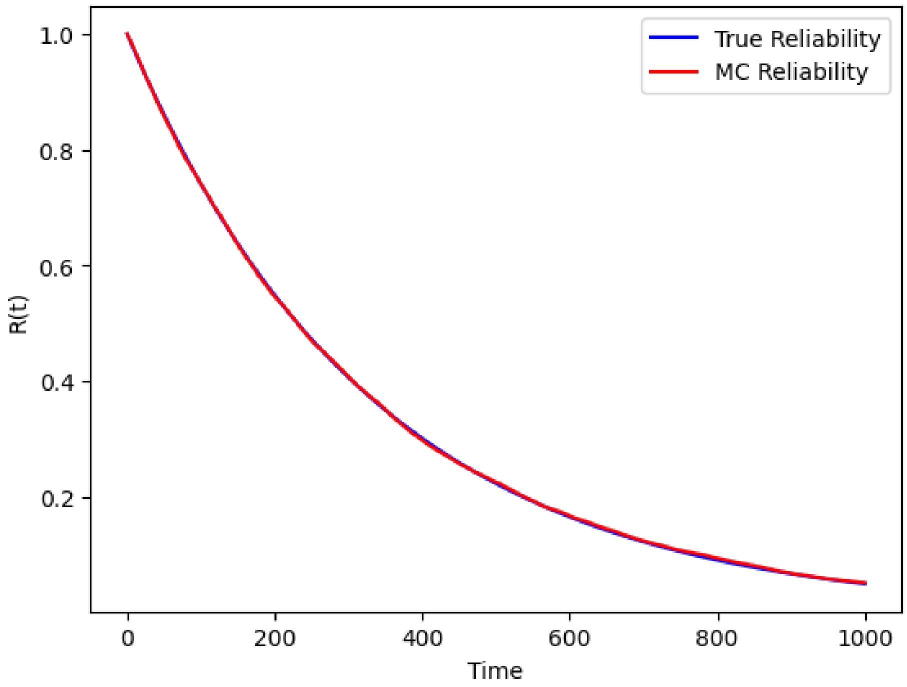
```
    else:  # Failed -> sample repair time
        repair_samples = -np.log(1 - np.random.rand(1)) / mu
        t += repair_samples
        repair_time = t
        if t < Tm:
            upper_b = int(np.searchsorted(Time_axis, repair_time)[0])
        else:
            upper_b = len(Time_axis)  # Mission time exceeded
        counter_q[lower_b:upper_b] += 1
        state = 0  # Repair finished
 # Calculate results
Rel_Tm = 1 - np.mean(unrel_counter)  # Reliability at mission time
Rel_MC = 1 - np.mean(unrel, axis=0)  # Time-dependent reliability
Rel_true = np.exp(-l * Time_axis)  # Analytic reliability
Av_MC = 1 - (counter_q / M)  # Instantaneous availability
Av_true = 1 - (l / (l + mu) - (l / (l + mu)) * np.exp(-(l + mu) * Time_axis))  # Analytic availability
```

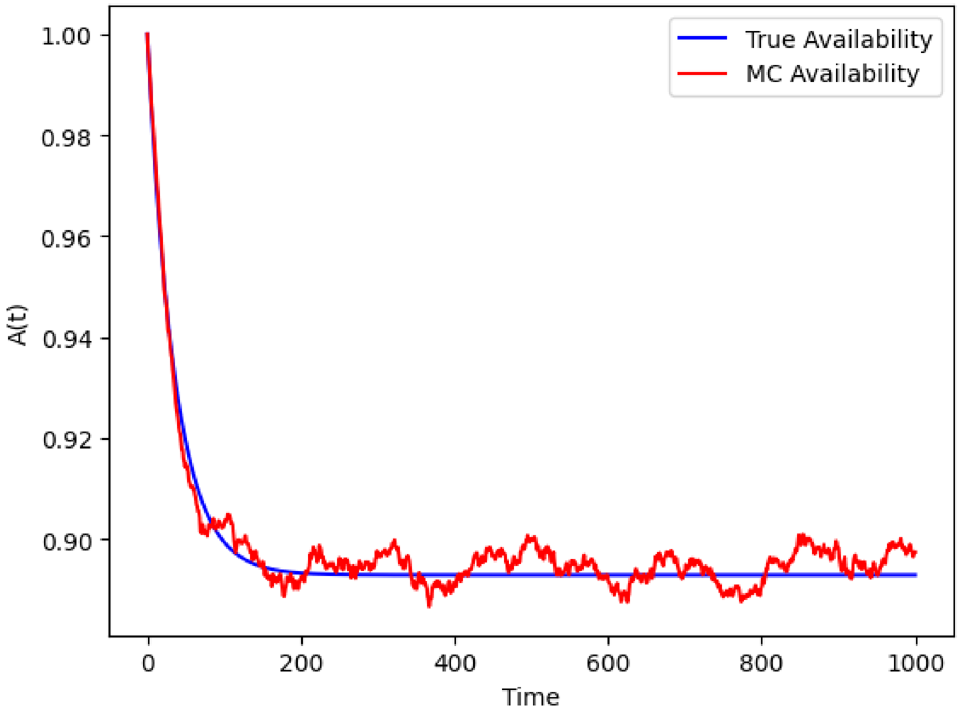
Exercise 2 – Results

```
# Plot results
plt.figure()
plt.plot(Time_axis, Rel_true, 'blue', label='True Reliability’)
plt.plot(Time_axis, Rel_MC, 'red', label='MC Reliability’)
plt.xlabel('Time’)
plt.ylabel('R(t)’)
plt.legend()
plt.figure()
plt.plot(Time_axis, Av_true, 'blue', label='True Availability’)
plt.plot(Time_axis, Av_MC, 'red', label='MC Availability’)
plt.xlabel('Time’)
plt.ylabel('A(t)’)
plt.legend()
plt.show()
```

Exercise 2 – Results

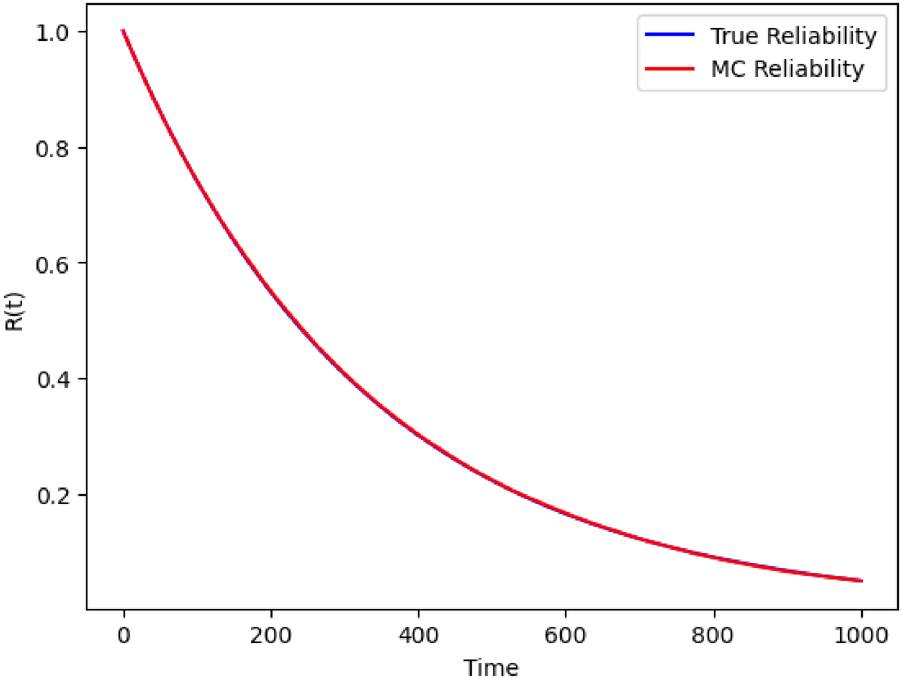
M = 10000

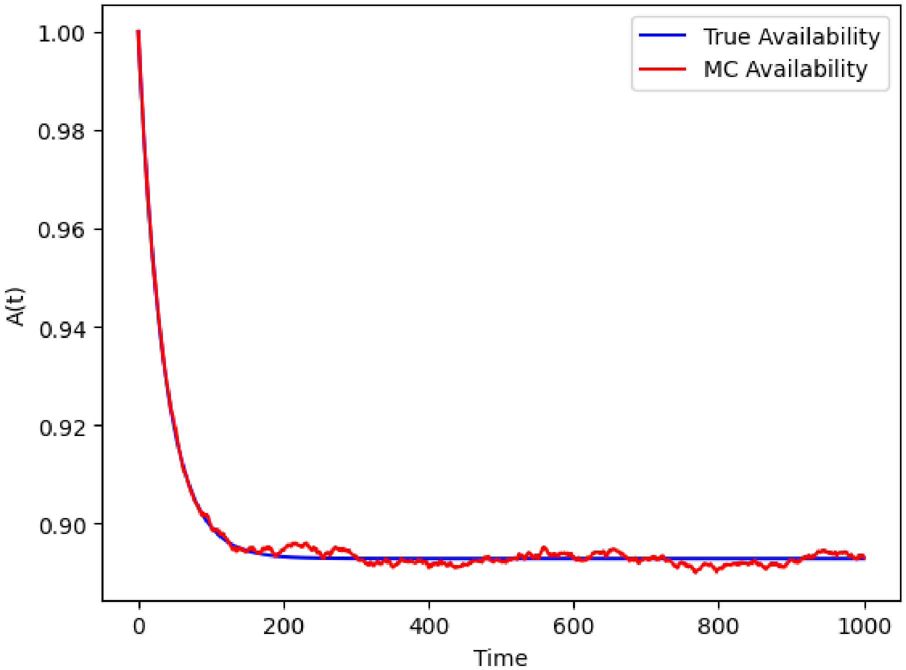




Exercise 2 – Results

M = 100000



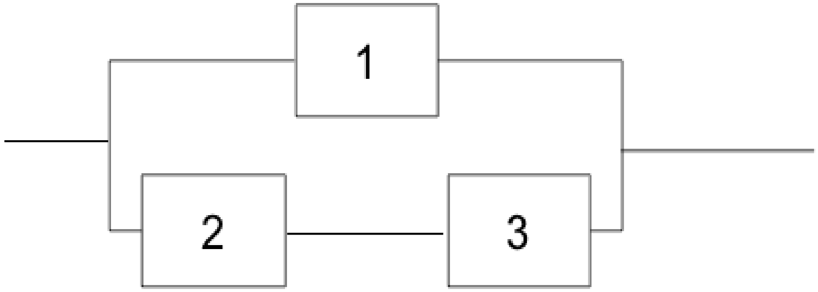


Exercise 3

Exercise 3

Consider the system in figure composed of three components(A, B, C). Each component can be in two different health states (1-nominal, 2-failed) with exponentially distributed transition times between them. Assuming a mission time $T = 500\ hours$, write the MC code for the estimation of:

- The time dependent reliability
- The instantaneous availability.
- The estimators uncertainty



	1	2	3
λ	$1\cdot 10^{-3}\ h^{-1}$	$2\cdot 10^{-2}\ h^{-1}$	$5\cdot 10^{-2}\ h^{-1}$
μ	$3\cdot 10^{-2}\ h^{-1}$	$5\cdot 10^{-2}\ h^{-1}$	$5\cdot 10^{-3}\ h^{-1}$

Exercise 3 – Initialization


```python
import numpy as np
import matplotlib.pyplot as plt

# System parameters
# Transition rates
lambda_1, mu_1 = 0.001, 0.03
lambda_2, mu_2 = 0.02, 0.05
lambda_3, mu_3 = 0.05, 0.005

# Transition matrices for each component
Trans_A = np.array([[0, lambda_1], [mu_1, 0]])
Trans_B = np.array([[0, lambda_2], [mu_2, 0]])
Trans_C = np.array([[0, lambda_3], [mu_3, 0]])
Trans_Mat = np.array([Trans_A, Trans_B, Trans_C])

# Failure states and initial state
failed_states = np.array([[1, 0, 1], [1, 1, 0], [1, 1, 1]])
initial_state = np.array([0, 0, 0])

# Mission time and time axis
Tm = 500
Dt = 1 # timestep
Time_axis = np.arange(0, Tm + 1, Dt)
N = 10000 # Number of Monte Carlo samples

# Predefine arrays for results
unrel_counter = np.zeros(N)
unrel = np.zeros((N, len(Time_axis)))
counter_q = np.zeros((N, len(Time_axis)))
```

Exercise 3 – Monte Carlo

```
# Monte Carlo Simulation
for n in range(N):
    t = 0
    unrel_flag = 0
    system_state = 1
    current_state = np.copy(initial_state)
    failure_flag = 0

    while t < Tm:
        # Calculate component transition rates
        lambda_out = np.array([
            Trans_A[current_state[0], :].sum(),
            Trans_B[current_state[1], :].sum(),
            Trans_C[current_state[2], :].sum()
        ])
        lambda_sys = lambda_out.sum()

        # Sample transition time and update total time
        t_trans = -np.log(np.random.rand()) / lambda_sys
        t += t_trans

        # Break if mission time is exceeded
        if t >= Tm:
          if system_state == 0:
              #increase all unavailability counter between lower_b and Tm
              counter_q[n, int(lower_b):] += 1
          break
```

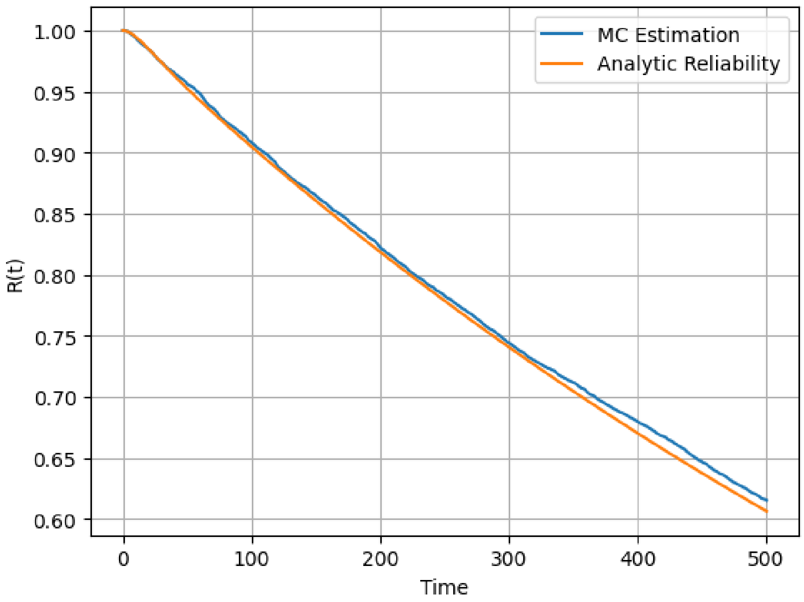
Exercise 3 – Monte Carlo

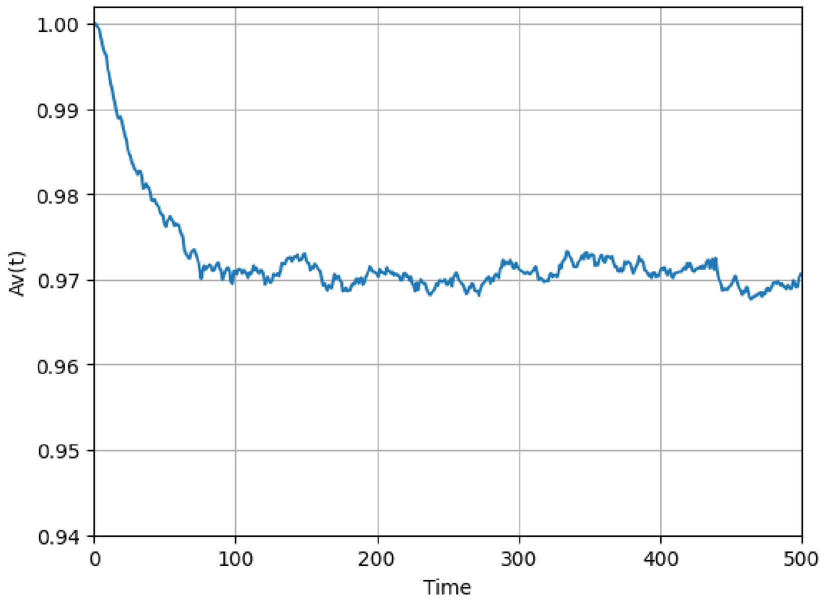

```
else:
        # Component and transition type sampling
        comp = np.searchsorted(np.cumsum(lambda_out) / lambda_sys, np.random.rand())
        current_state[comp] = 1 - current_state[comp]  # Update the component state

        # Check for failure configuration
        if any(np.array_equal(current_state, failed) for failed in failed_states):
            failure_flag = 1

        # If system fails within mission time
        if failure_flag:
            if unrel_flag == 0:
                unrel_flag = 1
                unrel[n, int(np.ceil(t)):] = 1  # Update unrel for reliability calculation

            if system_state == 1:
                failure_time = t
                lower_b = np.searchsorted(Time_axis, failure_time)
                system_state = 0

        # If system repairs within mission time
        elif system_state == 0:
            repair_time = t
            system_state = 1
            upper_b = np.searchsorted(Time_axis, repair_time)
            counter_q[n, lower_b:upper_b] += 1

        failure_flag = 0

    unrel_counter[n] = unrel_flag
```

Exercise 3 – Results

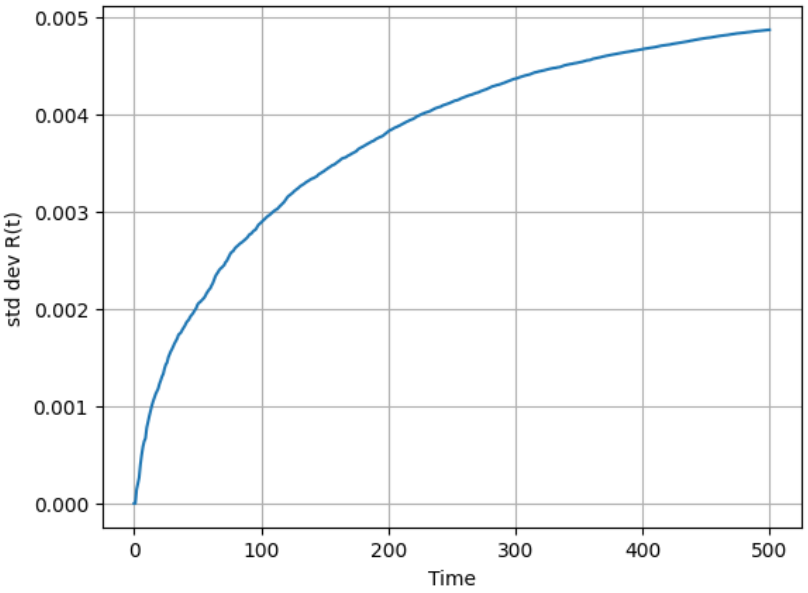

```
# Calculate reliability and availability from Monte Carlo samples
rel_MC = 1 - np.mean(unrel_counter)
Mean_rel = 1 - np.mean(unrel, axis=0)
sig_rel = np.sqrt((Mean_rel-Mean_rel**2)/N) #s_rel =np.sqrt(np.var(unrel,axis=0)/N)
rel_an = np.exp(-lambda_1 * Time_axis) + np.exp(-(lambda_2 + lambda_3) * Time_axis) - np.exp(-(lambda_1
+ lambda_2 + lambda_3) * Time_axis)
Av_MC = 1 - np.mean(counter_q, axis=0)
sig_av = np.sqrt((Av_MC-Av_MC**2)/N) #s_av =np.sqrt(np.var(counter_q,axis=0)/N)

# Plot results
plt.figure()
plt.plot(Time_axis, Mean_rel, label='MC Estimation')
plt.plot(Time_axis, rel_an, label='Analytic Reliability')
plt.xlabel('Time')
plt.ylabel('R(t)')
plt.legend()
plt.grid()

plt.figure()
plt.plot(Time_axis, Av_MC)
plt.xlabel('Time')
plt.ylabel('Av(t)')
plt.grid()
plt.axis([0, Tm, 0.94, 1.002])

plt.figure()
plt.plot(Time_axis, sig_rel)
plt.xlabel('Time')
plt.ylabel('std dev R(t)')
plt.grid()

plt.figure()
plt.plot(Time_axis, sig_av)
plt.xlabel('Time')
plt.ylabel('std dev Av(t)')
plt.grid()

plt.show()
```

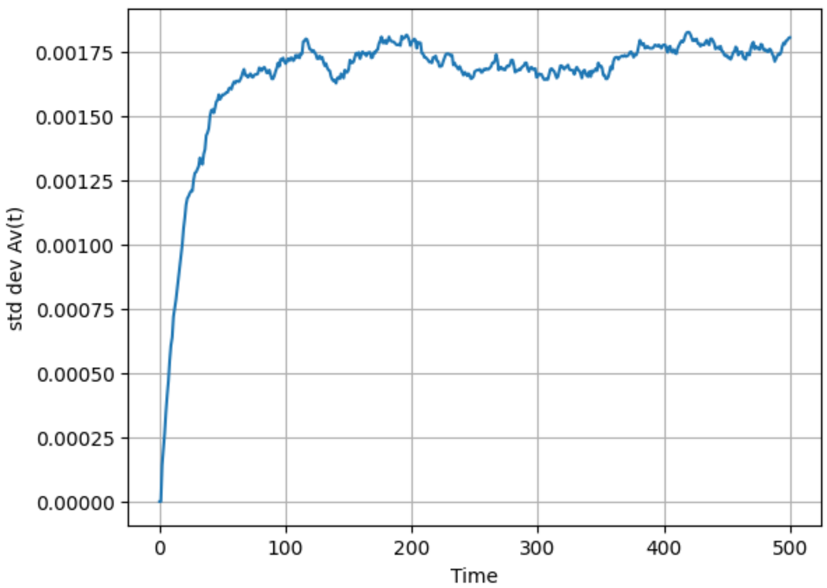
Exercise 3 – Results





Exercise 3 – Results

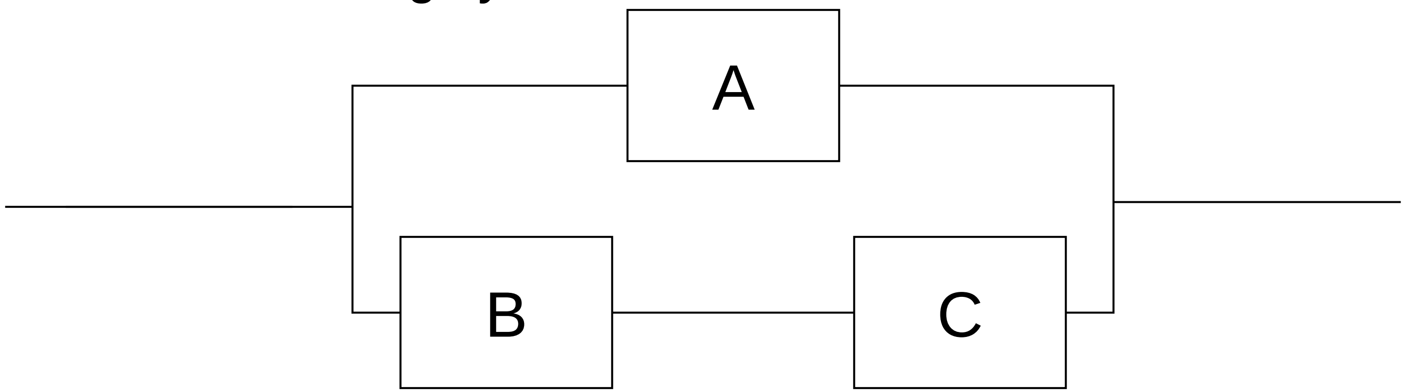




HOMEWORK

Exercise 4

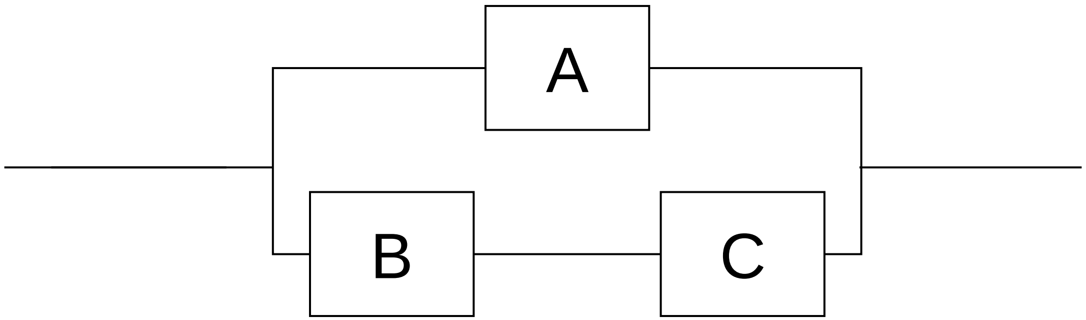
- Consider the following system



- Components can be in three states and the time of transition from one state to another is exponentially distributed:

Initial \ Arrival	1	2	3
1(nominal)	0	$\lambda_{1\rightarrow 2}^{A(B,C)}$	$\lambda_{1\rightarrow 3}^{A(B,C)}$
2 (degraded)	0	0	$\lambda_{2\rightarrow 3}^{A(B,C)}$
3 (failed)	$\lambda_{3\rightarrow 1}^{A(B,C)}$	$\lambda_{3\rightarrow 2}^{A(B,C)}$	0

Exercise 4



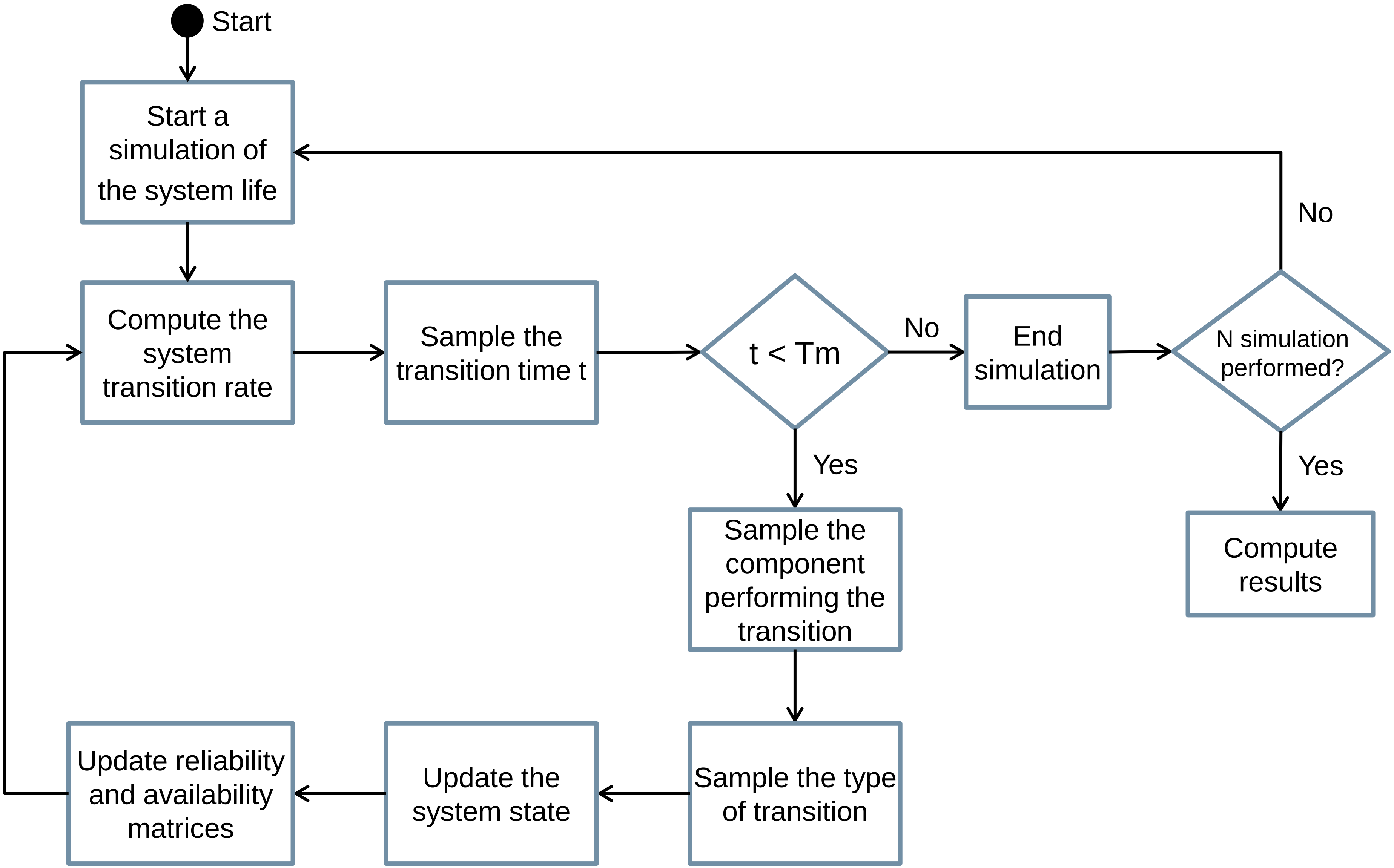
A	1	2	3
1	-	$3\ 10^{-3}$	10^{-3}
2	-	-	$6\ 10^{-3}$
3	$8\ 10^{-3}$	$5\ 10^{-3}$	-

B	1	2	3
1	-	$1\ 10^{-3}$	$5\ 10^{-3}$
2	-	-	$4\ 10^{-3}$
3	$7.5\ 10^{-3}$	$3.5\ 10^{-3}$	-

C	1	2	3
1	-	$8\ 10^{-3}$	$2.5\ 10^{-3}$
2	-	-	$2\ 10^{-3}$
3	$4\ 10^{-3}$	$1.5\ 10^{-3}$	-

- Estimate the **reliability** of the system at T_{miss} = 4000
- Estimate the **time dependent reliability** R(t)
- Estimate the **instataneous availability** A(t)

Flow diagram



Sampling the time of transition

- The rate of transition of the system out of its current configuration
- (1, 1, 1) is:

$$\lambda^{(1,1,1)} = \lambda_{1\to 2}^{A} + \lambda_{1\to 3}^{A} + \lambda_{1\to 2}^{B} + \lambda_{1\to 3}^{B} + \lambda_{1\to 2}^{C} + \lambda_{1\to 3}^{C}$$

- We are now in the position of sampling the first system transition time t_1, by applying the **inverse transform method**:

$$t_1 = t_0 - \frac{1}{\lambda^{(1,1,1)}} \ln(1 - R_t)$$

where $R_t \sim U[0,1)$

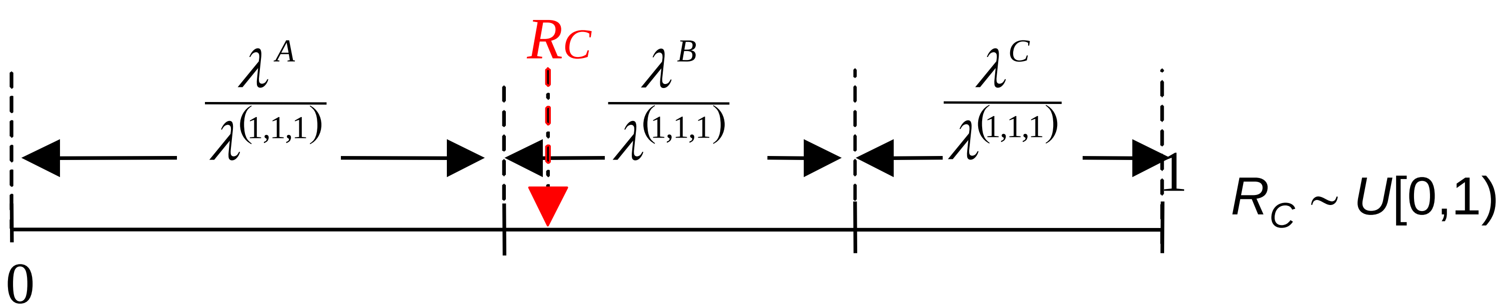
Sampling the kind of Transition

- Assuming that $t_1 < T_M$ (otherwise we would proceed to the successive trial), we now need to determine which component has undergone the transition
- The probabilities of components A, B, C undergoing a transition out of their initial nominal states 1, given that a transition occurs at time t_1, are:

$$\frac{\lambda^A}{\lambda^{(1,1,1)}}, \quad \frac{\lambda^B}{\lambda^{(1,1,1)}}, \quad \frac{\lambda^C}{\lambda^{(1,1,1)}}$$

$$\lambda^A = \lambda^A_{1\to2} + \lambda^A_{1\to3} \qquad \lambda^B = \lambda^B_{1\to2} + \lambda^B_{1\to3} \qquad \lambda^C = \lambda^C_{1\to2} + \lambda^C_{1\to3}$$

- Thus, we can apply the inverse transform method to the discrete distribution

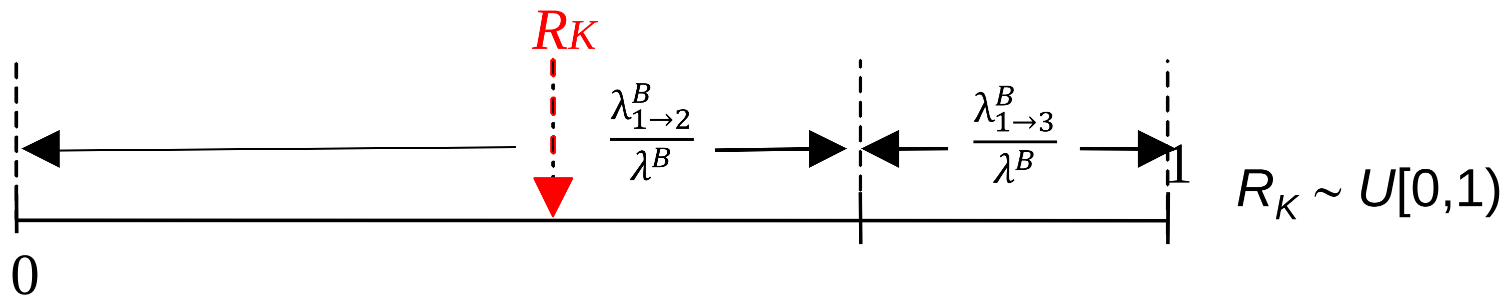


Sampling the kind of Transition

- Since component B is the one undergoing the transition we need to sample the new state of component B.
- The probabilities of components B undergoing a transition out of their initial nominal states 1 given that a transition occurs at time t_1, are:

$$\frac{\lambda^B_{1\to 2}}{\lambda^B} \qquad \frac{\lambda^B_{1\to 3}}{\lambda^B}$$

- Thus, we can apply the inverse transform method to the discrete distribution



Next step

- As a result of this first transition, at t_1 the system is operating in configuration (1,2,1).
- The simulation now proceeds to sampling the next transition time t_2 with the updated transition rate

$$\lambda^{(1,2,1)} = \lambda^{A}_{1\to2} + \lambda^{A}_{1\to3} + \lambda^{B}_{2\to3} + \lambda^{C}_{1\to2} + \lambda^{C}_{1\to3}$$

Exercise 4 – Initialization


```
import numpy as np
import matplotlib.pyplot as plt

# System parameters
# Transition rates
lambda_A_0_1, lambda_A_0_2 = 3e-3, 1e-3
lambda_A_1_2, lambda_A_2_0, lambda_A_2_1 = 6e-3, 8e-3, 5e-3
lambda_B_0_1, lambda_B_0_2, lambda_B_1_2 = 1e-3, 5e-3, 4e-3
lambda_B_2_0, lambda_B_2_1 = 7.5e-3, 3.5e-3
lambda_C_0_1, lambda_C_0_2, lambda_C_1_2 = 8e-3, 2.5e-3, 2e-3
lambda_C_2_0, lambda_C_2_1 = 4e-3, 1.5e-3

# Transition matrices
Trans_A = np.array([[0, lambda_A_0_1, lambda_A_0_2], [0, 0, lambda_A_1_2], [lambda_A_2_0, lambda_A_2_1, 0]])
Trans_B = np.array([[0, lambda_B_0_1, lambda_B_0_2], [0, 0, lambda_B_1_2], [lambda_B_2_0, lambda_B_2_1, 0]])
Trans_C = np.array([[0, lambda_C_0_1, lambda_C_0_2], [0, 0, lambda_C_1_2], [lambda_C_2_0, lambda_C_2_1, 0]])
Trans_Mat = [Trans_A, Trans_B, Trans_C]
cum_trans = [np.cumsum(mat, axis=1) for mat in Trans_Mat]

# Failed states and initial state
failed_states = np.array([[2, 0, 2], [2, 2, 0], [2, 1, 2], [2, 2, 1], [2, 2, 2]])
initial_state = np.array([0, 0, 0])
lambda_out = np.zeros(3)
Tm = 4000  # mission time
Dt = 1      # time bin width
Time_axis = np.arange(0, Tm + Dt, Dt)
N = 10000   # number of samples

# Initialize counters and results
unrel_counter = np.zeros(N)
unrel = np.zeros((N, len(Time_axis)))
counter_q = np.zeros(len(Time_axis))
```

```
for n in range(N):  # Main Monte Carlo cycle
    t = 0
    unrel_flag = 0  # System reliability flag
    current_state = initial_state.copy()
    system_failed = False

    while t < Tm:
        # Calculate the system transition rate
        for i in range(3):
            lambda_out[i] = np.sum(Trans_Mat[i][current_state[i], :])
        lambda_sys = np.sum(lambda_out)

        # Sample transition time
        t += -np.log(np.random.rand()) / lambda_sys
        if t >= Tm:
            if system_failed:
                # System failure extends until mission end
                counter_q[lower_b:] += 1
            break

        # Determine component transition
        comp_choice = np.searchsorted(np.cumsum(lambda_out) / lambda_sys, np.random.rand())
        state_choice = np.searchsorted(cum_trans[comp_choice][current_state[comp_choice], :] / lambda_out[comp_choice], np.random.rand())

        # Update component state
        current_state[comp_choice] = state_choice
```

Exercise 4 – Monte Carlo


```
    # Check if in a failure configuration
    if np.any(np.all(current_state == failed_states, axis=1)):
        if not system_failed:
            # Mark start of failure (downtime)
            system_failed = True
            lower_b = np.searchsorted(Time_axis, t)
            if not unrel_flag:
                unrel_flag = 1
                unrel[n, int(np.ceil(t)):] = 1  # Reliability calculation

    else:
        # System restored
        if system_failed:
            upper_b = np.searchsorted(Time_axis, t)  # repair end index
            counter_q[lower_b:upper_b] += 1
            system_failed = False

unrel_counter[n] = unrel_flag
```

Exercise 4 – Results


```
# Final reliability and availability calculations
rel_MC = 1 - np.mean(unrel_counter)
Mean_rel = 1 - np.mean(unrel, axis=0)  # Instantaneous reliability
Av_MC = 1 - (counter_q / N)            # Instantaneous availability

# Plotting results
plt.figure()
plt.plot(Time_axis, Mean_rel, label="Reliability R(t)")
plt.xlabel('Time')
plt.ylabel('R(t)')
plt.grid()

plt.figure()
plt.plot(Time_axis, Av_MC, label="Availability Av(t)")
plt.xlabel('Time')
plt.ylabel('Av(t)')
plt.grid()
plt.ylim(0.87, 1)

plt.show()
```

Exercise 4 – Results