STK3405 – Lecture 15

A. B. Huseby

Department of Mathematics
University of Oslo, Norway



Section 7.1.3

The Birnbaum measure of reliability importance for non-repairable systems



The time-dependent Birnbaum measure

The time-dependent version of the Birnbaum importance measure is defined as follows:

$$I_B^{(i)}(t) = h(1_i, \boldsymbol{p}(t)) - h(0_i, \boldsymbol{p}(t)), \quad t \geq 0, \quad i \in C.$$

We then choose a sufficiently dense set of t-values, $0 = t_0 < t_1 < \cdots < t_K$, and calculate $I_B^{(i)}(t)$, for these t-values.

The importance curve for $t \in [0, t_K]$ is then obtained by drawing a curve through the points:

$$(t_0, I_B^{(i)}(t_0)), (t_1, I_B^{(i)}(t_1)), \ldots, (t_K, I_B^{(i)}(t_K)).$$

If K is sufficiently large, a nice smooth curve is obtained.



Example – 2-out-of-3 system

EXAMPLE: Let (C, ϕ) be a 2-out-of-3 system with $C = \{1, 2, 3\}$ and structure function:

$$\phi(X_1, X_2, X_3) = X_1X_2 + X_1X_3 + X_2X_3 - 2X_1X_2X_3$$

From this it follows that the reliability of the system at time $t \geq 0$ is:

$$h(\boldsymbol{p}(t)) = p_1(t)p_2(t) + p_1(t)p_3(t) + p_2(t)p_3(t) - 2p_1(t)p_2(t)p_3(t)$$

Hence, we get:

$$I_B^{(1)}(t) = h(1_1, \boldsymbol{p}(t)) - h(0_1, \boldsymbol{p}(t)) = p_2(t) + p_3(t) - 2p_2(t)p_3(t)$$
$$I_B^{(2)}(t) = h(1_2, \boldsymbol{p}(t)) - h(0_2, \boldsymbol{p}(t)) = p_1(t) + p_3(t) - 2p_1(t)p_3(t)$$
$$I_B^{(3)}(t) = h(1_3, \boldsymbol{p}(t)) - h(0_3, \boldsymbol{p}(t)) = p_1(t) + p_2(t) - 2p_1(t)p_2(t)$$



Example – 2-out-of-3 system (cont.)

By inserting $p_i(t) = \bar{F}_i(t)$, $i = 1, 2, 3$ into $I_B^{(1)}(t)$, $I_B^{(2)}(t)$, $I_B^{(3)}(t)$, we can calculate the reliability importance of the three components for any $t \geq 0$.

We assume more specifically that the lifetime of component i, T_i is Weibull-distributed with parameters α_i and β_i, for all $i \in C$, where:

$$\alpha_1 = 2.0, \quad \alpha_2 = 2.5, \quad \alpha_3 = 3.0,$$

$$\beta_1 = 50.0, \quad \beta_2 = 60.0, \quad \beta_3 = 70.0.$$



Example – 2-out-of-3 system (cont.)

```
from math import exp
import matplotlib.pyplot as plt
import numpy as np

save_plots = True

# Time parameters
max_t = 120
steps = 120

# Weibull-parameters for the components
alpha = [2.0, 2.5, 3.0]
beta = [50.0, 60.0, 70.0]

# Number of components in the system
num_comps = len(alpha)

# The reliability function of the system
def reliability(pp):
    return pp[0] * pp[1] + pp[0] * pp[2] + pp[1] * pp[2] - 2 * pp[0] * pp[1] * pp[2]

# The survival function of a Weibull-distribution
def weibull(aa, bb, t):
    if t >= 0:
        return exp(- (t / bb) ** aa)
    else:
        return 1.0
```



Example – 2-out-of-3 system (cont.)

```
# Component reliability vector
p = np.zeros(num_comps)

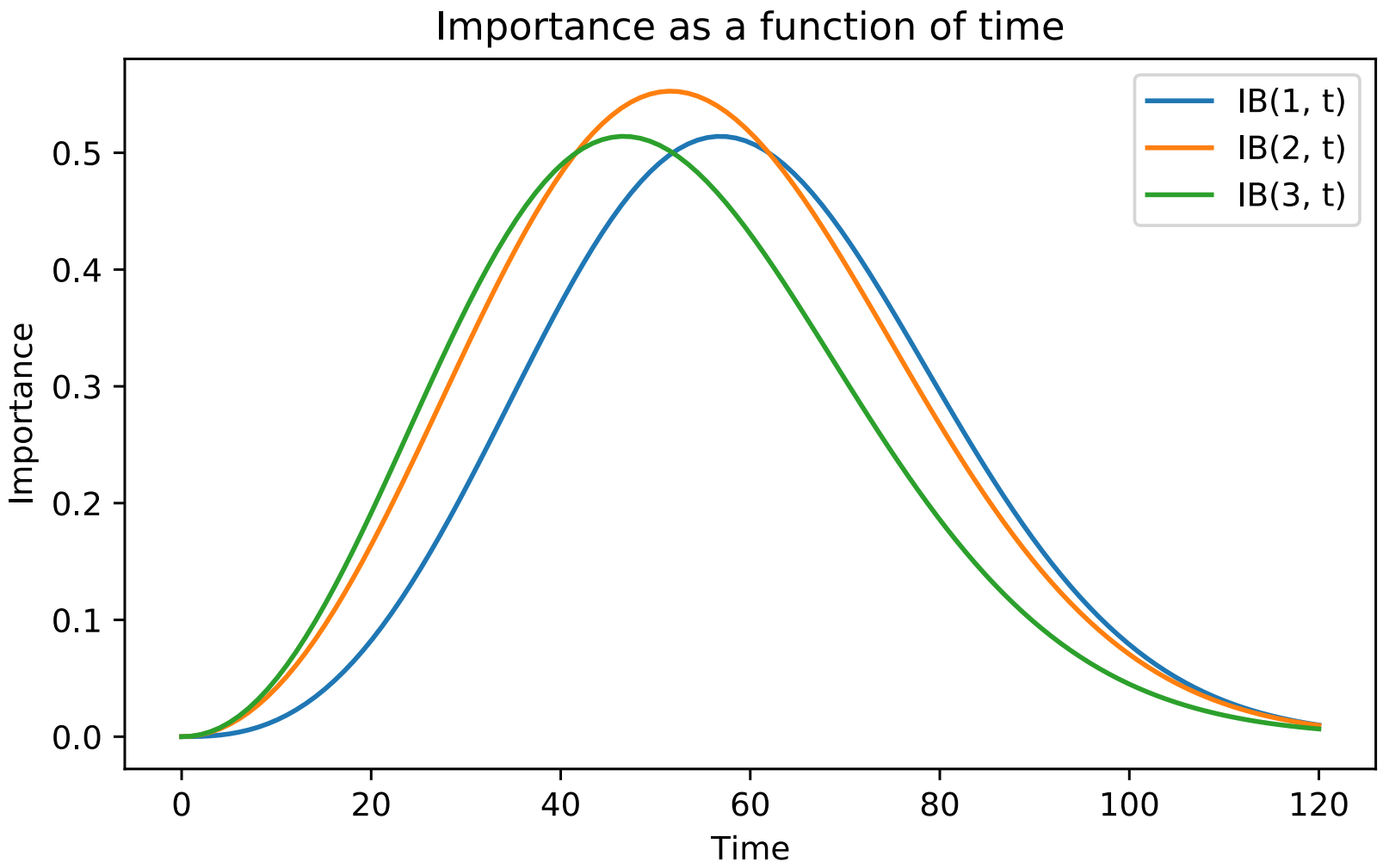
# Time values and Birnbaum measures
time = np.linspace(0.0, max_t, steps)
IB = np.zeros((num_comps, steps))

# Calculate the Birnbaum measures
for step in range(steps):
    for i in range(num_comps):
        p[i] = weibull(alpha[i], beta[i], time[step])
    for i in range(num_comps):
        temp = p[i]
        p[i] = 1
        h1 = reliability(p)
        p[i] = 0
        h0 = reliability(p)
        p[i] = temp
        IB[i][step] = h1 - h0

# Plot the results
fig = plt.figure(figsize = (7, 4))
for i in range(num_comps):
    plt.plot(time, IB[i], label='IB(' + str(i+1) + ", t)")
plt.xlabel('Time')
plt.ylabel('Importance')
plt.title("Importance as a function of time")
plt.legend()
if save_plots:
    plt.savefig("birnbaum_01/birnbaum_01.pdf")
plt.show()
```



Example – 2-out-of-3 system (cont.)





Estimating Birnbaum measure using simulation

The time-dependent Birnbaum measure can also be estimated using Monte Carlo simulation.

Our goal is to estimate the importance curves for $t \in [0, t_K]$ for a sufficiently dense set of t-values $0 = t_0 < t_1 < \cdots < t_K$.

The estimated importance values for the ith component are denoted:

$$\hat{I}_B^{(i)}(t_0), \hat{I}_B^{(i)}(t_1), \ldots, \hat{I}_B^{(i)}(t_K), \quad i \in C$$

The resulting estimated importance curve for $t \in [0, t_K]$ for the ith component is obtained by drawing a curve through the points:

$$(t_0, \hat{I}_B^{(i)}(t_0)), (t_1, \hat{I}_B^{(i)}(t_1)), \ldots, (t_K, \hat{I}_B^{(i)}(t_K)).$$



Estimating Birnbaum measure using simulation (c.)

We start by generating N samples of component lifetimes $\{T_{1k}, \ldots, T_{nk}\}$, $k = 1, \ldots, N$, where T_{ik} denotes the lifetime of the ith component in the kth simulation.

Furthermore, we assume that we are able to calculate the system lifetime as a function of the component lifetimes, either by using the formula based on minimal path sets:

$$S = \min_{1 \leq j \leq k} \max_{i \in K_j} T_i = \max_{1 \leq j \leq p} \min_{i \in P_j} T_i$$

or by using the formula based on sorted component lifetimes:

$$S = T_{i_{j^*}}$$

where:

$$j^* = \min\{j : \phi(\boldsymbol{X}_j) = 0\}$$



Estimating Birnbaum measure using simulation (c.)

We now consider the component lifetimes generated in the kth simulation, i.e., $\{T_{1k}, \ldots, T_{nk}\}$, and focus on component i.

For component i we calculate two system lifetimes:

$$L_{ik} = \text{The system lifetime given that } T_{ik} \text{ is replaced by } 0$$
$$U_{ik} = \text{The system lifetime given that } T_{ik} \text{ is replaced by } \infty$$

Thus, it follows that:

- L_{ik} is the lifetime of the system given that component i is failed all the time
- U_{ik} is the lifetime of the system given that component i is functioning all the time



Estimating Birnbaum measure using simulation (c.)

We then consider a point of time $t \in [0, t_K]$.

- If $t < L_{ik}$, we know that the system is functioning regardless of the state of component i
- If $t \geq U_{ik}$, we know that the system is failed regardless of the state of component i
- If $L_{ik} \leq t < U_{ik}$, the system is functioning if component i is functioning, and failed if component i is failed.

Thus, we conclude that in the kth simulation component i is critical for the system if and only if $L_{ik} \leq t < U_{ik}$.



Estimating Birnbaum measure using simulation (c.)

In each simulation we keep track of the points of time when component i is critical, and introduce:

$$D_{ij} = \text{The number of times component } i \text{ is critical at time } t_j.$$

for $i \in C$ and $0 = t_0 < t_1 < \cdots < t_K$.

We then obtain the following unbiased estimates for the importance of component i at the points of time $t_0, t_1, \ldots, t_K$:

$$\hat{I}_B^{(i)}(t_j) = \frac{D_{ij}}{N}, \quad i \in C.$$



Estimating Birnbaum measure using simulation (c.)

EXAMPLE: Let (C, ϕ) be a bridge system with $C = \{1, 2, 3, 4, 5\}$ and minimal path sets:

$$P_1 = \{1, 4\}, \quad P_2 = \{1, 3, 5\}, \quad P_3 = \{2, 3, 4\}, \quad P_4 = \{2, 5\}.$$

The lifetime of component i, T_i is Weibull-distributed with parameters α_i and β_i, for all $i \in C$ where:

$$\alpha_1 = 2.0, \quad \alpha_2 = 2.5, \quad \alpha_3 = 3.0, \quad \alpha_4 = 1.0, \quad \alpha_5 = 1.5,$$
$$\beta_1 = 50.0, \quad \beta_2 = 60.0, \quad \beta_3 = 70.0, \quad \beta_4 = 40.0, \quad \beta_5 = 50.0.$$

The Birnbaum measures $I_B^{(1)}(t), \ldots, I_B^{(5)}(t)$ for $t \geq 0$, are estimated by running a Monte Carlo simulation with $N = 100000$ simulations.



Estimating Birnbaum measure using simulation (c.)

```
from math import *
from random import *
import matplotlib.pyplot as plt
import numpy as np

save_plots = True

# Time parameters
max_t = 120
steps = 120

# Number of simulations
num_sims = 100000

# Weibull-parameters for the components
alpha = [2.0, 2.5, 3.0, 1.0, 1.5]
beta = [50.0, 60.0, 70.0, 40.0, 50.0]

# Number of components and minimal paths
num_comps = 5
num_paths = 4

# Birnbaum measures
IB = np.zeros((num_comps, steps))

# A component is identified as critical when t is in [L, U)
L = np.zeros(num_comps)
U = np.zeros(num_comps)

# Minimal path sets
paths = [{1,4}, {1,3,5}, {2,3,4}, {2,5}]

# Calculate the system lifetime given the component lifetimes
def sys_lifetime(tt):
    sys_life = 0.0
    for j in range(num_paths):
        path_life = np.inf
        for i in paths[j]:
            if tt[i-1] < path_life:
                path_life = tt[i-1]
        if path_life > sys_life:
            sys_life = path_life
    return sys_life
```



Estimating Birnbaum measure using simulation (c.)

```
# The simulated lifetimes of the components
t = np.zeros(num_comps)

# Time steps
time = np.linspace(0.0, max_t, steps)

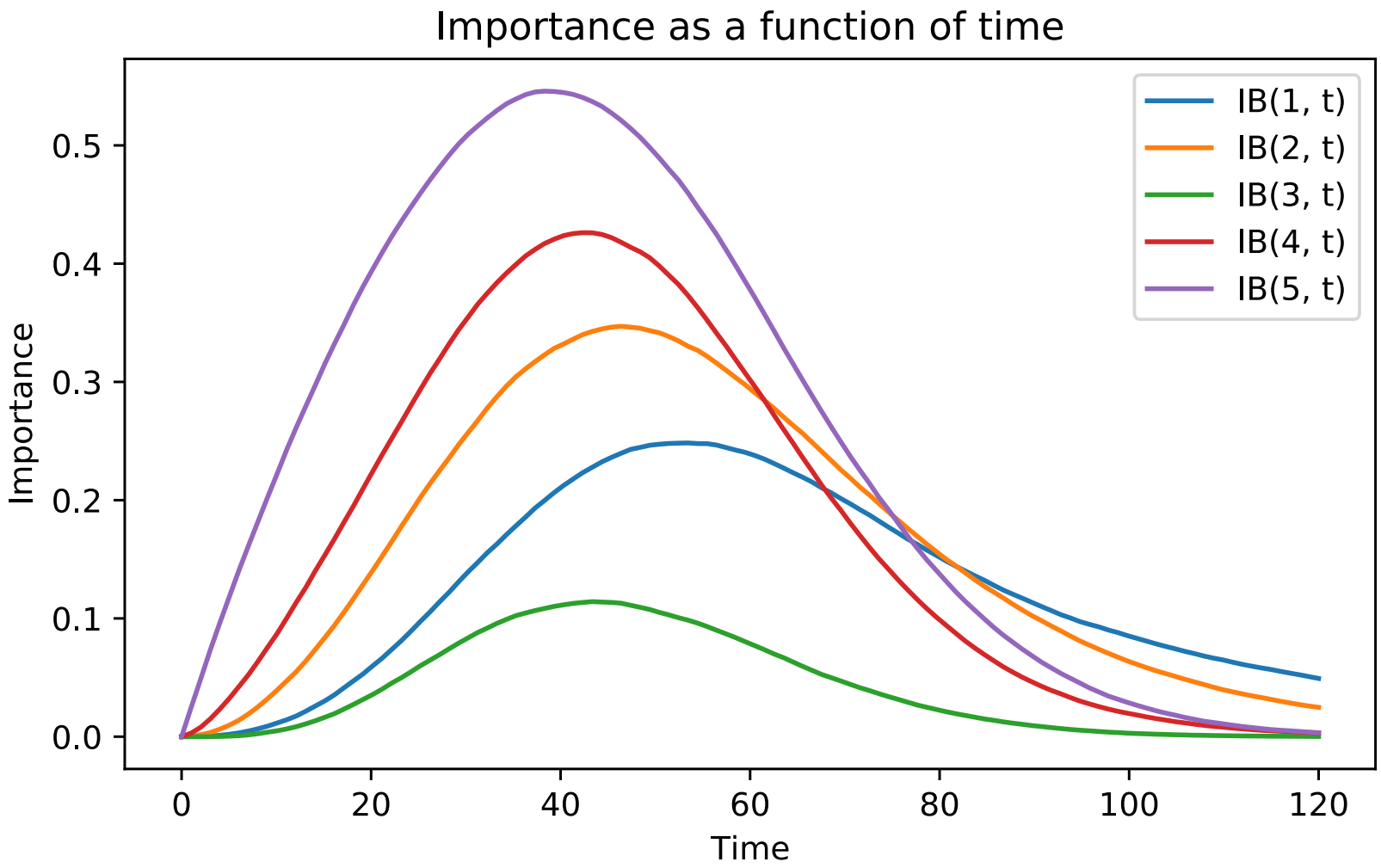
# Run simulations
for k in range(num_sims):
    for i in range(num_comps):
        t[i] = weibullvariate(beta[i], alpha[i])
    for i in range(num_comps):
        temp = t[i]
        t[i] = np.inf
        U[i] = sys_lifetime(t)
        t[i] = 0
        L[i] = sys_lifetime(t)
        t[i] = temp
    for step in range(steps):
        for i in range(num_comps):
            if L[i] <= time[step] and time[step] < U[i]:
                IB[i][step] += 1

# Calculate average importance values
for step in range(steps):
    for i in range(num_comps):
        IB[i][step] = IB[i][step] / num_sims

fig = plt.figure(figsize = (7, 4))
for i in range(num_comps):
    plt.plot(time, IB[i], label='IB(' + str(i+1) + ", t)")
plt.xlabel('Time')
plt.ylabel('Importance')
plt.title("Importance as a function of time")
plt.legend()
if save_plots:
    plt.savefig("birnbaum_02/birnbaum_02.pdf")
plt.show()
```



Estimating Birnbaum measure using simulation (c.)





MC simulation using sorted component lifetimes (c.)

EXAMPLE: Let (C, ϕ) be a threshold system with component set $C = \{1, 2, 3, 4, 5\}$ with structure function:

$$\phi(\boldsymbol{X}) = \mathrm{I}(\sum_{i=1}^{5} \omega_i X_i \geq 5.0)$$

where the component weights $\omega_1, \ldots, \omega_5$ are:

$$\omega_1 = 2.0, \quad \omega_2 = 4.0, \quad \omega_3 = 2.3, \quad \omega_4 = 3.5, \quad \omega_5 = 1.9$$

The lifetime of component i, T_i is Weibull-distributed with parameters α_i and β_i, for all $i \in C$ where:

$$\alpha_1 = 2.0, \quad \alpha_2 = 2.5, \quad \alpha_3 = 3.0, \quad \alpha_4 = 1.0, \quad \alpha_5 = 1.5,$$
$$\beta_1 = 50.0, \quad \beta_2 = 60.0, \quad \beta_3 = 70.0, \quad \beta_4 = 40.0, \quad \beta_5 = 50.0.$$

The Birnbaum measures $I_B^{(1)}(t), \ldots, I_B^{(5)}(t)$ for $t \geq 0$, are estimated by running a Monte Carlo simulation with $N = 100000$ simulations.



Estimating Birnbaum measure using simulation (c.)

```
from math import *
from random import *
import matplotlib.pyplot as plt
import numpy as np

save_plots = True

# Time parameters
max_t = 120
steps = 120

# Number of simulations
num_sims = 100000

# Weibull-parameters for the components
alpha = [2.0, 2.5, 3.0, 1.0, 1.5]
beta = [50.0, 60.0, 70.0, 40.0, 50.0]

# Threshold system weights and threshold
w = [2.0, 4.0, 2.3, 3.5, 1.9]
threshold = 5.0

# Number of components and minimal paths
num_comps = 5

# Birnbaum measures
IB = np.zeros((num_comps, steps))

# A component is identified as critical when t is in [L, U)
L = np.zeros(num_comps)
U = np.zeros(num_comps)

# Component state vector
x = np.zeros(num_comps)

# The structure function of the threshold system
def phi(xx):
    wsum = 0.0
    for i in range(num_comps):
        wsum += w[i] * xx[i]
    if wsum >= threshold:
        return 1
    else:
        return 0
```



Estimating Birnbaum measure using simulation (c.)

```
# The structure function of the threshold system
def phi(xx):
    wsum = 0.0
    for i in range(num_comps):
        wsum += w[i] * xx[i]
    if wsum >= threshold:
        return 1
    else:
        return 0

# Calculate the system lifetime given the component lifetimes
def sys_lifetime(tt):
    # Initialize component states
    for i in range(num_comps):
        x[i] = 1
    # Initialize system state and lifetime
    sys_state = phi(x)
    sys_life = 0.0
    # Determine the ordering of the failure times
    order = tt.argsort()
    # Initialize failure counter
    c = 0
    # Switch off one component at a time in the order of the failure times
    while (sys_state == 1) and (c < num_comps):
        x[order[c]] = 0
        sys_life = tt[order[c]]
        sys_state = phi(x)
        c += 1
    if sys_state == 0:
        return sys_life
    else: # This happens only for trivial systems where phi(0,...,0) = 1.
        return np.inf
```



Estimating Birnbaum measure using simulation (c.)

```
# The simulated lifetimes of the components
t = np.zeros(num_comps)

# Time steps
time = np.linspace(0.0, max_t, steps)

# Run simulations
for k in range(num_sims):
    for i in range(num_comps):
        t[i] = weibullvariate(beta[i], alpha[i])
    for i in range(num_comps):
        temp = t[i]
        t[i] = np.inf
        U[i] = sys_lifetime(t)
        t[i] = 0
        L[i] = sys_lifetime(t)
        t[i] = temp
    for step in range(steps):
        for i in range(num_comps):
            if L[i] <= time[step] and time[step] < U[i]:
                IB[i][step] += 1

# Calculate average importance values
for step in range(steps):
    for i in range(num_comps):
        IB[i][step] = IB[i][step] / num_sims

fig = plt.figure(figsize = (7, 4))
for i in range(num_comps):
    plt.plot(time, IB[i], label='IB(' + str(i+1) + ", t)")
plt.xlabel('Time')
plt.ylabel('Importance')
plt.title("Importance as a function of time")
plt.legend()
if save_plots:
    plt.savefig("birnbaum_03/birnbaum_03.pdf")
plt.show()
```



Exact Birnbaum measures using ROBDD

Instead of using MC simulations, we could use an ROBDD-representation of the system, and then calculate the Birnbaum importance measures as a functions of time, $I_B^{(1)}(t), \ldots, I_B^{(5)}(t)$ analytically.

RECALL: By using an ROBDD-representation we can calculate the reliability function $h(\boldsymbol{p})$ analytically for any $\boldsymbol{p}$.

We then choose a sufficiently dense set of t-values, $0 = t_0 < t_1 < \cdots < t_K$, and calculate $I_B^{(i)}(t)$, for these t-values.

The importance curve for $t \in [0, t_K]$ is then obtained by drawing a curve through the points:

$$(t_0, I_B^{(i)}(t_0)), (t_1, I_B^{(i)}(t_1)), \ldots, (t_K, I_B^{(i)}(t_K)).$$

If K is sufficiently large, a nice smooth curve is obtained.



Exact Birnbaum measure using ROBDD

```
from math import exp
import matplotlib.pyplot as plt
import numpy as np
from c_robdd import ROBDDSystem
from c_threshold import ROBDDThreshold

save_plots = True

# Time parameters
max_t = 120
steps = 120

# Weibull-parameters for the components
alpha = [2.0, 2.5, 3.0, 1.0, 1.5]
beta = [50.0, 60.0, 70.0, 40.0, 50.0]

# Number of components in the system
num_comps = len(alpha)

# Threshold system weights and threshold
a = [2.0, 4.0, 2.3, 3.5, 1.9]
b = 5.0

# Calculate the weight sum
asum = 0
for i in range(num_comps):
    asum += a[i]

# The ROBDD representation of the threshold system
sys = ROBDDSystem(ROBDDThreshold(a, asum, b))

# The reliability function of the system
def reliability(pp):
    result = sys.calculateReliability(pp)
    return result[1]
```



Exact Birnbaum measure using ROBDD (c.)

```
# The survival function of a Weibull-distribution
def weibull(aa, bb, t):
    if t >= 0:
        return exp(- (t / bb) ** aa)
    else:
        return 1.0

# Component reliability vector
p = np.zeros(num_comps)

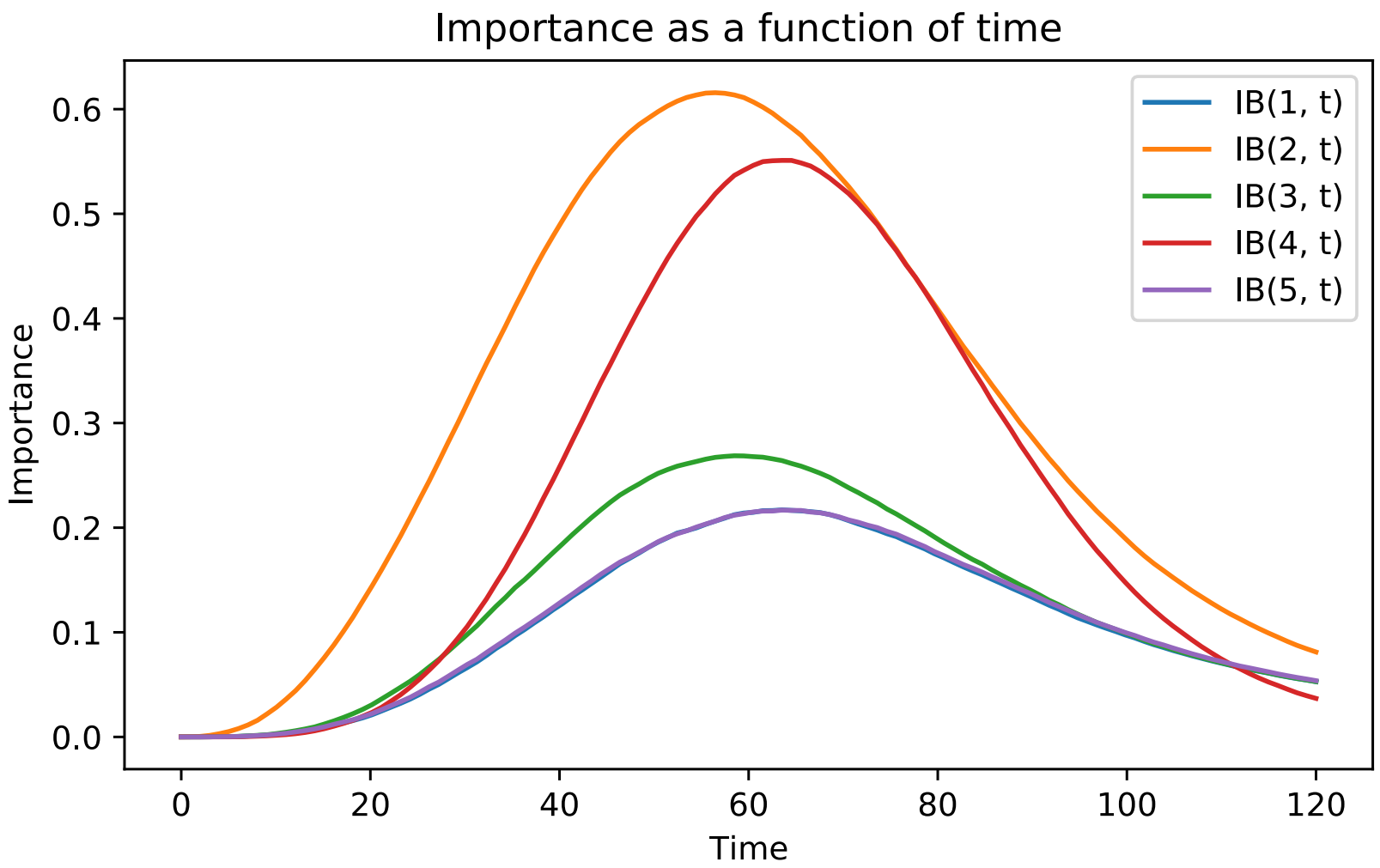
# Time values and Birnbaum measures
time = np.linspace(0.0, max_t, steps)
IB = np.zeros((num_comps, steps))

# Calculate the Birnbaum measures
for step in range(steps):
    for i in range(num_comps):
        p[i] = weibull(alpha[i], beta[i], time[step])
    for i in range(num_comps):
        temp = p[i]
        p[i] = 1
        h1 = reliability(p)
        p[i] = 0
        h0 = reliability(p)
        p[i] = temp
        IB[i][step] = h1 - h0

# Plot the results
fig = plt.figure(figsize = (7, 4))
for i in range(num_comps):
    plt.plot(time, IB[i], label='IB(' + str(i+1) + ", t)")
plt.xlabel('Time')
plt.ylabel('Importance')
plt.title("Importance as a function of time")
plt.legend()
if save_plots:
    plt.savefig("birnbaum_03b/birnbaum_03b.pdf")
plt.show()
```

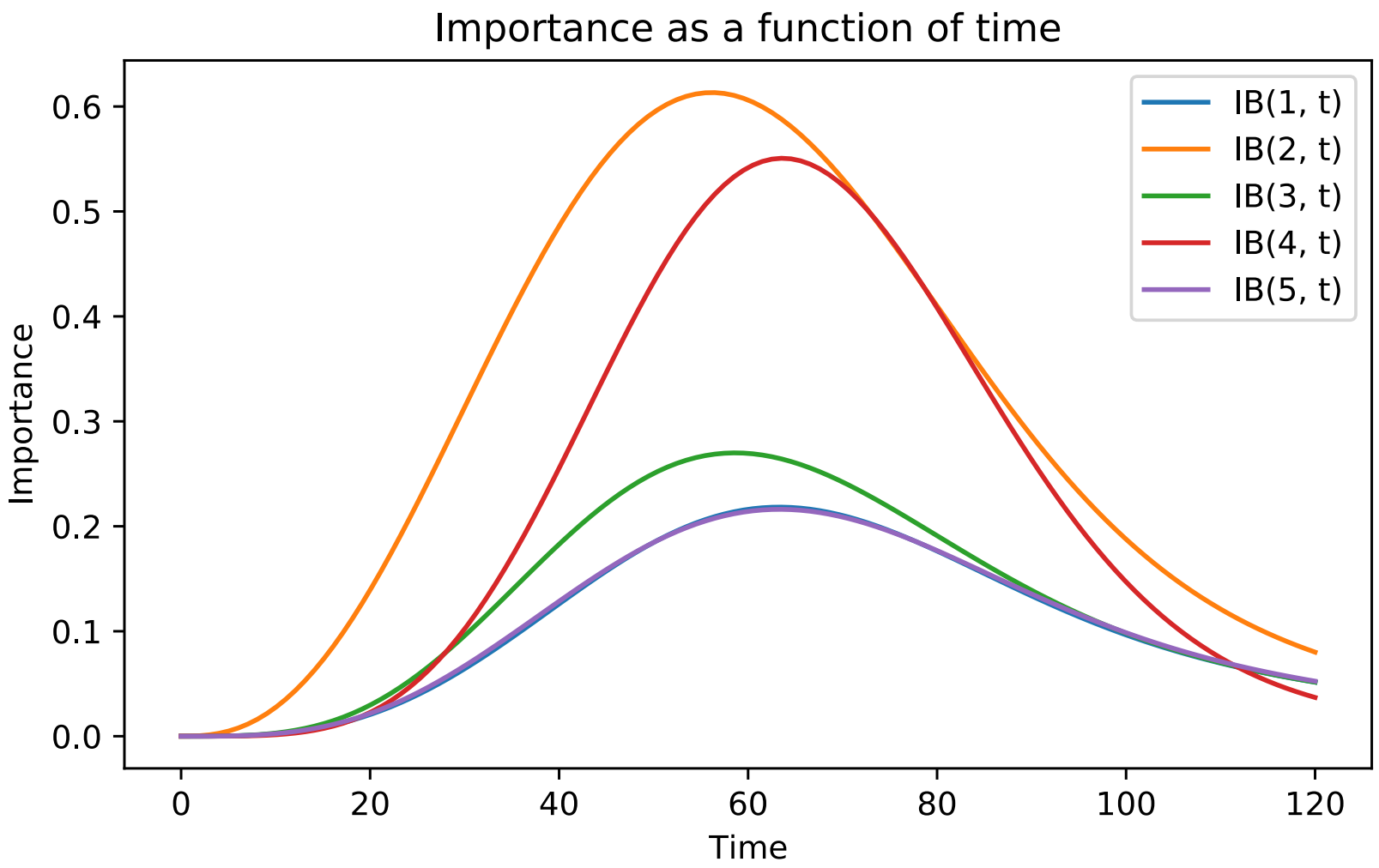


Estimated Birnbaum measures



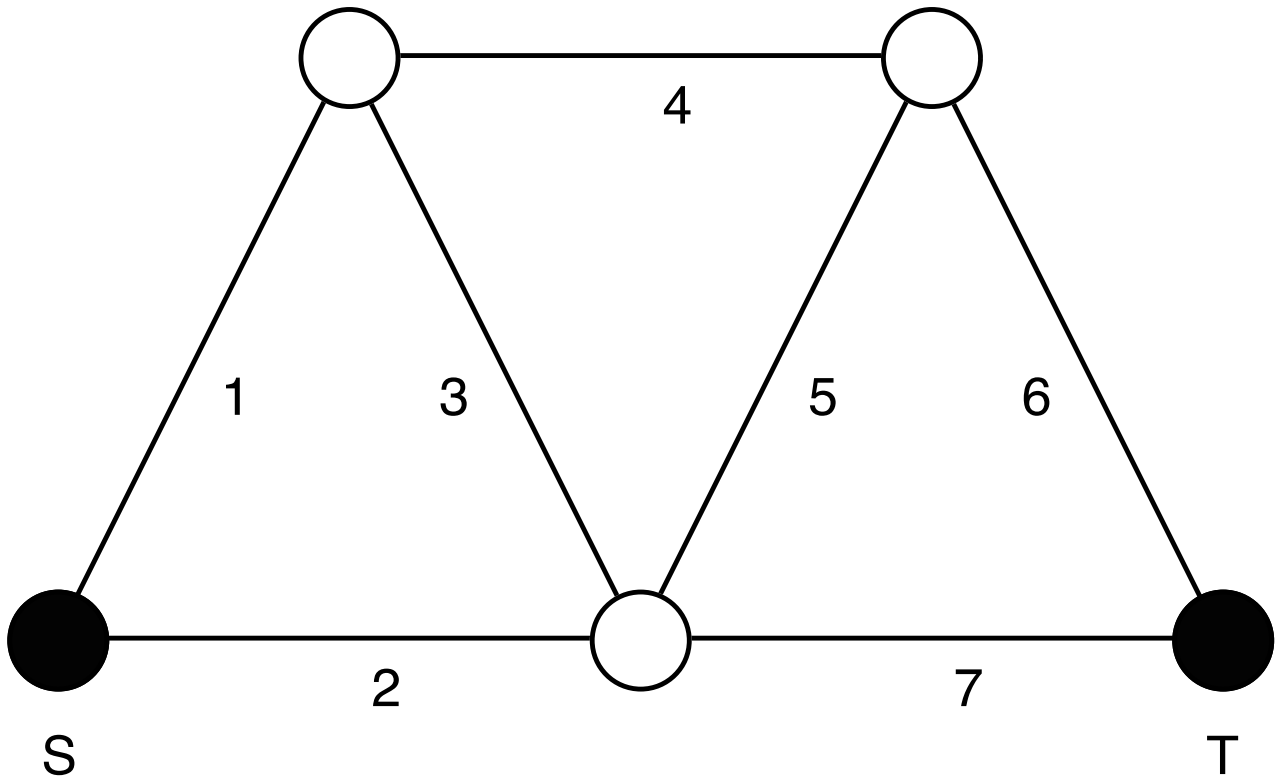


Exact Birnbaum measures





Estimating Birnbaum measure using simulation (c.)





Estimating Birnbaum measure using simulation (c.)

EXAMPLE: Let (C, ϕ) be an undirected network system with component set $C = \{1, \ldots, 7\}$. The structure function of this system is given by:

$$\phi(\boldsymbol{X}) = X_4 \cdot \phi(1_4, \boldsymbol{X}) + (1 - X_4) \cdot \phi(0_4, \boldsymbol{X}),$$

where:

$$\begin{aligned}\phi(1_4, \boldsymbol{X}) = &(X_3 \amalg X_5) \cdot (X_1 \amalg X_2) \cdot (X_6 \amalg X_7) \\ &+ (1 - X_3 \amalg X_5) \cdot ((X_1 X_6) \amalg (X_2 X_7))\end{aligned}$$

$$\phi(0_4, \boldsymbol{X}) = ((X_1 X_3) \amalg X_2) \cdot ((X_5 X_6) \amalg X_7)$$



Estimating Birnbaum measure using simulation (c.)

The lifetime of component i, T_i is Weibull-distributed with parameters α_i and β_i, for all $i \in C$ where:

$$\alpha_1 = 2.0, \quad \alpha_2 = 2.5, \quad \alpha_3 = 3.0, \quad \alpha_4 = 1.0,$$
$$\alpha_5 = 1.5, \quad \alpha_6 = 2.0, \quad \alpha_7 = 2.5,$$

$$\beta_1 = 50.0, \quad \beta_2 = 60.0, \quad \beta_3 = 70.0, \quad \beta_4 = 40.0,$$
$$\beta_5 = 50.0, \quad \beta_6 = 55.0, \quad \beta_7 = 65.0.$$

The Birnbaum measures $I_B^{(1)}(t), \ldots, I_B^{(7)}(t)$ for $t \geq 0$, are estimated by running a Monte Carlo simulation with $N = 100000$ simulations.



Estimating Birnbaum measure using simulation (c.)

```
from math import *
from random import *
import matplotlib.pyplot as plt
import numpy as np

save_plots = True

# Time parameters
max_t = 120
steps = 120

# Number of simulations
num_sims = 100000

# Weibull-parameters for the components
alpha = [2.0, 2.5, 3.0, 1.0, 1.5, 2.0, 2.5]
beta = [50.0, 60.0, 70.0, 40.0, 50.0, 55.0, 65.0]

# Number of components
num_comps = 7

# Birnbaum measures
IB = np.zeros((num_comps, steps))

# A component is identified as critical when t is in [L, U)
L = np.zeros(num_comps)
U = np.zeros(num_comps)

# Component state vector
x = np.zeros(num_comps)

# The coproduct function
def coprod(x1, x2):
    return x1 + x2 - x1 * x2
```



Estimating Birnbaum measure using simulation (c.)

```
# The structure function given that component 3 is functioning
def phi_3_1(xx):
    return coprod(xx[2], xx[4]) * coprod(xx[0], xx[1]) * coprod(xx[5], xx[6]) \
        + (1 - coprod(xx[2], xx[4])) * coprod(xx[0] * xx[5], xx[1] * xx[6])

# The structure function given that component 3 is failed
def phi_3_0(xx):
    return coprod(xx[0] * xx[2], xx[1]) * coprod(xx[4] * xx[5], xx[6])

# The structure function of the network system (using pivotal decomposition)
def phi(xx):
    return xx[3] * phi_3_1(xx) + (1 - xx[3]) * phi_3_0(xx)

# Calculate the system lifetime given the component lifetimes
def sys_lifetime(tt):
    # Initialize component states
    for i in range(num_comps):
        x[i] = 1
    # Initialize system state and lifetime
    sys_state = phi(x)
    sys_life = 0.0
    # Determine the ordering of the failure times
    order = tt.argsort()
    # Initialize failure counter
    c = 0
    # Switch off one component at a time in the order of the failure times
    while (sys_state == 1) and (c < num_comps):
        x[order[c]] = 0
        sys_life = tt[order[c]]
        sys_state = phi(x)
        c += 1
    if sys_state == 0:
        return sys_life
    else: # This happens only for trivial systems where phi(0,...,0) = 1.
        return np.inf
```



Estimating Birnbaum measure using simulation (c.)

```
# The simulated lifetimes of the components
t = np.zeros(num_comps)

# Time steps
time = np.linspace(0.0, max_t, steps)

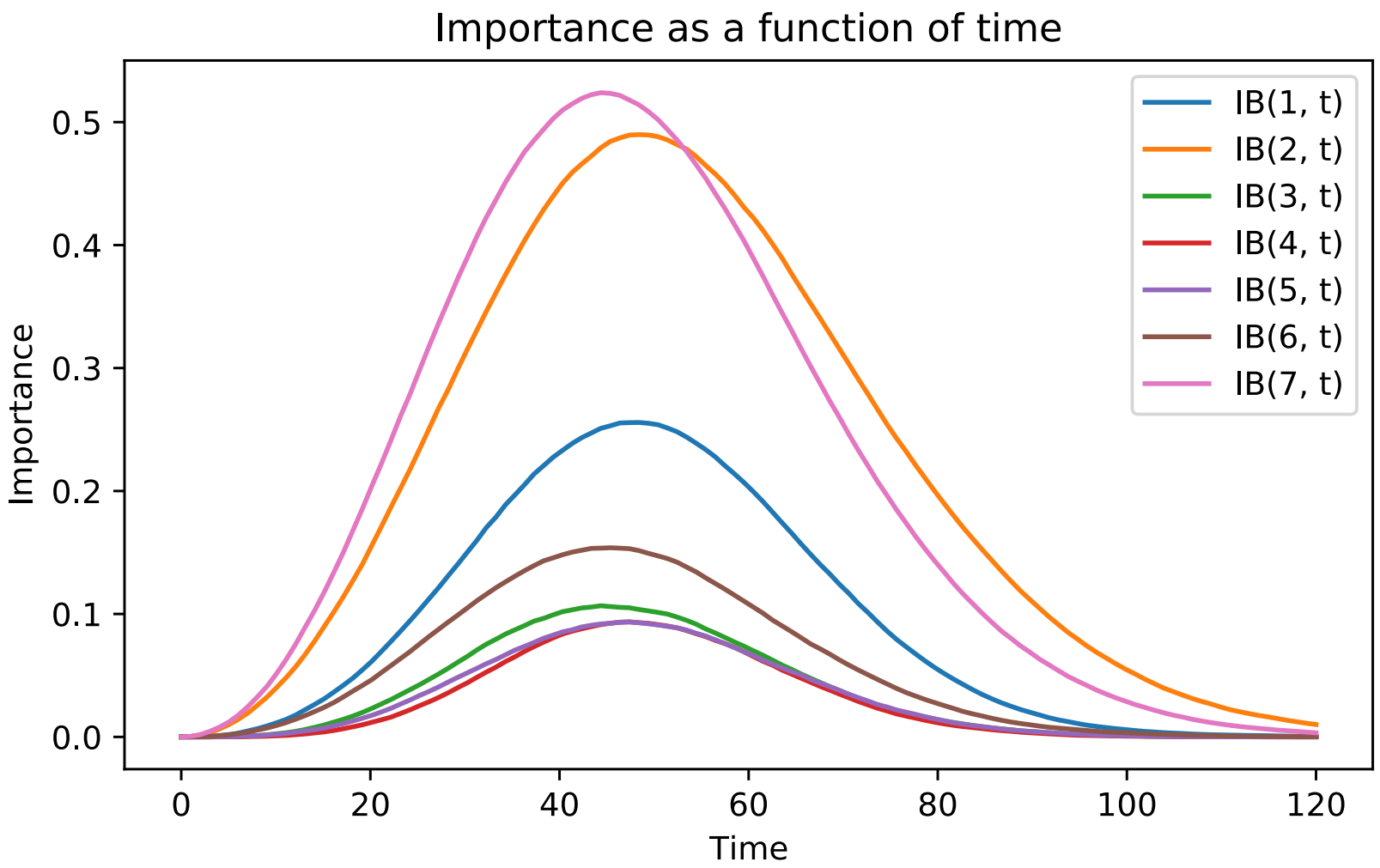
# Run simulations
for k in range(num_sims):
    for i in range(num_comps):
        t[i] = weibullvariate(beta[i], alpha[i])
    for i in range(num_comps):
        temp = t[i]
        t[i] = np.inf
        U[i] = sys_lifetime(t)
        t[i] = 0
        L[i] = sys_lifetime(t)
        t[i] = temp
    for step in range(steps):
        for i in range(num_comps):
            if L[i] <= time[step] and time[step] < U[i]:
                IB[i][step] += 1

# Calculate average importance values
for step in range(steps):
    for i in range(num_comps):
        IB[i][step] = IB[i][step] / num_sims

fig = plt.figure(figsize = (7, 4))
for i in range(num_comps):
    plt.plot(time, IB[i], label='IB(' + str(i+1) + ", t)")
plt.xlabel('Time')
plt.ylabel('Importance')
plt.title("Importance as a function of time")
plt.legend()
if save_plots:
    plt.savefig("birnbaum_04/birnbaum_04.pdf")
plt.show()
```



Estimating Birnbaum measure using simulation (c.)





The Barlow-Proschan measure of reliability importance



The Barlow-Proschan measure

Definition (Barlow-Proschan measure)

Let (C, ϕ) be a non-trivial binary monotone system where $C = \{1, \ldots, n\}$. Moreover, let T_i denote the lifetime of component i, $i \in C$, and let S denote the lifetime of the system.

The Barlow-Proschan measure of the reliability importance of component $i \in C$ is defined as:

$$\begin{aligned} I_{B-P}^{(i)} &= P(\text{Component } i \text{ fails at the same time as the system}) \\ &= P(T_i = S). \end{aligned}$$



The Barlow-Proschan measure (cont.)

Theorem (Probability of system failure)

Let (C, ϕ) be a non-trivial binary monotone system where $C = \{1, \ldots, n\}$. Moreover, let T_i denote the lifetime of component i, $i \in C$, and let S denote the lifetime of the system.

Assume that $T_1, \ldots, T_n$ are independent and absolutely continuously distributed.

Then S is absolutely continuously distributed as well, and we have:

$$\sum_{i=1}^{n} I_{B-P}^{(i)} = 1.$$



The Barlow-Proschan measure (cont.)

PROOF: Since we have assumed that the system is non-trivial, the lifetime of the system, S can be expressed as:

$$S = \max_{1 \leq j \leq p} \min_{i \in P_j} T_i,$$

where $P_1, \ldots, P_p$ are the minimal path sets of the system. This implies that:

$$P(\bigcup_{i=1}^{n} \{T_i = S\}) = 1.$$



The Barlow-Proschan measure (cont.)

Since $T_1, \ldots, T_n$ are absolutely continuously distributed, the probability of having two or more components failing at the same time is zero.

This implies e.g., that $P(\{T_i = S\} \cap \{T_j = S\}) = 0$ for $i \neq j$. Thus, when calculating the probability of the union of the events $\{T_i = S\}$, $i = 1, \ldots, n$, all intersections can be ignored as they have zero probability of occurring.

Hence, we get:

$$1 = P(\bigcup_{i=1}^{n}\{T_i = S\}) = \sum_{i=1}^{n} P(T_i = S) = \sum_{i=1}^{n} I_{B-P}^{(i)},$$

where the second equality follows by ignoring all intersections of events $\{T_i = S\}$, $i = 1, \ldots, n$.

The last equality follows by the definition of $I_{B-P}^{(i)}$, and hence, the proof is complete.



The Barlow-Proschan measure (cont.)

Theorem (Integral formula for the Barlow-Proschan measure)

Let (C, ϕ) be a non-trivial binary monotone system where $C = \{1, \ldots, n\}$, and let T_i denote the lifetime of component i, $i \in C$.

Assume that $T_1, \ldots, T_n$ are independent, absolutely continuously distributed with densities $f_1, \ldots, f_n$ respectively. Then, we have:

$$I_{B-P}^{(i)} = \int_0^\infty I_B^{(i)}(t) f_i(t) dt,$$

where $I_B^{(i)}(t)$ denotes the Birnbaum measure of the reliability importance of component i at time t.



The Barlow-Proschan measure (cont.)

PROOF: From the definitions of the Barlow-Proschan measure and the Birnbaum measure, it follows that:

$$
\begin{aligned}
I_{B-P}^{(i)} &= P(\text{Component } i \text{ fails at the same time as the system}) \\
&= \int_0^\infty P(\text{Component } i \text{ is critical at time } t) \cdot f_i(t)dt \\
&= \int_0^\infty I_B^{(i)}(t) f_i(t) dt.
\end{aligned}
$$



Estimating Barlow-Proschan measure using simulation

In order to compute the Barlow-Proschan measure the integral formula is not always easy to use. For this purpose it is actually better to use the original definition.

Using Monte Carlo simulation it is easy to estimate the probability that a given component fails at the same time as the system. By running N simulations we simply count the number of times this happens for each of the components.

For all $i \in C$ we let:

$$W_i = \text{Number of times comp. } i \text{ fails at the same time as the system}$$

The Barlow-Proschan measure of importance can then be estimated by the following unbiased estimator:

$$\hat{I}_{B-P}^{(i)} = \frac{W_i}{N}, \quad i \in C.$$



Estimating Barlow-Proschan measure (c.)

EXAMPLE: Let (C, ϕ) be a bridge system with $C = \{1, 2, 3, 4, 5\}$ and minimal path sets:

$$P_1 = \{1, 4\}, \quad P_2 = \{1, 3, 5\}, \quad P_3 = \{2, 3, 4\}, \quad P_4 = \{2, 5\}.$$

The lifetime of component i, T_i is Weibull-distributed with parameters α_i and β_i, for all $i \in C$ where:

$$\alpha_1 = 2.0, \quad \alpha_2 = 2.5, \quad \alpha_3 = 3.0, \quad \alpha_4 = 1.0, \quad \alpha_5 = 1.5,$$
$$\beta_1 = 50.0, \quad \beta_2 = 60.0, \quad \beta_3 = 70.0, \quad \beta_4 = 40.0, \quad \beta_5 = 50.0.$$

The Barlow-Proschan measures $I_{B-P}^{(1)}, \ldots, I_{B-P}^{(5)}$ for $t \geq 0$, are estimated by running a Monte Carlo simulation with $N = 100000$ simulations.



Estimating Barlow-Proschan measure (c.)

```
from math import *
from random import *
import matplotlib.pyplot as plt
import numpy as np

save_plots = True

# Number of simulations
num_sims = 100000

# Weibull-parameters for the components
alpha = [2.0, 2.5, 3.0, 1.0, 1.5]
beta = [50.0, 60.0, 70.0, 40.0, 50.0]

# Number of components and minimal paths
num_comps = 5
num_paths = 4

# Minimal path sets
paths = [{1,4}, {1,3,5}, {2,3,4}, {2,5}]

# Calculate the system lifetime given the component lifetimes
def sys_lifetime(tt):
    sys_life = 0.0
    for j in range(num_paths):
        path_life = np.inf
        for i in paths[j]:
            if tt[i-1] < path_life:
                path_life = tt[i-1]
        if path_life > sys_life:
            sys_life = path_life
    return sys_life
```



Estimating Barlow-Proschan measure (c.)

```
# The simulated lifetimes of the components
t = np.zeros(num_comps)

# The number of times the components are critical when failing
c = np.zeros(num_comps)

for k in range(num_sims):
    for i in range(num_comps):
        t[i] = weibullvariate(beta[i], alpha[i])
    s = sys_lifetime(t)
    for i in range(num_comps):
        if t[i] == s:
            c[i] += 1

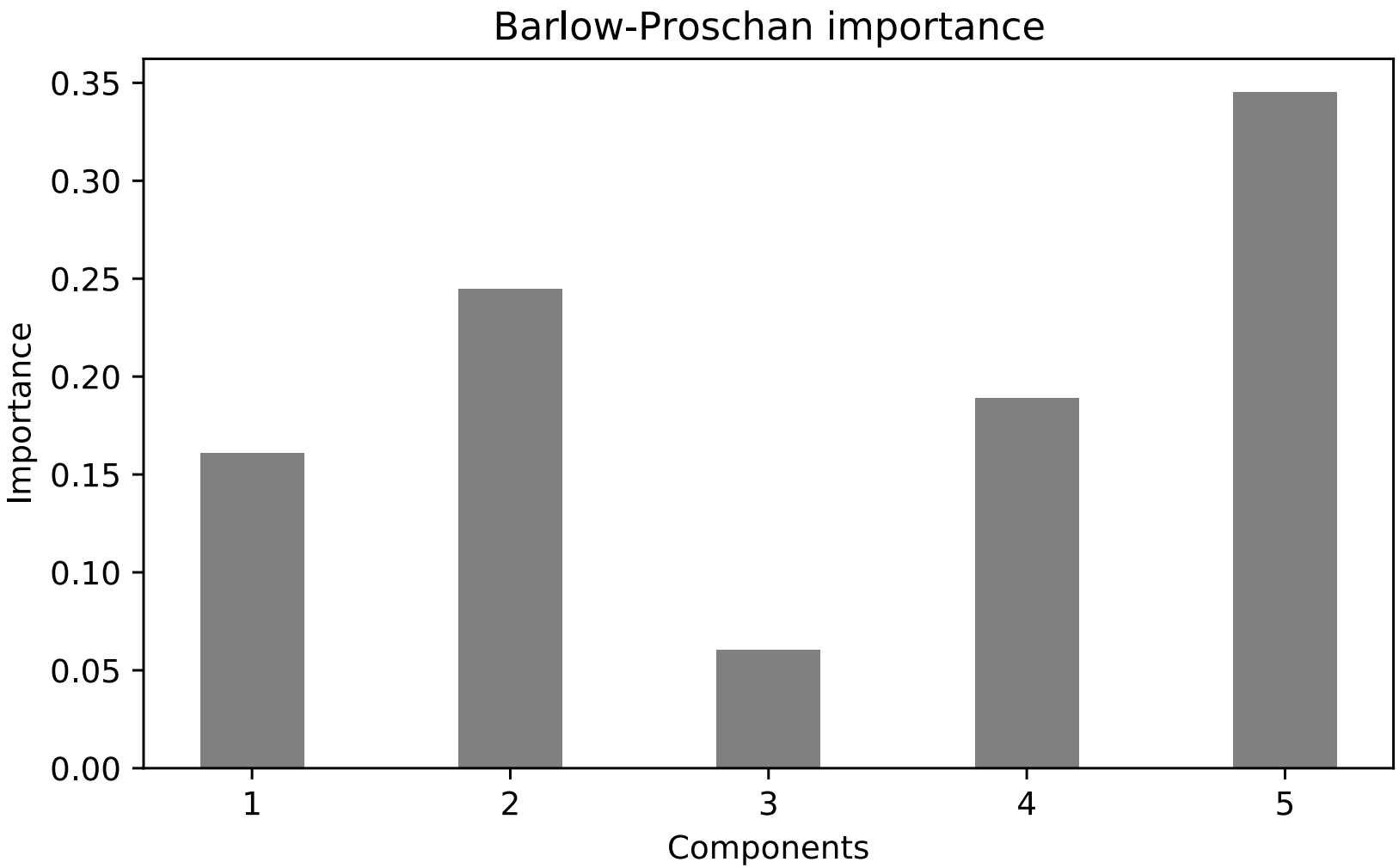
bp_imp = np.zeros(num_comps)
for i in range(num_comps):
    bp_imp[i] = c[i] / num_sims

comp = np.linspace(1, num_comps, num_comps)

fig = plt.figure(figsize = (7, 4))
plt.bar(comp, bp_imp, color ='gray', width = 0.4)
plt.xlabel("Components")
plt.ylabel("Importance")
plt.title("Barlow-Proschan importance")
if save_plots:
    plt.savefig("b_p_imp_01/importance_BP1.pdf")
plt.show()
```



Estimating Barlow-Proschan measure (c.)





Estimating Barlow-Proschan measure (c.)

EXAMPLE: Let (C, ϕ) be a threshold system with component set $C = \{1, 2, 3, 4, 5\}$ with structure function:

$$\phi(\boldsymbol{X}) = \mathrm{I}(\sum_{i=1}^{5} \omega_i X_i \geq 5.0)$$

where the component weights $\omega_1, \ldots, \omega_5$ are:

$$\omega_1 = 2.0, \quad \omega_2 = 4.0, \quad \omega_3 = 2.3, \quad \omega_4 = 3.5, \quad \omega_5 = 1.9$$

The lifetime of component i, T_i is Weibull-distributed with parameters α_i and β_i, for all $i \in C$ where:

$$\alpha_1 = 2.0, \quad \alpha_2 = 2.5, \quad \alpha_3 = 3.0, \quad \alpha_4 = 1.0, \quad \alpha_5 = 1.5,$$
$$\beta_1 = 50.0, \quad \beta_2 = 60.0, \quad \beta_3 = 70.0, \quad \beta_4 = 40.0, \quad \beta_5 = 50.0.$$

The Barlow-Proschan measures $I_{B-P}^{(1)}, \ldots, I_{B-P}^{(5)}$ for $t \geq 0$, are estimated by running a Monte Carlo simulation with $N = 100000$ simulations.



Estimating Barlow-Proschan measure (c.)

```
from math import *
from random import *
import matplotlib.pyplot as plt
import numpy as np

save_plots = True

# Number of simulations
num_sims = 100000

# Weibull-parameters for the components
alpha = [2.0, 2.5, 3.0, 1.0, 1.5]
beta = [50.0, 60.0, 70.0, 40.0, 50.0]

# Threshold system weights and threshold
w = [2.0, 4.0, 2.3, 3.5, 1.9]
threshold = 5.0

# Number of components and minimal paths
num_comps = 5

# Component state vector
x = np.zeros(num_comps)

# The structure function of the threshold system
def phi(xx):
    wsum = 0.0
    for i in range(num_comps):
        wsum += w[i] * xx[i]
    if wsum >= threshold:
        return 1
    else:
        return 0
```



Estimating Barlow-Proschan measure (c.)

```
# Calculate the system lifetime given the component lifetimes
def sys_lifetime(tt):
    # Initialize component states
    for i in range(num_comps):
        x[i] = 1
    # Initialize system state and lifetime
    sys_state = phi(x)
    sys_life = 0.0
    # Determine the ordering of the failure times
    order = tt.argsort()
    # Initialize failure counter
    c = 0
    # Switch off one component at a time in the order of the failure times
    while (sys_state == 1) and (c < num_comps):
        x[order[c]] = 0
        sys_life = tt[order[c]]
        sys_state = phi(x)
        c += 1
    if sys_state == 0:
        return sys_life
    else: # This happens only for trivial systems where phi(0,...,0) = 1.
        return np.inf
```



Estimating Barlow-Proschan measure (c.)

```
# The simulated lifetimes of the components
t = np.zeros(num_comps)

# The number of times the components are critical when failing
c = np.zeros(num_comps)

for k in range(num_sims):
    for i in range(num_comps):
        t[i] = weibullvariate(beta[i], alpha[i])
    s = sys_lifetime(t)
    for i in range(num_comps):
        if t[i] == s:
            c[i] += 1

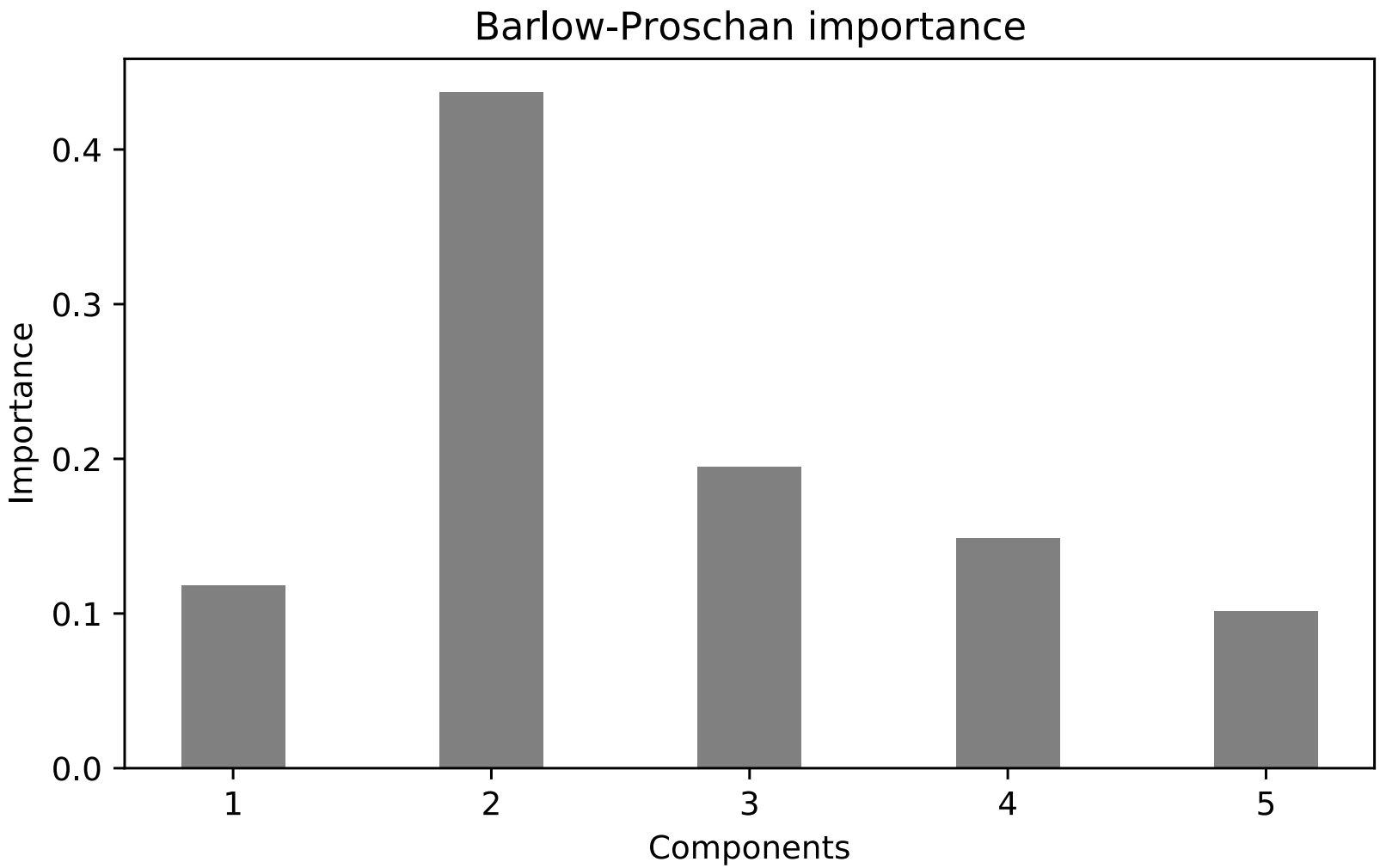
bp_imp = np.zeros(num_comps)
for i in range(num_comps):
    bp_imp[i] = c[i] / num_sims

comp = np.linspace(1, num_comps, num_comps)

fig = plt.figure(figsize = (7, 4))
plt.bar(comp, bp_imp, color ='gray', width = 0.4)
plt.xlabel("Components")
plt.ylabel("Importance")
plt.title("Barlow-Proschan importance")
if save_plots:
    plt.savefig("b_p_imp_02/importance_BP2.pdf")
plt.show()
```

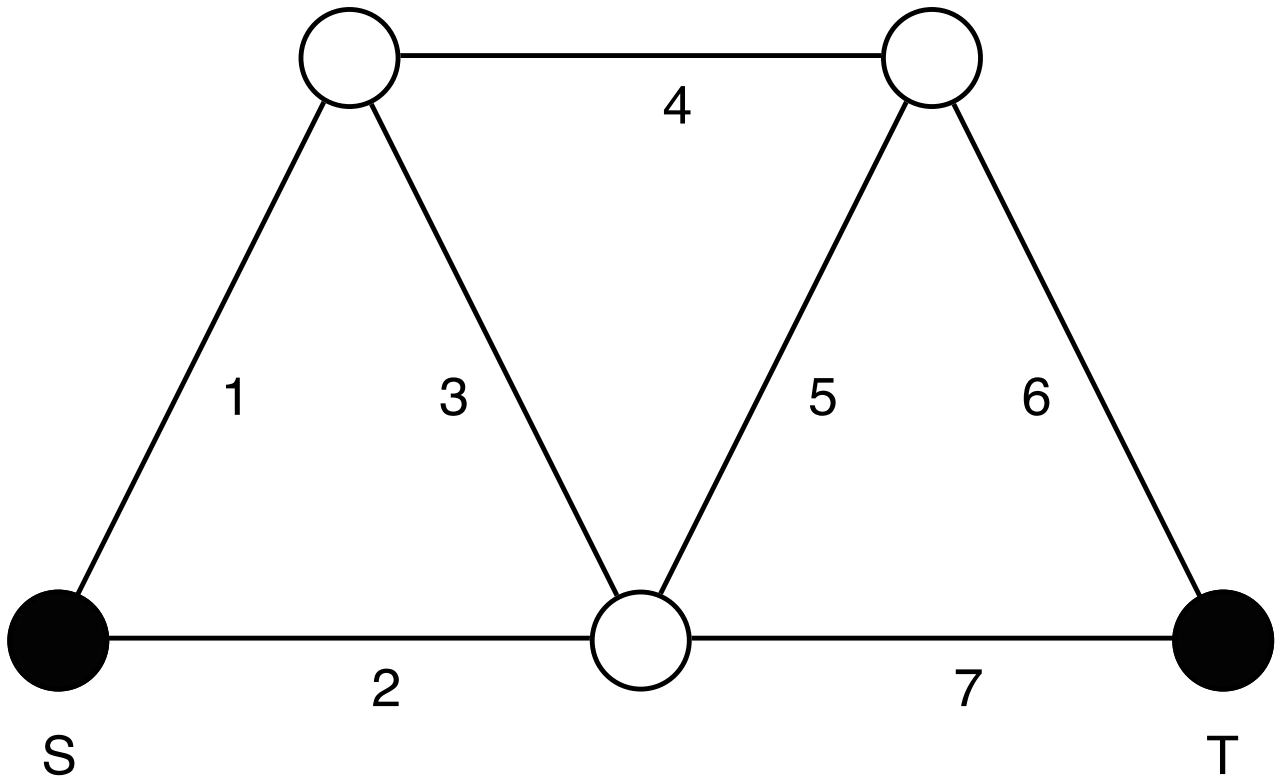


Estimating Barlow-Proschan measure (c.)





Estimating Birnbaum measure using simulation (c.)





Estimating Barlow-Proschan measure (c.)

EXAMPLE: Let (C, ϕ) be an undirected network system with component set $C = \{1, \ldots, 7\}$. The structure function of this system is given by:

$$\phi(\boldsymbol{X}) = X_4 \cdot \phi(1_4, \boldsymbol{X}) + (1 - X_4) \cdot \phi(0_4, \boldsymbol{X}),$$

where:

$$\begin{aligned}\phi(1_4, \boldsymbol{X}) &= (X_3 \amalg X_5) \cdot (X_1 \amalg X_2) \cdot (X_6 \amalg X_7) \\ &\quad + (1 - X_3 \amalg X_5) \cdot ((X_1 X_6) \amalg (X_2 X_7))\end{aligned}$$

$$\phi(0_4, \boldsymbol{X}) = ((X_1 X_3) \amalg X_2) \cdot ((X_5 X_6) \amalg X_7)$$



Estimating Birnbaum measure using simulation (c.)

The lifetime of component i, T_i is Weibull-distributed with parameters α_i and β_i, for all $i \in C$ where:

$$\alpha_1 = 2.0, \quad \alpha_2 = 2.5, \quad \alpha_3 = 3.0, \quad \alpha_4 = 1.0,$$
$$\alpha_5 = 1.5, \quad \alpha_6 = 2.0, \quad \alpha_7 = 2.5,$$

$$\beta_1 = 50.0, \quad \beta_2 = 60.0, \quad \beta_3 = 70.0, \quad \beta_4 = 40.0,$$
$$\beta_5 = 50.0, \quad \beta_6 = 55.0, \quad \beta_7 = 65.0.$$

The Barlow-Proschan measures $I_{B-P}^{(1)}, \ldots, I_{B-P}^{(7)}$ for $t \geq 0$, are estimated by running a Monte Carlo simulation with $N = 100000$ simulations.



Estimating Barlow-Proschan measure (c.)

```
from math import *
from random import *
import matplotlib.pyplot as plt
import numpy as np

save_plots = True

# Number of simulations
num_sims = 100000

# Weibull-parameters for the components
alpha = [2.0, 2.5, 3.0, 1.0, 1.5, 2.0, 2.5]
beta = [50.0, 60.0, 70.0, 40.0, 50.0, 55.0, 65.0]

# Number of components
num_comps = 7

# Component state vector
x = np.zeros(num_comps)

# The coproduct function
def coprod(x1, x2):
    return x1 + x2 - x1 * x2

# The structure function given that component 3 is functioning
def phi_3_1(xx):
    return coprod(xx[2], xx[4]) * coprod(xx[0], xx[1]) * coprod(xx[5], xx[6]) \
        + (1 - coprod(xx[2], xx[4])) * coprod(xx[0] * xx[5], xx[1] * xx[6])

# The structure function given that component 3 is failed
def phi_3_0(xx):
    return coprod(xx[0] * xx[2], xx[1]) * coprod(xx[4] * xx[5], xx[6])

# The structure function of the network system (using pivotal decomposition)
def phi(xx):
    return xx[3] * phi_3_1(xx) + (1 - xx[3]) * phi_3_0(xx)
```



Estimating Barlow-Proschan measure (c.)

```
# Calculate the system lifetime given the component lifetimes
def sys_lifetime(tt):
    # Initialize component states
    for i in range(num_comps):
        x[i] = 1
    # Initialize system state and lifetime
    sys_state = phi(x)
    sys_life = 0.0
    # Determine the ordering of the failure times
    order = tt.argsort()
    # Initialize failure counter
    c = 0
    # Switch off one component at a time in the order of the failure times
    while (sys_state == 1) and (c < num_comps):
        x[order[c]] = 0
        sys_life = tt[order[c]]
        sys_state = phi(x)
        c += 1
    if sys_state == 0:
        return sys_life
    else: # This happens only for trivial systems where phi(0,...,0) = 1.
        return np.inf
```



Estimating Barlow-Proschan measure (c.)

```
# The simulated lifetimes of the components
t = np.zeros(num_comps)

# The number of times the components are critical when failing
c = np.zeros(num_comps)

for k in range(num_sims):
    for i in range(num_comps):
        t[i] = weibullvariate(beta[i], alpha[i])
    s = sys_lifetime(t)
    for i in range(num_comps):
        if t[i] == s:
            c[i] += 1

bp_imp = np.zeros(num_comps)
for i in range(num_comps):
    bp_imp[i] = c[i] / num_sims

comp = np.linspace(1, num_comps, num_comps)

fig = plt.figure(figsize = (7, 4))
plt.bar(comp, bp_imp, color ='gray', width = 0.4)
plt.xlabel("Components")
plt.ylabel("Importance")
plt.title("Barlow-Proschan importance")
if save_plots:
    plt.savefig("b_p_imp_03/importance_BP3.pdf")
plt.show()
```



Estimating Barlow-Proschan measure (c.)

