# STK3405 – Lecture 12
# Part 2

A. B. Huseby

Department of Mathematics
University of Oslo, Norway



# Chapter 6

## Computing reliability using conditional Monte Carlo methods



# Section 6.3

**System reliability when all the component state variables have identical reliabilities**



# Identical component reliabilities

If $p_1 = \cdots = p_n = p$, it is possible to improve things even further. Then $S$ has a binomial distribution, and the conditional distribution of $\boldsymbol{X}$ given $S$ is given by:

$$P(\boldsymbol{X} = \boldsymbol{x} \mid S = s) = \frac{p^{\sum_{i=1}^n x_i}(1-p)^{n-\sum_{i=1}^n x_i} \cdot I(\sum_{i=1}^n x_i = s)}{\binom{n}{s} p^s (1-p)^{n-s}} = \frac{1}{\binom{n}{s}} ,$$

for all $\boldsymbol{x}$ such that $\sum_{i=1}^n x_i = s$.

From this it follows that:

$$\theta_s = E[\phi \mid S = s] = \sum_{\{\boldsymbol{x} \mid \sum_{i=1}^n x_i = s\}} \phi(\boldsymbol{x}) P(\boldsymbol{X} = \boldsymbol{x} \mid S = s) = \frac{b_s}{\binom{n}{s}} ,$$

where $b_s$ is the number of path sets with $s$ components, $s = 0, \ldots, n$.



# Identical component reliabilities (cont.)

The system reliability, $h$, expressed as a function of $p$, is given by:

$$h(p) = \sum_{s=0}^{n} \theta_s P(S = s) = \sum_{s=0}^{n} \frac{b_s}{\binom{n}{s}} \binom{n}{s} p^s (1-p)^{n-s}.$$

NOTE 1: $\theta_0, \ldots, \theta_n$ do not depend on $p$. Thus, by estimating these quantities, we get an estimate of $h(p)$ for all $p \in [0, 1]$.

NOTE 2: $\theta_s$ can be interpreted as the fraction of path sets of size $s$ among all sets of size $s$. Thus, $\theta_s$ can be estimated by sampling random sets of size $s$ and calculating the frequency of path sets among the sampled sets.

NOTE 3: Assuming that $(C, \phi)$ is non-trivial we have $\theta_0 = 0$. Thus, we can focus on estimating $\theta_1, \ldots, \theta_n$.



# Identical component reliabilities (cont.)

In the following we let:

$$\phi(A) = \phi(\mathbf{1}^A, \mathbf{0}), \quad \text{for all } A \subseteq C.$$

**ALGORITHM:** For $i = 1, \ldots, N$ do:

STEP 1. Generate a random permutation $P_i = (c_{i,1}, \ldots, c_{i,n})$ of the component set $C$.

STEP 2. Let $A_{i,s} = \{c_{i,1}, \ldots, c_{i,s}\}, \quad s = 1, \ldots, n.$

STEP 3. Calculate $\phi(A_{i,s}), \quad s = 1, \ldots, n.$

Note that for $s = 1, \ldots, n$, $A_{i,s}$ is a random set of size $s$, $i = 1, \ldots, N$. Hence, unbiased estimators for $\theta_1, \ldots, \theta_n$ are given by:

$$\hat{\theta}_s = \frac{1}{N} \sum_{i=1}^{N} \phi(A_{i,s}), \quad s = 1, \ldots, n.$$



# Identical component reliabilities (cont.)

Using the estimates $\hat{\theta}_1, \ldots, \hat{\theta}_n$ as well as $\hat{\theta}_0 = 0$, we get the following improved conditional Monte Carlo estimate:

$$\hat{h}_{CMC}(p) = \sum_{s=0}^{n} \hat{\theta}_s P(S = s)$$

$$= \sum_{s=0}^{n} \hat{\theta}_s \binom{n}{s} p^s (1-p)^{n-s}$$

NOTE: With this method all the $\theta_s$'s are estimated from the full sample of $N$. Thus, we do not need to partition the sample into subsamples.

The downside is that $\hat{\theta}_1, \ldots, \hat{\theta}_n$ become dependent. This makes it more difficult to calculate the resulting variance of $\hat{h}_{CMC}(p)$.



# Identical component reliabilities (cont.)

As a comparison we describe how we can estimate $h(p)$ using a crude Monte Carlo method.

If $U \sim \text{Unif}(0,1)$, we have:

$$P(U \leq p) = p, \quad 0 \leq p \leq 1.$$

Hence, if $X(p) = I(U \leq p)$, it follows that $X(p) \sim \text{Binary}(p)$, and in particular:

$$P(X(p) = 1) = P(U \leq p) = p.$$

To generate $X_1(p), \ldots, X_N(p)$ so that $X_i(p) \sim \text{Binary}(p)$, $i = 1, \ldots, N$, we generate $U_1, \ldots, U_N$, where $U_i \sim \text{Unif}(0,1)$, $i = 1, \ldots, N$, and then let:

$$X_i(p) = I(U_i \leq p), \quad i = 1, \ldots, N.$$



# Identical component reliabilities (cont.)

Now let $(C, \phi)$ be a binary monotone system where $C = \{1, \ldots, n\}$, and let $\boldsymbol{X} = (X_1, \ldots, X_n)$ be the vector of the component state variables. We also assume that:

$$P(X_j = 1) = p, \quad j = 1, \ldots, n.$$

In order to estimate $h(p) = E[\phi(\boldsymbol{X})]$ as a function of $p \in [0, 1]$, we generate $N$ vectors of independent uniformly distributed variables:

$$\boldsymbol{U}_i = (U_{i,1}, \ldots, U_{i,n}), \quad i = 1, \ldots, N,$$

and let $\boldsymbol{X}_i(p) = (X_{i,1}(p), \ldots, X_{i,n}(p))$, $i = 1, \ldots, N$ by:

$$X_{i,j}(p) = I(U_{i,j} \leq p), \quad i = 1, \ldots, N, \; j = 1, \ldots, n.$$



# Identical component reliabilities (cont.)

We then estimate $h(p) = E[\phi(\boldsymbol{X})]$ as a function of $p \in [0, 1]$ by:

$$\hat{h}_{MC}(p) = \frac{1}{N} \sum_{i=1}^{N} \phi(\boldsymbol{X}_i(p))$$

Since $X_{i,j}(p)$ is non-decreasing in $p$ and $\phi$ is non-decreasing in each argument, it follows that $\hat{h}_{MC}(p)$ is a non-decreasing function of $p$.

When running the simulation, it is not possible to store $\phi(\boldsymbol{X}_i(p))$ for all values of $p \in [0, 1]$. Instead, we let:

$$0 = p_0 \leq p_1 \leq \cdots \leq p_K = 1$$

where $K$ is sufficiently large, and estimate:

$$\hat{h}_{MC}(p_r) = \frac{1}{N} \sum_{i=1}^{N} \phi(\boldsymbol{X}_i(p_r)), \quad r = 0, 1, \ldots, K.$$



# Example – Bridge structure (cont.)

```
################################################
#                                              #
#    Estimate reliability function using       #
#    crude Monte Carlo simulations.            #
#                                              #
################################################

import matplotlib.pyplot as plt
import numpy as np

# Number of simulations
num_sims = 1000

# Random number generator seed. Set to 'None' for a random sequence
seed_num = 1234
gen = np.random.default_rng(seed = seed_num)

# Number of points
num_points = 101

# Save plot to file or not?
save_plot = True

# Name of system
sys_name = "bridge"

# Number of components:
n = 5
```



# Example – Bridge structure (cont.)

```
def coprod(x, y):
    return x + y - x*y

def phi(xx):
    return xx[3] * coprod(xx[1], xx[2]) * coprod(xx[4], xx[5]) + (1-xx[3]) * coprod(xx[1] * xx[4], xx[2] * xx[5])

def hh(pp):
    return pp * coprod(pp, pp) * coprod(pp, pp) + (1-pp)* coprod(pp * pp, pp * pp)

X = np.zeros(n + 1, dtype = int)        # The component state variables (X[0] is not used)

p = np.linspace(0, 1, num_points)
T = np.zeros(num_points, dtype = int)

for _ in range(num_sims):
    U = gen.uniform(0.0, 1.0, n + 1)    # Uniform variables (U[0] is not used)
    for j in range(num_points):
        for i in range(1, n + 1):
            if U[i] <= p[j]:
                X[i] = 1
            else:
                X[i] = 0
        T[j] += phi(X)
```



# Example – Bridge structure (cont.)

```
h_hat = [T[i] / num_sims for i in range(num_points)]

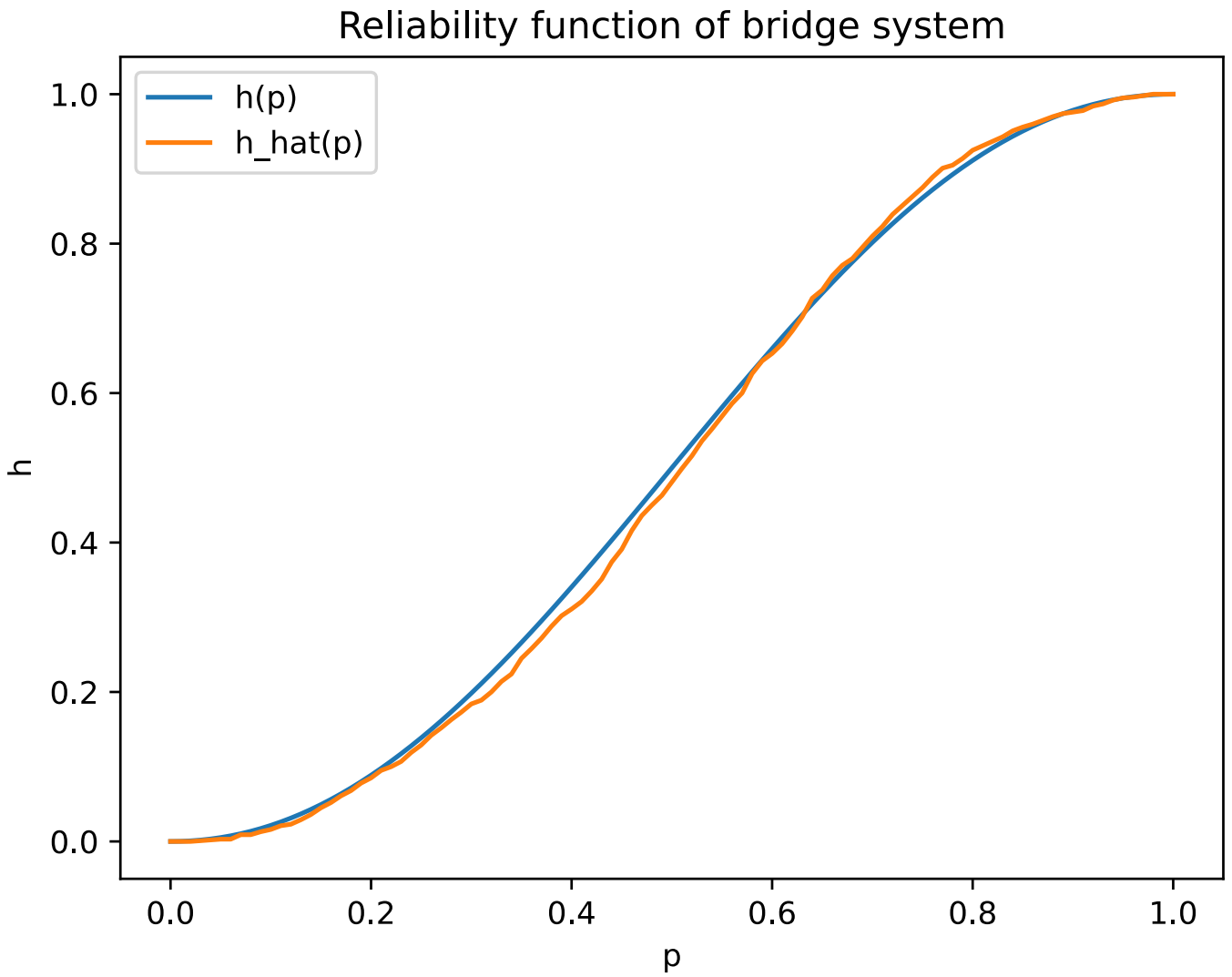
h = np.zeros(num_points)                                    # True reliability function

for j in range(num_points):
    h[j] = hh(p[j])


plt.plot(p, h, label='h(p)')
plt.plot(p, h_hat, label='h_hat(p)')
plt.xlabel('p')
plt.ylabel('h')
plt.title('Reliability function of ' + sys_name + " system")
plt.legend()
if save_plot:
    plt.savefig("crude/" + sys_name + ".pdf")
plt.show()
```



# Example – Bridge structure (cont.)






# Example – Bridge structure (cont.)

```
################################################
#                                              #
#    Estimate reliability function using       #
#    conditional Monte Carlo simulations.      #
#                                              #
################################################

from scipy.stats import binom
import matplotlib.pyplot as plt
import numpy as np

# Number of simulations
num_sims = 1000

# Random number generator seed. Set to 'None' for a random sequence
seed_num = 1234
gen = np.random.default_rng(seed = seed_num)

# Number of points
num_points = 101

# Save plot to file or not?
save_plot = True

# Name of system
sys_name = "bridge"

# Number of components:
n = 5
```



# Example – Bridge structure (cont.)

```
def coprod(x, y):
    return x + y - x*y

def phi(xx):
    return xx[3] * coprod(xx[1], xx[2]) * coprod(xx[4], xx[5]) + (1-xx[3]) * coprod(xx[1] * xx[4], xx[2] * xx[5])

def hh(pp):
    return pp * coprod(pp, pp) * coprod(pp, pp) + (1-pp)* coprod(pp * pp, pp * pp)


C = list(range(1, n + 1))              # The component set [1, 2, ..., n]
X = np.zeros(n + 1, dtype = int)       # The component state variables (X[0] is not used)
T = np.zeros(n + 1, dtype = int)       # T[s] counts the number of random path sets of size s = 0, 1, ..., n

for _ in range(num_sims):
    for i in range(1, n + 1):
        X[i] = 0
    sys_state = phi(X)                 # phi(0,...0) will always be zero unless we have a trivial system
    T[0] += sys_state
    gen.shuffle(C)                     # Generate a random permutation of the component set C
    for i in range(1, n + 1):
        X[C[i-1]] = 1
        if sys_state == 0:             # If sys_state = 1 already, we know phi(X) = 1 since phi is non-decreasing
            sys_state = phi(X)
        T[i] += sys_state

s_values = list(range(n + 1))                          # Set of possible values of S = X[1] + ... + X[n]

theta_hat = [T[s] / num_sims for s in s_values]        # theta_hat[s] = Estimated conditional reliability given S = s
print(theta_hat)
```



# Example – Bridge structure (cont.)

```
p = np.linspace(0, 1, num_points)
h = np.zeros(num_points)                                    # True reliability function
h_hat = np.zeros(num_points)                                # Estimated reliability function

for j in range(num_points):
    h[j] = hh(p[j])
    dist = [binom.pmf(s, n, p[j]) for s in s_values]
    for s in range(n + 1):
        h_hat[j] += theta_hat[s] * dist[s]

plt.plot(p, h, label='h(p)')
plt.plot(p, h_hat, label='h_hat(p)')
plt.xlabel('p')
plt.ylabel('h')
plt.title('Reliability function of ' + sys_name + " system")
plt.legend()
if save_plot:
    plt.savefig("cmc/" + sys_name + ".pdf")
plt.show()
```



# Example – Bridge structure (cont.)

