University of Oslo
Department of Mathematics

Fall, 2023
Mandatory assingment: 1

Author: Jonas Semprini Næss

STK3405/4405: Introduction to Risk and Reliability Analysis

Problem 1.

a.) *Find all the minimal path and cut sets of the system* (C, ϕ)*, and determine:*

$$d = \text{The length of the shortest path}$$

$$c = \text{The length of the shortest cut}$$

Solution:

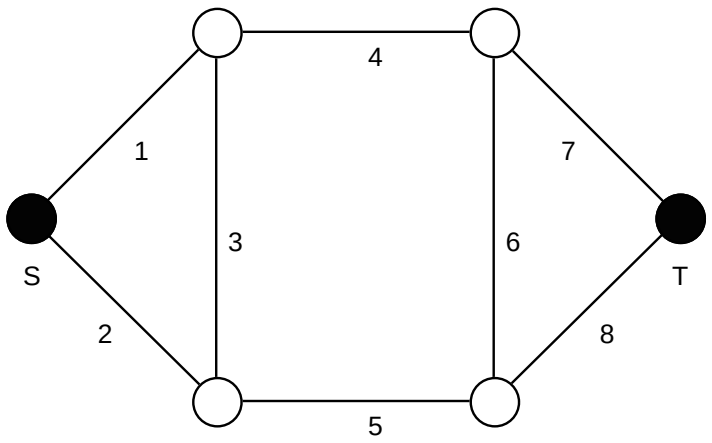

Figure 1: Visual of the 2-terminal undirected network system

We recall that a minimal path set is a path set that cannot be reduced without loosing the validity of being a path set. In this respective example, the following path sets are minimal

$$P_1 = \{1, 4, 7\} \;\; P_2 = \{2, 5, 8\}$$

$$P_3 = \{1, 3, 5, 8\} \;\; P_4 = \{2, 5, 6, 7\} \;\; P_5 = \{1, 4, 6, 8\} \;\; P_6 = \{2, 3, 4, 7\}$$

$$P_7 = \{1, 3, 5, 6, 7\} \;\; P_8 = \{2, 3, 4, 6, 8\}\,.$$

Meaning the 2-terminal system has has two minimal path sets of order 3, four minimal path sets of order 4 and two minimal path sets of order 5. Hence is

$$d = \min_{P_i \in P} |P_i| = 3, \quad i = 1, \ldots, 8.$$

Next we turn to the cut sets and apply the same reasoning, which yields

$$C_1 = \{1,2\}\ \ C_2 = \{7,8\}\ \ C_3 = \{4,5\}$$
$$C_4 = \{1,3,5\}\ \ C_5 = \{2,3,4\}$$
$$C_6 = \{2,5,6,7\}\ \ C_7 = \{1,4,6,8\}\,.$$

Thus has the 2-terminal system three minimal cut sets of order 2, two minimal cut sets of order 3, and two minimal cut sets of order 4. Then

$$d = \min_{C_j \in C} |C_j| = 2, \quad j = 1, \ldots, 7.$$

b.) *Show that the structure function of the system* (C, ϕ)*, can be written as:*

$$\begin{aligned}\phi(\boldsymbol{X}) = {} & X_3X_6 \cdot \left(X_1 \coprod X_2\right) \cdot \left(X_4 \coprod X_5\right) \cdot \left(X_7 \coprod X_8\right) \\ & + X_3(1-X_6) \cdot \left(X_1 \coprod X_2\right) \cdot \left(\left(X_4X_7 \coprod X_5X_8\right)\right) \\ & + (1-X_3)(X_6) \cdot \left(\left(X_1X_4 \coprod X_2X_5\right)\right) \cdot \left(X_7 \coprod X_8\right) \\ & + (1-X_3)(1-X_6) \cdot \left(\left(X_1X_4X_7 \coprod X_2X_5X_8\right)\right)\end{aligned}$$

Solution:

To solve this problem we are going to apply pivotal decomposition, and firstly notice that we can write it as follows

$$\phi_1(\boldsymbol{x}) = x_i\phi(1_i, \boldsymbol{x}) + (1-x_i)\phi(0_i, \boldsymbol{x}), \quad i = 1, \ldots, 8 \tag{1}$$
$$\phi_2(\boldsymbol{x}) = x_j\phi(1_j, \boldsymbol{x}) + (1-x_j)\phi(0_j, \boldsymbol{x}), \quad j = 1, \ldots, 8,\ \neq i \tag{2}$$

where $\phi(\boldsymbol{x}) = \phi_1(\boldsymbol{x}) \cdot \phi_2(\boldsymbol{x})$ [1]. Now introduce component three, which gives

$$\begin{aligned}\phi_1(\boldsymbol{X}) &= X_3\phi(1_3, \boldsymbol{X}) + (1-X_3)\phi(0_3, \boldsymbol{X}) \\ &= X_3\left(X_1 \coprod X_2\right) \cdot \left(X_4 \coprod X_5\right) + (1-X_3)\left(X_1X_4 \coprod X_2X_5\right)\end{aligned}$$

and equally for component six

$$\begin{aligned}\phi_2(\boldsymbol{X}) &= X_6\phi(1_6, \boldsymbol{X}) + (1-X_6)\phi(0_6, \boldsymbol{X}) \\ &= X_6\left(X_7 \coprod X_8\right) + (1-X_6)\left(X_4X_7 \coprod X_5X_8\right)\end{aligned}$$

[1] Note that this looks very much like module decomposition (even though it is not due to the given sets of states not having an empty intersection), but it was easier to specify this way.

Where we neglect the $(X_4 \coprod X_5)$ term due to redundancy on the system state. Hence do we have that

$$\begin{aligned}
\phi(\boldsymbol{x}) &= X_3 \left(X_1 \coprod X_2\right) \cdot \left(X_4 \coprod X_5\right) + (1 - X_3) \left(X_1 X_4 \coprod X_2 X_5\right) \\
&\cdot \left(X_6 \left(X_7 \coprod X_8\right) + (1 - X_6) \left(X_4 X_7 \coprod X_5 X_8\right)\right) \\
&= X_3 X_6 \left(X_1 \coprod X_2\right) \cdot \left(X_4 \coprod X_5\right) \cdot \left(X_7 \coprod X_8\right) \\
&+ X_3 (1 - X_6) \cdot \left(X_1 \coprod X_2\right) \cdot \left(\left(X_4 X_7 \coprod X_5 X_8\right)\right) \\
&+ (1 - X_3)(X_6) \cdot \left(\left(X_1 X_4 \coprod X_2 X_5\right)\right) \cdot \left(X_7 \coprod X_8\right) \\
&+ (1 - X_3)(1 - X_6) \left(X_1 X_4 \coprod X_2 X_5\right) \left(X_4 X_7 \coprod X_5 X_8\right)
\end{aligned}$$

where $(X_1 X_4 \coprod X_2 X_5) \cdot (X_4 X_7 \coprod X_5 X_8) = (X_1 X_4 X_7 \coprod X_2 X_5 X_8)$, since X_1, X_4, X_7 and X_2, X_5, X_8 are in respective series systems when component three and six are not working. Thus is

$$\begin{aligned}
\phi(\boldsymbol{X}) &= X_3 X_6 \cdot \left(X_1 \coprod X_2\right) \cdot \left(X_4 \coprod X_5\right) \cdot \left(X_7 \coprod X_8\right) \\
&+ X_3 (1 - X_6) \cdot \left(X_1 \coprod X_2\right) \cdot \left(\left(X_4 X_7 \coprod X_5 X_8\right)\right) \\
&+ (1 - X_3)(X_6) \cdot \left(\left(X_1 X_4 \coprod X_2 X_5\right)\right) \cdot \left(X_7 \coprod X_8\right) \\
&+ (1 - X_3)(1 - X_6) \cdot \left(\left(X_1 X_4 X_7 \coprod X_2 X_5 X_8\right)\right).
\end{aligned}$$

And we are done. ■

c.) *Use the result from (b) to find a similar expression for:*

$$h(\boldsymbol{p}) = P(\phi(\boldsymbol{X}) = 1) = E[\phi(\boldsymbol{X})]$$

Solution:

The reliability of a binary monotone system we remember is defined as the probability that the system is functioning, which more broadly can be described as

$$h = \mathbb{E}[\phi(\boldsymbol{X})] = \sum_{x \in \{0,1\}^n} \phi(\boldsymbol{X}) \mathbb{P}(\boldsymbol{X} = \boldsymbol{x}).$$

This implies that are we able, in like manner, as for the structure function, to calculate

the reliability of $\phi(\boldsymbol{X})$ to be

$$
\begin{aligned}
h(\boldsymbol{p}) = p_3 p_6 \cdot \left(p_1 \coprod p_2\right) \cdot \left(p_4 \coprod p_5\right) \cdot \left(p_7 \coprod p_8\right) \\
+ p_3(1-p_6) \cdot \left(p_1 \coprod p_2\right) \cdot \left(\left(p_4 p_7 \coprod p_5 p_8\right)\right) \\
+ (1-p_3)(p_6) \cdot \left(\left(p_1 p_4 \coprod p_2 p_5\right)\right) \cdot \left(p_7 \coprod p_8\right) \\
+ (1-p_3)(1-p_6) \cdot \left(\left(p_1 p_4 p_7 \coprod p_2 p_5 p_8\right)\right).
\end{aligned}
$$

d.) *The modified script should calculate the true reliability of the system using the expression for the reliability function $h(\boldsymbol{p})$ derived in (c), and also estimate this reliability based on the simulation results. Finally, the script should create a plot of the convergence curve described above and save this in a file in a folder called crudegen.*

Solution:

When running the implemented code (1) using crude Monte Carlo simulations, we obtain the following plot

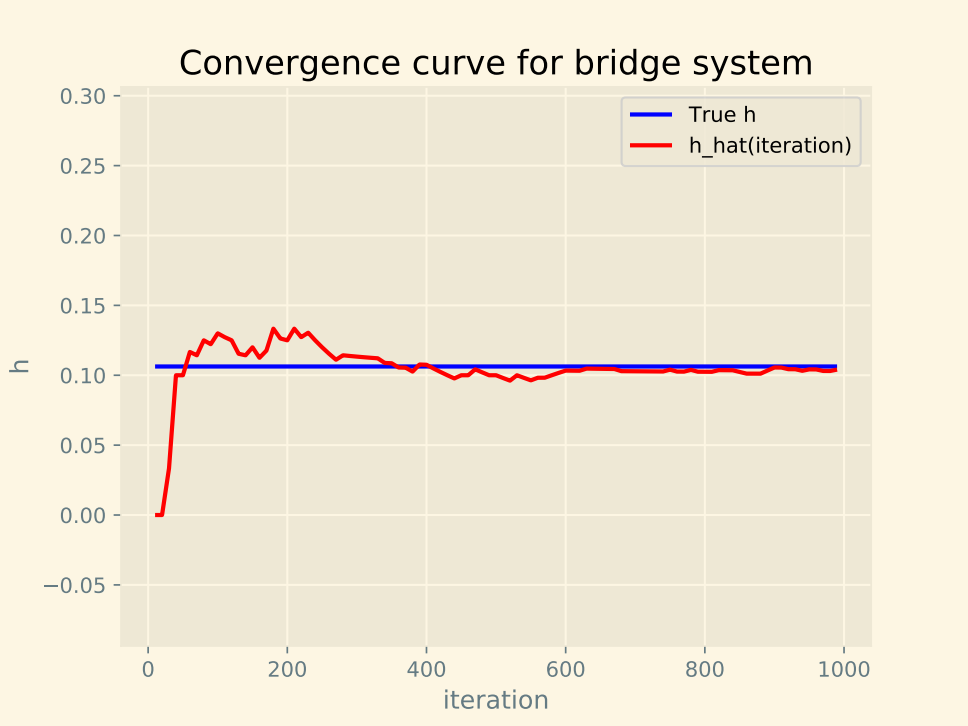

Figure 2: Plot showing the 1000 crude Monte Carlo simulations, and the corresponding estimate it produces (i.e red curve)

with the following estimate

Actual: h	**Estimated**: $\hat{h}$
0.106272	0.103

Table 1: Values of true reliability against the estimated Monte Carlo reliability

e.) *Assuming that* $\hat{\theta}_0, \hat{\theta}_1, \ldots, \hat{\theta}_8$ *are unbiased estimates i.e* $E[\hat{\theta}_s] = \theta_s$, $s = 0, 1, \ldots, 8$ *show that*

$$E(\hat{h}_{CMC}) = h$$

Solution:

By definiton we know that the estimated reliability doing conditional Monte Carlo simulations is

$$\hat{h}_{CMC} = \sum_{s=0}^{8} \hat{\theta}_s \mathbb{P}(S = s)$$

which if we write out the sum gives

$$\hat{h}_{CMC} = \hat{\theta}_0 \mathbb{P}(S = 0) + \hat{\theta}_1 \mathbb{P}(S = 1) + \cdots + \hat{\theta}_8 \mathbb{P}(S = 8).$$

Then we take the expected value of the sum

$$\mathbb{E}\left[\sum_{s=0}^{8} \hat{\theta}_s \mathbb{P}(S = s)\right] = \mathbb{E}\left[\hat{\theta}_0 \mathbb{P}(S = 0)\right] + \mathbb{E}\left[\hat{\theta}_1 \mathbb{P}(S = 1)\right] + \cdots + \mathbb{E}\left[\hat{\theta}_8 \mathbb{P}(S = 8)\right]$$

and apply the fact that $\hat{\theta}_0, \hat{\theta}_1, \ldots, \hat{\theta}_8$ are unbiased estimates

$$\begin{aligned}
\mathbb{E}\left[\sum_{s=0}^{8} \hat{\theta}_s \mathbb{P}(S = s)\right] &= \mathbb{E}\left[\theta_0 \mathbb{P}(S = 0)\right] + \mathbb{E}\left[\theta_1 \mathbb{P}(S = 1)\right] + \cdots + \mathbb{E}\left[\theta_8 \mathbb{P}(S = 8)\right] \\
&= \sum_{s=0}^{8} \theta_s \mathbb{P}(S = s) \\
&= h
\end{aligned}$$

and we are done. ■

f.) *Explain why* $\theta_0 = \theta_1 = \theta_2 = 0$, *while* $\theta_7 = \theta_8 = 1$. *How can this be used to make the conditional simulations more efficient?*

Solution:

When running conditional Monte Carlo simulations we would like to minimise the variance of the estimate we are making as much as possible. This can be done thorugh constructing S in such a way that it contains as much information about ϕ as possible. This means that in the process we only calculate the conditional probabilities we really need, and can only cost us a time complexity of $\mathcal{O}(n)$. Nonetheless, is it possible to improve the time

complexity further when applying specified focus and precision on the estimates that affect the simulations the most.

We recall the results in (a.) found through the minimal path and cut sets, and observe in relation to the Monte Carlo simulations that it is superflous to spend time on estimating any $\hat{\theta}_s,\ s < d\ \vee s > n - c$ since these estimates are more or less deterministic already.

Hence in summary, is it beneficial to let

$$\begin{cases} \hat{\theta}_s = 0, & s < d \\ \hat{\theta}_s = 1, & s > n - c \end{cases}$$

where n is the number of components. This then says that in our example

$$\begin{cases} \hat{\theta}_s = 0, & s < 3 \\ \hat{\theta}_s = 1, & s > 6 \end{cases}$$

which is what we wanted to show. ■

g.) *The modified script should calculate the distribution of the variable S defined above, and estimate the conditional expectations $\theta_3, \ldots, \theta_6$. Moreover, the script should calculate the true reliability of the system using the expression for the reliability function $h(p)$ derived in (c), and also estimate this reliability based on the simulation results. Finally, the script should create a plot of the convergence curve described above and save this in a file in a folder called cmcgen.*

Solution:

The code showcased in (2) produces the follwing plot and results

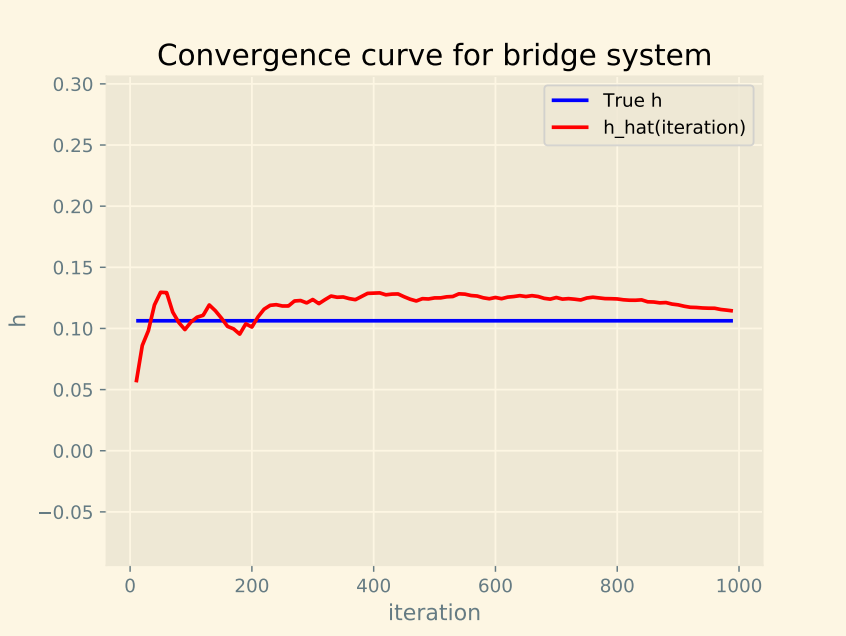

Figure 3: Plot showing the 1000 conditional Monte Carlo simulations, and the corresponding estimate it produces (i.e red curve)

$\mathbb{P}(S = s)$	Probability
$\mathbb{P}(S = 1)$	0.003528
$\mathbb{P}(S = 2)$	0.031248000000000005
$\mathbb{P}(S = 3)$	0.11588200000000001
$\mathbb{P}(S = 4)$	0.23502200000000006
$\mathbb{P}(S = 5)$	0.28492000000000006
$\mathbb{P}(S = 5)$	0.211212
$\mathbb{P}(S = 6)$	0.093402
$\mathbb{P}(S = 7)$	0.022518000000000003
$\mathbb{P}(S = 8)$	0.002268

Table 2: Distribution of probability values for S

θ_s	Estimate
θ_3	0.0
θ_4	0.0
θ_5	0.19047619047619047
θ_6	0.5321100917431193

Table 3: Estimate values of θ_s for $s = 3, 4, 5, 6$

Actual: h	**Estimated**: $\hat{h}$
0.106272	0.11471700393184797

Table 4: Values of true reliability against the estimated Monte Carlo reliability

Where we observe that the estimate for $\hat{h}$ is actually worse for conditioned Monte Carlo simulations (although very slighty). Why this is so might vary, but logically it might have to with the fact that our system is not particulalry large in size (number of components), hence short paths to estimate, whereas the Monte Carlo simulations might not accurately enough capture the conditional relationships.

h.) *By using the result from (c) show that:*

$$\begin{aligned} h(p) = p^2 \cdot \left(p \coprod p\right) \cdot \left(p \coprod p\right) \cdot \left(p \coprod p\right) \\ + 2p(1-p) \cdot \left(p \coprod p\right) \cdot \left(p^2 \coprod p^2\right) \\ + (1-p)^2 \cdot \left(p^3 \coprod p^3\right) \end{aligned}$$

Solution:

From (c.) we had that

$$
\begin{aligned}
h(\boldsymbol{p}) = {} & p_3 p_6 \cdot \left(p_1 \coprod p_2\right) \cdot \left(p_4 \coprod p_5\right) \cdot \left(p_7 \coprod p_8\right) \\
& + p_3(1-p_6) \cdot \left(p_1 \coprod p_2\right) \cdot \left(\left(p_4 p_7 \coprod p_5 p_8\right)\right) \\
& + (1-p_3)(p_6) \cdot \left(\left(p_1 p_4 \coprod p_2 p_5\right)\right) \cdot \left(p_7 \coprod p_8\right) \\
& + (1-p_3)(1-p_6) \cdot \left(\left(p_1 p_4 p_7 \coprod p_2 p_5 p_8\right)\right).
\end{aligned}
$$

where if we plug in that all probabilities are equal gives

$$
\begin{aligned}
h(p) = {} & p \cdot p \cdot \left(p \coprod p\right) \cdot \left(p \coprod p\right) \cdot \left(p \coprod p\right) && (3) \\
& + p(1-p) \cdot \left(p \coprod p\right) \cdot \left(\left(p \cdot p \coprod p \cdot p\right)\right) && (4) \\
& + (1-p)(p) \cdot \left(\left(p \cdot p \coprod p \cdot p\right)\right) \cdot \left(p \coprod p\right) && (5) \\
& + (1-p)(1-p) \cdot \left(\left(p \cdot p \cdot p \coprod p \cdot p \cdot p\right)\right). && (6)
\end{aligned}
$$

which is nothing more than

$$
\begin{aligned}
h(p) = {} & p^2 \cdot \left(p \coprod p\right) \cdot \left(p \coprod p\right) \cdot \left(p \coprod p\right) \\
& + 2p(1-p) \cdot \left(p \coprod p\right) \cdot \left(p^2 \coprod p^2\right) \\
& + (1-p)^2 \cdot \left(p^3 \coprod p^3\right)
\end{aligned}
$$

when observing that (4) ad (5) are the same expression. ■

i.) *The modified script should create a plot containing both the true reliability function, as well as the estimated reliability function based on the simulation results, and save this in a file in a folder called crude.*

Solution:

By code (3) we obtain the followig plot

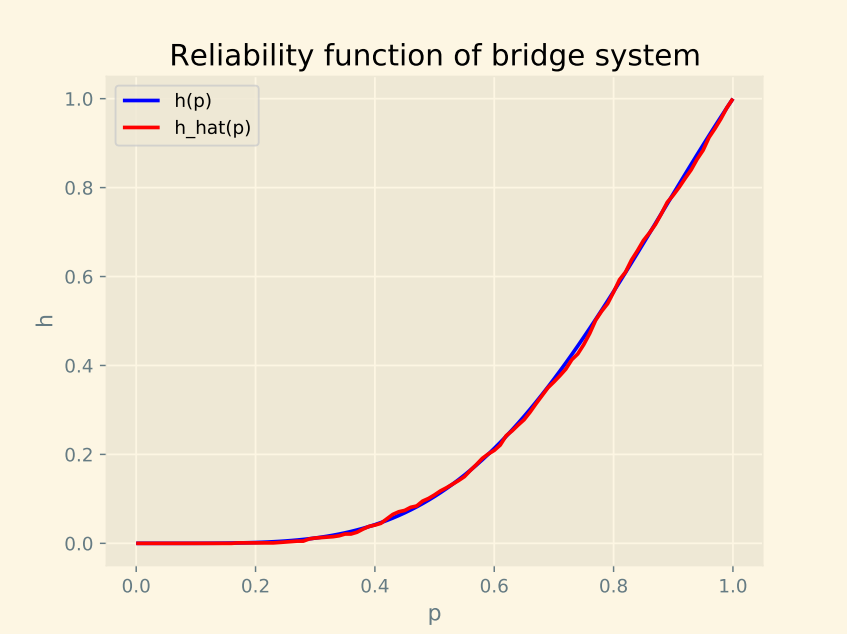

Figure 4: Plot showing the 1000 crude Monte Carlo simulations for equal probabilities, and the corresponding estimate it produces (i.e red curve)

j.) *The modified script should create a plot containing both the true reliability function, as well as the estimated reliability function based on the simulation results, and save this in a file in a folder called cmc.*

Solution:

The script at (4) produces the following plot and result

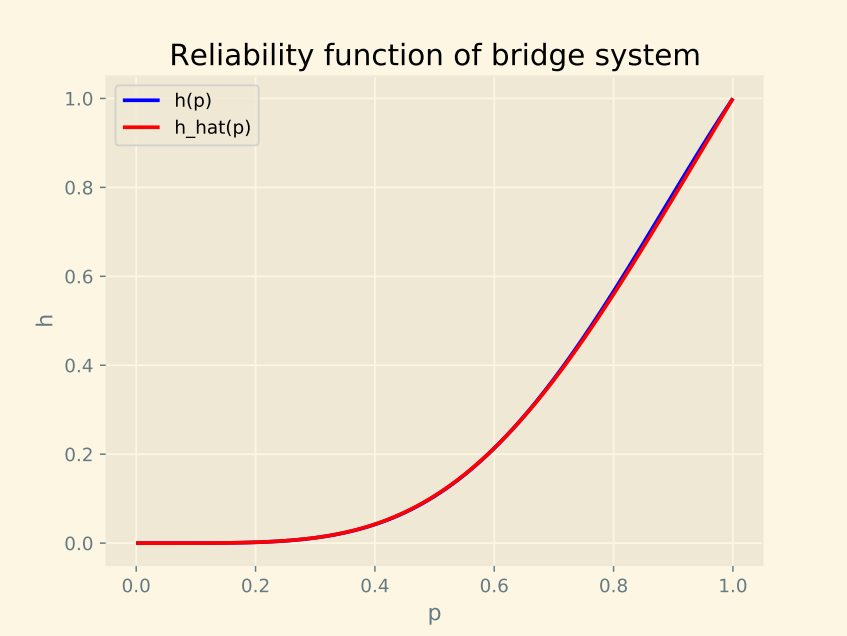

Figure 5: Plot showing the 1000 conditional Monte Carlo simulations for equal probabilities, and the corresponding estimate it produces (i.e red curve)

$\hat{\theta}_s$	**Estimation**
$\hat{\theta}_0$	0
$\hat{\theta}_1$	0
$\hat{\theta}_2$	0
$\hat{\theta}_3$	0
$\hat{\theta}_4$	0
$\hat{\theta}_5$	0.146
$\hat{\theta}_6$	0.438
$\hat{\theta}_7$	0.721
$\hat{\theta}_8$	1

Table 5: Values of true reliability against the estimated Monte Carlo reliability

which by clear observation is an even better and more accurate estimation of h than the crude Monte Carlo method provides (though still only minor differences).

Now seeing as the probabilities are equal the conditional Monte Carlo excels due to its employment of conditional information. Since the probabilities are equal they now become dependent and thus can cause an increase in variance, which conditional Monte Carlo simulations deal with more adeptly than the crude simulations.

Appendix

Source code

Listing 1: Crude Monte Carlo Reliability (exercise d.)

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

plt.style.use("Solarize_Light2")
# Number of simulations
num_sims = 1000

# Interval between h_hat calculations
h_interv = 10

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

# Save plot to file or not?
save_plot = True

# Name of system
sys_name = "bridge"
```

```
# Number of components:
n = 8

# Component reliabilities
px = [
   0.0,
   0.6,
   0.3,
   0.5,
   0.4,
   0.7,
   0.5,
   0.3,
   0.6,
] # P(X_i = px[i]), i = 1, .., n. px[0] is not used


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


def phi(xx):
   system = (
      xx[3]
      * xx[6]
      * coprod(xx[1], xx[2])
      * coprod(xx[4], xx[5])
      * coprod(xx[7], xx[8])
   )

   +((1 - xx[3]) * xx[6] * coprod(xx[1] * xx[4], xx[2] * xx[5]) *
      coprod(xx[7], xx[8]))

   +((1 - xx[6]) * xx[3] * coprod(xx[1], xx[2]) * coprod(xx[4] * xx[7],
      xx[5] * xx[8]))

   +((1 - xx[3]) * (1 - xx[6]) * coprod(xx[1] * xx[4] * xx[7], xx[2] *
      xx[5] * xx[8]))
   return system


def hh(pp):
   rel = (
      pp[3]
      * pp[6]
      * coprod(pp[1], pp[2])
      * coprod(pp[4], pp[5])
      * coprod(pp[7], pp[8])
   )
```

```
   +((1 - pp[3]) * pp[6] * coprod(pp[1] * pp[4], pp[2] * pp[5]) *
      coprod(pp[7], pp[8]))

   +((1 - pp[6]) * pp[3] * coprod(pp[1], pp[2]) * coprod(pp[4] * pp[7],
      pp[5] * pp[8]))

   +((1 - pp[3]) * (1 - pp[6]) * coprod(pp[1] * pp[4] * pp[7], pp[2] *
      pp[5] * pp[8]))
   return rel


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

I = []
H = []
H_hat = []
T = 0

# Calculate the true system reliability
h = hh(px)

for sim in range(num_sims):
   U = gen.uniform(0.0, 1.0, n + 1) # Uniform variables (U[0] is not used)
   for m in range(1, n + 1):
      if U[m] <= px[m]:
         X[m] = 1
      else:
         X[m] = 0
   T += phi(X)
   if sim > 0 and sim % h_interv == 0:
      h_hat = T / sim
      I.append(sim)
      H.append(h)
      H_hat.append(h_hat)

# Estimate final system reliability
h_hat = T / num_sims

print("h = ", h, ", h_hat = ", h_hat)

plt.plot(I, H, "b", label="True h")
plt.plot(I, H_hat, "r", label="h_hat(iteration)")

plt.ylim(h - 0.2, h + 0.2)
plt.xlabel("iteration")
plt.ylabel("h")

plt.title("Convergence curve for " + sys_name + " system")
plt.legend()
```

```
if save_plot:
   plt.savefig("crudegen/" + sys_name + ".pdf")
plt.show()
```

Listing 2: Conditional Monte Carlo Reliability (exercise g.)

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

plt.style.use("Solarize_Light2")

# Number of simulations
num_sims = 1000

# Interval between h_hat calculations
h_interv = 10

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

# Save plot to file or not?
save_plot = True

# Name of system
sys_name = "bridge"

# Number of components:
n = 8

# Length of shortest path
d = 3

# Length of shortest cut
c = 2

# Component reliabilities
px = [
   0,
   0.6,
   0.3,
   0.5,
   0.4,
   0.7,
   0.5,
   0.3,
   0.6,
] # P(X_i = px[i]), i = 1, .., n. px[0] is not used
```

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


def phi(xx):
   system = (
      xx[3]
      * xx[6]
      * coprod(xx[1], xx[2])
      * coprod(xx[4], xx[5])
      * coprod(xx[7], xx[8])
   )

   +((1 - xx[3]) * xx[6] * coprod(xx[1] * xx[4], xx[2] * xx[5]) *
      coprod(xx[7], xx[8]))

   +((1 - xx[6]) * xx[3] * coprod(xx[1], xx[2]) * coprod(xx[4] * xx[7],
      xx[5] * xx[8]))

   +((1 - xx[3]) * (1 - xx[6]) * coprod(xx[1] * xx[4] * xx[7], xx[2] *
      xx[5] * xx[8]))
   return system


def hh(pp):
   rel = (
      pp[3]
      * pp[6]
      * coprod(pp[1], pp[2])
      * coprod(pp[4], pp[5])
      * coprod(pp[7], pp[8])
   )

   +((1 - pp[3]) * pp[6] * coprod(pp[1] * pp[4], pp[2] * pp[5]) *
      coprod(pp[7], pp[8]))

   +((1 - pp[6]) * pp[3] * coprod(pp[1], pp[2]) * coprod(pp[4] * pp[7],
      pp[5] * pp[8]))

   +((1 - pp[3]) * (1 - pp[6]) * coprod(pp[1] * pp[4] * pp[7], pp[2] *
      pp[5] * pp[8]))
   return rel


# Compute the distributions of S_1, ... , S_n, where S_m = X_m + ... + X_n,
   m = 1, ..., n
ps = np.zeros(
   [n + 1, n + 1]
) # P(S_m = s) = ps[m,s], s = 0, 1, ..., (n-m+1). ps[0,s] is not used
```

```
ps[n, 0] = 1.0 - px[n] # P(S_n = 0) = 1 - P(X_n = 1)
ps[n, 1] = px[n] # P(S_n = 1) = P(X_n = 1)

for j in range(1, n):
   m = n - j
   ps[m, 0] = ps[m + 1, 0] * (1.0 - px[m]) # P(S_m = 0) = P(S_{m+1} = 0) *
      P(X_m = 0)

   for s in range(1, n - m + 1):
      ps[m, s] = ps[m + 1, s - 1] * px[m] + ps[m + 1, s] * (
         1.0 - px[m]
      ) # P(S_m = s) = P(S_{m+1} = s-1) * P(X_m = 1)
      # + P(S_{m+1} = s) * P(X_m = 0)

   ps[m, n - m + 1] = (
      ps[m + 1, n - m] * px[m]
   ) # P(S_m = n-m+1) = P(S_{m+1} = n-m) * P(X_m = 1)

# Print the distribution of S = S_1
for s in range(n + 1):
   print("P(S = " + str(s) + ") =", ps[1, s])

# Calculate pdc = P(d <= S_1 <= n-c)
pdc = 0
for s in range(d, n - c + 1):
   pdc += ps[1, s]

# Calculate pcn = P(n-c < S_1 <= n)
pcn = 0
for s in range(n - c + 1, n + 1):
   pcn += ps[1, s]


# Sample S_1 from the set {d, ... , n-c}
def sampleS():
   u = gen.uniform(0.0, pdc)
   for s in range(d, n - c):
      if u < ps[1, s]:
         return s
      else:
         u -= ps[1, s]
   return n - c


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 random path sets of size s = d, 1, ..., n-c
V = np.zeros(n + 1, dtype=int) # V[s] counts random sets of size s = d, 1,
```

```
   ..., n-c

I = []
H = []
H_hat = []

# Calculate the true system reliability
h = hh(px)

# Run the simulations
for sim in range(num_sims):
   s = sampleS()
   V[s] += 1
   U = gen.uniform(0.0, 1.0, n + 1) # Uniform variables (U[0] is not used)
   sumx = 0
   for m in range(1, n):
      if sumx < s:
         p = px[m] * ps[m + 1, s - sumx - 1] / ps[m, s - sumx]
         if U[m] <= p:
            X[m] = 1
         else:
            X[m] = 0
         sumx += X[m]
      else:
         X[m] = 0
   if sumx < s:
      X[n] = 1
   else:
      X[n] = 0
   T[s] += phi(X)
   if sim > 0 and sim % h_interv == 0:
      h_hat = pcn
      for s in range(d, n - c + 1):
         if V[s] > 0:
            h_hat += ps[1, s] * T[s] / V[s]
      I.append(sim)
      H.append(h)
      H_hat.append(h_hat)

# Print the estimated conditional reliabilities, theta_s, s = d, ..., n-c
for s in range(d, n - c + 1):
   print("theta_" + str(s) + " =", T[s] / V[s])

# Estimate final system reliability
h_hat = pcn

for s in range(d, n - c + 1):
   h_hat += ps[1, s] * T[s] / V[s]

print("h = ", h, ", h_hat = ", h_hat)
```

```
plt.plot(I, H, color="blue", label="True h")
plt.plot(I, H_hat, color="red", label="h_hat(iteration)")
plt.ylim(h - 0.2, h + 0.2)
plt.xlabel("iteration")
plt.ylabel("h")
plt.title("Convergence curve for " + sys_name + " system")
plt.legend()
if save_plot:
   plt.savefig("cmcgen/" + sys_name + ".pdf")
plt.show()
```

Listing 3: Crude Monte Carlo Reliability (exercise i.)

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

plt.style.use("Solarize_Light2")
# 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 = 8


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


def phi(xx):
   system = (
      xx[3]
      * xx[6]
      * coprod(xx[1], xx[2])
      * coprod(xx[4], xx[5])
      * coprod(xx[7], xx[8])
   )
```

```
    +((1 - xx[3]) * xx[6] * coprod(xx[1] * xx[4], xx[2] * xx[5]) *
       coprod(xx[7], xx[8]))

    +((1 - xx[6]) * xx[3] * coprod(xx[1], xx[2]) * coprod(xx[4] * xx[7],
       xx[5] * xx[8]))

    +((1 - xx[3]) * (1 - xx[6]) * coprod(xx[1] * xx[4] * xx[7], xx[2] *
       xx[5] * xx[8]))
    return system


def hh(pp):
    rel = pp**2 * coprod(pp, pp) * coprod(pp, pp) * coprod(pp, pp)

    +(2 * pp * (1 - pp) * coprod(pp**2, pp**2) * coprod(pp, pp))

    +((1 - pp) ** 2 * coprod(pp**3, pp**3))
    return rel


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)

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, color="blue", label="h(p)")
plt.plot(p, h_hat, color="red", 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()
```

Listing 4: Conditional Monte Carlo Reliability (exercise j.)

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

plt.style.use("Solarize_Light2")
# 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 = 8


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


def phi(xx):
   system = (
      xx[3]
      * xx[6]
      * coprod(xx[1], xx[2])
      * coprod(xx[4], xx[5])
      * coprod(xx[7], xx[8])
   )

   +((1 - xx[3]) * xx[6] * coprod(xx[1] * xx[4], xx[2] * xx[5]) *
      coprod(xx[7], xx[8]))

   +((1 - xx[6]) * xx[3] * coprod(xx[1], xx[2]) * coprod(xx[4] * xx[7],
      xx[5] * xx[8]))
```

```
   +((1 - xx[3]) * (1 - xx[6]) * coprod(xx[1] * xx[4] * xx[7], xx[2] *
      xx[5] * xx[8]))
   return system


def hh(pp):
   rel = pp**2 * coprod(pp, pp) * coprod(pp, pp) * coprod(pp, pp)

   +(2 * pp * (1 - pp) * coprod(pp**2, pp**2) * coprod(pp, pp))

   +((1 - pp) ** 2 * coprod(pp**3, pp**3))
   return rel


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)

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, color="blue", label="h(p)")
plt.plot(p, h_hat, color="red", 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()
```