STK3405 – Lecture 7

A. B. Huseby

Department of Mathematics
University of Oslo, Norway



Section 4.4.

k-out-of-*n* systems



k-out-of-n systems

RECALL: A k-out-of-n system is a binary monotone system (C, ϕ) where $C = \{1, \ldots, n\}$ which is functioning if and only if at least k out of the n components are functioning.

Thus, ϕ can be written as:

$$\phi(\boldsymbol{x}) = I(\sum_{i=1}^{n} x_i \geq k) \tag{1}$$

In order to evaluate the reliability of a k-out-of-n system it is convenient to introduce the following random variable:

$$S = \sum_{i=1}^{n} X_i.$$

Thus, S is the number of functioning components. This implies that:

$$h = P(\phi(\boldsymbol{X}) = 1) = P(S \geq k).$$



The distribution of S

We now explain how to find the distribution of S in the general case where $P(X_i = 1) = p_i$, for $i \in C$, and where the p_is may be unequal.

We also let $q_i = 1 - p_i$, for $i \in C$, and introduce the partial sums:

$$S_j = \sum_{i=1}^{j} X_i, \quad j = 1, \ldots, n.$$

NOTE: $S_1 = X_1$ while $S_n = S$.

The distribution of S is found by computing sequentially the distributions of $S_1, \ldots, S_n$.



The distribution of S (cont.)

Since $S_1 = X_1$, it follows that $P(S_1 = 0) = q_1$ and that $P(S_1 = 1) = p_1$. This determines the distribution of S_1 completely.

Assume that the distributions of $S_1, \ldots, S_j$ where $j < n$ have been determined, and consider the distribution of S_{j+1}.

By conditioning on X_{j+1} we get the following formula:

$$\begin{aligned} P(S_{j+1} = s) &= P(S_{j+1} = s | X_{j+1} = 0) \cdot P(X_{j+1} = 0) \\ &\quad + P(S_{j+1} = s | X_{j+1} = 1) \cdot P(X_{j+1} = 1) \\ &= P(S_j = s) \cdot q_{j+1} + P(S_j = s - 1) \cdot p_{j+1} \end{aligned}$$

for $s = 0, 1, \ldots, j + 1$.



The distribution of S (cont.)

NOTE: If $s = 0$, then $P(S_j = s - 1) = 0$, while if $s = j + 1$, then $P(S_j = s) = 0$.

Thus, in these cases the formula simplifies to:

$$P(S_{j+1} = 0) = P(S_j = 0) \cdot q_{j+1}$$

$$P(S_{j+1} = j + 1) = P(S_j = j) \cdot p_{j+1}$$

Given the distribution of S_j we need to do $2(j + 1)$ multiplications and j additions to find the distribution of S_{j+1}.

Repeating this process for $j = 1, \ldots, n - 1$, the distribution of $S = S_n$ is found.

The total number of multiplications is:

$$2 \cdot 2 + 2 \cdot 3 + \cdots 2 \cdot n = (n - 1)(n + 2),$$

while the number of additions is:

$$1 + 2 + \cdots + (n - 1) = \frac{(n - 1)n}{2}$$



The reliability of a k-out-of-n system

Having found the distribution of S the reliability of the system is given by:

$$P(\phi = 1) = P(S \geq k) = \sum_{s=k}^{n} P(S = s)$$

The number of additions need to calculate this last expression is $n - k$.

The total number of algebraic operations (multiplications and additions) grows approximately proportional to n^2.

CONCLUSION: The order of this algorithm is $O(n^2)$.

Since n^2 is a *polynomial* in n, this algorithm is a *polynomial time* algorithm, which means that we can easily handle very large systems of this type.



Example

Example

Let (C, ϕ) be a 6-out-of-8 system where the components have the following reliabilities:

$$p_1 = 0.75, \quad p_2 = 0.80, \quad p_3 = 0.82, \quad p_4 = 0.65$$
$$p_5 = 0.88, \quad p_6 = 0.91, \quad p_7 = 0.92, \quad p_8 = 0.86$$

We also let $S = X_1 + \cdots + X_8$. Using the derived formula we get that:

$$P(S = 6) = 0.2797, \quad P(S = 7) = 0.3701, \quad P(S = 8) = 0.2026$$

Hence, the reliability of the system is:

$$h = P(\phi = 1) = P(S \geq 6) = 0.2797 + 0.3701 + 0.2026 = 0.8524$$



Example (cont.)

```
#####################################################################
#                                                                   #
#  This program calculates the reliability of a k-out-of-n system   #
#  by finding the distribution of:                                  #
#                                                                   #
#     S = X_{1} + ... + X_{n}.                                      #
#                                                                   #
#####################################################################

import numpy as np

# Component reliabilities (p[0] is not in use)
p = [0.0, 0.75, 0.80, 0.82, 0.65, 0.88, 0.91, 0.92, 0.86]

# Threshold
b = 6

# Number of components
n = len(p) - 1


# After each iteration ps[s] = P(X_{1} + ... + X_{i+1} = s)
ps = np.zeros(n + 1)

# Temporary storage of ps[s]
ph = np.zeros(n + 1)

# Calculate ps[0] = P(X_{1} = 0) and ps[1] = P(X_{1} = 1):
ps[0] = 1 - p[1]
ps[1] = p[1]

# Save these values for the next iteration
ph[0] = ps[0]
ph[1] = ps[1]
```



Example (cont.)

```
# Save these values for the next iteration
ph[0] = ps[0]
ph[1] = ps[1]


for j in range(1, n):
    # Calculate P(X_{1} + ... + X_{j+1} = 0):
    ps[0] = ph[0] * (1 - p[j+1])

    # Calculate P(X_{1} + ... + X_{j+1} = s), s = 1, ..., j:
    for s in range(1, j+1):
        ps[s] = ph[s] * (1 - p[j+1]) + ph[s-1] * p[j+1]

    # Calculate P(X_{1} + ... + X_{j+1} = j+1):
    ps[j+1] = ph[j] * p[j+1]

    # Save these values for the next iteration
    for s in range(j+2):
        ph[s] = ps[s]


# At this stage ps[s] = P(X_{1} + ... + X_{n} = s), s = 0, 1, ..., n

for s in range(n+1):
    print("P(S = ", s, ") = ", ps[s])

# Calculate the system reliability h = P(S = b) + ... + P(S = n)
h = 0

for s in range(b, n+1):
    h += ps[s]

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



Example (cont.)

```
P(S =  0 ) =  3.810239999999997e-07
P(S =  1 ) =  1.8479663999999987e-05
P(S =  2 ) =  0.0003766519199999998
P(S =  3 ) =  0.004206380991999999
P(S =  4 ) =  0.028107056319999997
P(S =  5 ) =  0.11486580081600001
P(S =  6 ) =  0.279747929296
P(S =  7 ) =  0.37005429136000006
P(S =  8 ) =  0.2026230286080002
h =  0.8524252492640001
```



Section 4.5.

Threshold systems



Threshold systems

Definition

A *threshold system* is a binary monotone system (C, ϕ), where the structure function has the following form:

$$\phi(\boldsymbol{x}) = I(\sum_{i=1}^{n} a_i x_i \geq b), \tag{2}$$

where $a_1, \ldots, a_n$ and b are non-negative real numbers, and $I(\cdot)$ denotes the indicator function.

The parameters $a_1, \ldots, a_n$ are referred to as the *weights* of the system, while b is referred to as the *threshold*.

NOTE: If $a_i = 0$ for some $i \in C$, this implies that component i is irrelevant. Thus, in a coherent threshold system, all the weights must at least be *positive*.



Threshold systems (cont.)

Proposition (Rational weights)

Let (C, ϕ) be a threshold system with weights $a_1, \ldots, a_n$ and threshold b. Then there exists alternative weights $a_1^*, \ldots, a_n^* \in \mathbb{Q}$ such that:

$$\phi(\boldsymbol{x}) = I(\sum_{i=1}^{n} a_i^* x_i \geq b)$$

PROOF: Let $\mathcal{X}_0 = \{\boldsymbol{x} : \phi(\boldsymbol{x}) = 0\}$ and $\mathcal{X}_1 = \{\boldsymbol{x} : \phi(\boldsymbol{x}) = 1\}$. Thus, it follows that:

$$\sum_{i=1}^{n} a_i x_i < b, \text{ for all } \boldsymbol{x} \in \mathcal{X}_0$$

$$\sum_{i=1}^{n} a_i x_i \geq b, \text{ for all } \boldsymbol{x} \in \mathcal{X}_1$$



Threshold systems (cont.)

We then let c be a constant defined as follows:

$$c = \sup\{\sum_{i=1}^{n} a_i x_i : \boldsymbol{x} \in \mathcal{X}_0\}.$$

Since $\mathcal{X}_0$ is a *finite set*, there exists a vector $\boldsymbol{x}_0 = (x_{1,0}, \ldots, x_{n,0}) \in \mathcal{X}_0$ such that:

$$\sum_{i=1}^{n} a_i x_{i,0} = c$$

Since $\sum_{i=1}^{n} a_i x_i < b$ for all $\boldsymbol{x} \in \mathcal{X}_0$, it follows that $c < b$.

We then let:

$$\epsilon = \frac{b - c}{2n} > 0.$$



Threshold systems (cont.)

We may now choose alternative weights $a_1^*, \ldots, a_n^*$ such that:

$$a_i^* \in [a_i, a_i + \epsilon], \quad i = 1, \ldots, n,$$

If $\boldsymbol{x} \in \mathcal{X}_0$, we have:

$$\sum_{i=1}^{n} a_i^* x_i \leq \sum_{i=1}^{n} a_i x_i + n\epsilon \leq c + \frac{b-c}{2} = \frac{b+c}{2} < b.$$

On the other hand, if $\boldsymbol{x} \in \mathcal{X}_1$, we have:

$$\sum_{i=1}^{n} a_i^* x_i \geq \sum_{i=1}^{n} a_i x_i \geq b$$

Thus, we have:

$$\phi(\boldsymbol{x}) = I(\sum_{i=1}^{n} a_i x_i \geq b) = I(\sum_{i=1}^{n} a_i^* x_i \geq b) \quad \text{for all } \boldsymbol{x} \in \mathcal{X}_0 \cup \mathcal{X}_1 = \{0,1\}^n.$$



Threshold systems (cont.)

Finally, since the set $\mathbb{Q}$ is *dense*, the intervals $[a_i, a_i + \epsilon]$ contain infinitely many rational numbers.

Hence, we may choose the alternative weights so that $a_1^*, \ldots, a_n^* \in \mathbb{Q}$, as claimed.

Proposition (Integer weights)

Let (C, ϕ) be a threshold system with weights $a_1, \ldots, a_n$ and threshold b. Then there exists alternative weights $a_1', \ldots, a_n' \in \mathbb{N}$ and threshold b' such that:

$$\phi(\boldsymbol{x}) = I(\sum_{i=1}^{n} a_i' x_i \geq b')$$



Threshold systems (cont.)

PROOF: We have already shown that we may replace the weights $a_1, \ldots, a_n$ by rational weights $a_1^*, \ldots, a_n^*$ without changing the system, i.e.:

$$\phi(\boldsymbol{x}) = I(\sum_{i=1}^{n} a_i x_i \geq b) = I(\sum_{i=1}^{n} a_i^* x_i \geq b) \quad \text{for all } \boldsymbol{x} \in \{0,1\}^n.$$

We then let $w > 0$ be the least common denominator of the weights $a_1^*, \ldots, a_n^*$, and get:

$$\phi(\boldsymbol{x}) = I(\sum_{i=1}^{n} a_i^* x_i \geq b) = I(\sum_{i=1}^{n} w \cdot a_i^* x_i \geq w \cdot b) = I(\sum_{i=1}^{n} a_i' x_i \geq b'),$$

where $a_i' = (w \cdot a_i^*)$, $i = 1, \ldots, n$, and $b' = (w \cdot b)$.

From this it follows that $a_1', \ldots, a_n' \in \mathbb{N}$ as claimed.



Threshold systems (cont.)

NOTE: This implies that the class of threshold systems with integer weights is equal to the class of all threshold systems.

In cases where all the weights are integers with a *common divisor D*, this divisor can be factored out by dividing all weights as well as the threshold by D.

It turns out that having *integer weights* and at the same time making these weights as *small* as possible can reduce the computational complexity significantly.

NOTE: If (C, ϕ) is a threshold system where $a_1 = \cdots = a_n = 1$ and $b = k$, (C, ϕ) is a k-out-of-n system.

Despite the similarity between threshold systems and k-out-of-n systems, calculating the reliability of a threshold system in general is NP-hard.

By placing mild restrictions on the weights, it is possible to calculate the reliability of the system in polynomial time.



Threshold systems (cont.)

Let (C, ϕ) be a threshold system, where $a_1, \ldots, a_n$ are positive integers, and let $\boldsymbol{X} = (X_1, \ldots, X_n)$ be the vector of component state variables. $X_1, \ldots, X_n$ are assumed to be independent.

We let $P(X_i = 1) = p_i$, and $q_i = 1 - p_i$, for all $i \in C$, and introduce:

$$S = \sum_{i=1}^{n} a_i X_i,$$

NOTE: The system is functioning if and only if $S \geq b$. Thus, the reliability of the system can be derived from the distribution of S.



Threshold systems (cont.)

To determine this distribution we again introduce the integer valued partial sums:

$$S_j = \sum_{i=1}^{j} a_i X_i, \quad j = 1, 2, \ldots, n.$$

Thus, in particular: $S_1 = a_1 X_1$ while $S_n = S$.

It is also convenient to introduce:

$$d_j = \sum_{i=1}^{j} a_i, \quad j = 1, 2, \ldots, n.$$

The set of possible values for S_j is denoted $\mathcal{A}_j$, $j = 1, \ldots, n$. Hence:

$$\mathcal{A}_j \subseteq \{0, 1, \ldots, d_j\}, \quad j = 1, \ldots, n. \tag{3}$$

More specifically:

$$\begin{aligned} \mathcal{A}_1 &= \{0, a_1\}, \\ \mathcal{A}_j &= \mathcal{A}_{j-1} \cup \{s + a_j : s \in \mathcal{A}_{j-1}\}, \quad j = 2, \ldots, n. \end{aligned} \tag{4}$$



A simple example

Assume that $n = 4$, $a_1 = 1$, $a_2 = 2$, $a_3 = 3$ and $a_4 = 4$.

Then we have:

$$d_1 = 1, \quad , d_2 = 3, \quad d_3 = 6, \quad d_4 = 10.$$

Hence, we get:

$$\begin{aligned}
\mathcal{A}_1 &= \{0, 1\} \\
\mathcal{A}_2 &= \mathcal{A}_1 \cup \{s + a_2 : s \in \{0, 1\}\} \\
&= \{0, 1, 2, 3\} \\
\mathcal{A}_3 &= \mathcal{A}_2 \cup \{s + a_3 : s \in \{0, 1, 2, 3\}\} \\
&= \{0, 1, 2, 3, 4, 5, 6\} \\
\mathcal{A}_4 &= \mathcal{A}_3 \cup \{s + a_4 : s \in \{0, 1, \ldots, 6\}\} \\
&= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}
\end{aligned}$$



Threshold systems (cont.)

The distribution of S is determined by computing sequentially the distributions of $S_1, \ldots, S_n$.

Since $S_1 = a_1 X_1$, it follows that $P(S_1 = 0) = q_1$ and that $P(S_1 = a_1) = p_1$.

We then assume that we have determined the distributions of $S_1, \ldots, S_j$ where $j < n$, and consider the distribution of S_{j+1}. By conditioning on X_{j+1} we get for $s \in \mathcal{A}_{j+1}$:

$$P(S_{j+1} = s) = P(S_j = s) \cdot q_{j+1} + P(S_j = s - a_{j+1}) \cdot p_{j+1}. \tag{5}$$

Note that if $s < a_{j+1}$, we have $P(S_j = s - a_{j+1}) = 0$, while if $s > d_j$, we have $P(S_j = s) = 0$. Thus, in these cases (5) simplifies to:

$$P(S_{j+1} = s) = P(S_j = s) \cdot q_{j+1}, \quad s < a_{j+1}, \tag{6}$$

$$P(S_{j+1} = s) = P(S_j = s - a_{j+1}) \cdot p_{j+1}, \quad s > d_j. \tag{7}$$



Threshold systems (cont.)

To calculate the distribution of S_{j+1}:

(5) is applied at most $\max(0, d_j - a_{j+1} + 1)$ times

(6) is applied at most a_{j+1} times

(7) is applied at most $d_{j+1} - d_j = a_{j+1}$ times

When (5) is applied, we need to do two multiplications and one addition, while (6) and (7) only a single multiplication is required.

In the worst case we need to do $2 \cdot \max(0, d_j - a_{j+1} + 1) + 2 \cdot a_{j+1}$ multiplications and $\max(0, d_j - a_{j+1} + 1)$ additions.



Threshold systems (cont.)

In the case where $a_1 = \cdots = a_n = 1$, we have $d_j = j$, $j = 1, \ldots, n$.

Thus, in this case we need to do $2 \cdot \max(0, j - 1 + 1) + 2 \cdot 1 = 2(j + 1)$ multiplications and $\max(0, j - 1 + 1) = j$ additions.

Unfortunately, in other cases the number of operations may grow much faster.



Example

Let (C, ϕ) be a threshold system with weights $a_1, \ldots, a_n$ given by:

$$a_j = 2^{j-1}, \quad j = 1, \ldots, n.$$

We then have:

$$\begin{aligned}
\mathcal{A}_1 &= \{0, 2^0\} = \{0, 1\}, \\
\mathcal{A}_2 &= \mathcal{A}_1 \cup \{0 + 2^1, 2^0 + 2^1\} = \{0, 1, 2, 3\} \\
&\ldots \\
\mathcal{A}_j &= \{0, 1, \ldots, (2^j - 1)\} \\
&\ldots \\
\mathcal{A}_n &= \{0, 1, \ldots, (2^n - 1)\}
\end{aligned}$$

Hence, $|\mathcal{A}_j| = 2^j$, $j = 1, \ldots, n$, which means that the size of the $\mathcal{A}_j$ grows exponentially fast. In this case any algorithm for calculating the distribution of $S = S_n$ must be at least of order $O(2^n)$.



Example (cont.)

We note that the number of possible values for S is equal to the number of binary vectors $\boldsymbol{x} = (x_1, \ldots, x_n)$, i.e., 2^n. In fact each possible value s of S corresponds to a unique vector $\boldsymbol{x}(s) = (x_1(s), \ldots, x_n(s))$ where the entries represents the digits in the binary representation of the number s. That is:

$$s = \sum_{i=1}^{n} x_i(s) 2^{i-1}$$

Hence, it follows that the distribution of S can be determined by:

$$P(S = s) = \prod_{i=1}^{n} P(X_i = x_i(s)) = \prod_{i=1}^{n} p_i^{x_i(s)} q_i^{1-x_i(s)}, \quad s = 0, 1, \ldots, 2^n - 1.$$

However, calculating the full distribution of S using this simple explicit formula still requires the calculation of 2^n such products.



Example (cont.)

It should be noted, however, that for a given threshold value b, it is possible to compute the reliability of the system without calculating the distribution of S.

Assume e.g., that $b = 2^{k-1}$ for some integer $k \in \{1, \ldots, n\}$. In this case all sets of the form $P = \{j\}$, where $j \geq k$ will be minimal path sets for the system, while all components in the set $\{1, \ldots, k-1\}$ are irrelevant.

Thus, the system is essentially a simple parallel system of the components $k, \ldots, n$, and the reliability of the system is given by:

$$P(\phi = 1) = \coprod_{j=k}^{n} p_j$$



Threshold systems (cont.)

We see that when the reliability of a threshold system is calculated using the distribution of S, the running time of the algorithm may in the worst cases grows exponentially in the number of components.

However, by putting a mild restriction on the weights, it becomes possible to calculate the reliability of a threshold system in polynomial time.



Example

Let (C, ϕ) be a threshold system with positive integer weights $a_1, \ldots, a_n$, and assume that:

$$a_j \leq M, \quad j = 1, \ldots, n,$$

for some suitable fixed integer $M > 0$. In this case we have:

$$d_j = \sum_{i=1}^{j} a_i \leq Mj$$

Hence, since a_{j+1} is assumed to be a positive integer, we also have that:

$$d_j - a_{j+1} + 1 \leq Mj$$

Calculating the distribution of S_{j+1} from the distribution of S_j, an upper bound on the number of multiplications needed is:

$$2\max(0, d_j - a_{j+1} + 1) + 2a_{j+1} \leq 2Mj + 2M = 2M(j+1)$$

Similarly, an upper bound on the number of addition needed is:

$$\max(0, d_j - a_{j+1} + 1) \leq Mj$$



Example (cont.)

An upper bound on the total number of multiplications needed to calculate the distribution of S is then:

$$2 \cdot 2M + 2 \cdot 3M + \cdots 2 \cdot nM = M(n-1)(n+2),$$

while an upper bound on the number of additions needed is:

$$M + 2M + \cdots + (n-1)M = \frac{M(n-1)n}{2}$$

Since M is a fixed constant, it follows that the distribution of S, and hence also the reliability of the system for any threshold b, can be calculated in $O(n^2)$-time.



Example 1

Let (C, ϕ) be a threshold system of 8 components with the following reliabilities:

$$p_1 = 0.75, \quad p_2 = 0.80, \quad p_3 = 0.82, \quad p_4 = 0.65$$
$$p_5 = 0.88, \quad p_6 = 0.91, \quad p_7 = 0.92, \quad p_8 = 0.86$$

The component weights are:

$$a_1 = 2, \quad a_2 = 3, \quad a_3 = 5, \quad a_4 = 6$$
$$a_5 = 7, \quad a_6 = 8, \quad a_7 = 8, \quad a_8 = 11$$

and the threshold is $b = 32$.

We also let $S = a_1 X_1 + \cdots + a_8 X_8$, and note that since $a_1 + \cdots + a_8 = 50$, the set of possible values for S is a subset of the set $\{0, 1, 2, \ldots, 50\}$.



Example 1 (cont.)

Using (5), (6) and (7) we find the distribution of S, and get the reliability of the system:

$$\begin{aligned} h &= P(\phi = 1) = P(S \geq b) \\ &= P(S = 32) + P(S = 33) + \cdots + P(S = 50) = 0.9238 \end{aligned}$$



Example 1 (cont.)

```
#####################################################################
#                                                                   #
#  This program calculates the reliability of a threshold system    #
#  by finding the distribution of:                                  #
#                                                                   #
#     S = a_{1} X_{1} + ... + a_{n} X_{n}.                          #
#                                                                   #
#####################################################################

import numpy as np

# Component reliabilities (p[0] is not in use)
p = [0.0, 0.75, 0.80, 0.82, 0.65, 0.88, 0.91, 0.92, 0.86]

# Component weights (a[0] is not in use)
a = [0, 2, 3, 5, 6, 7, 8, 8, 11]

# Threshold
b = 32

# Number of components
n = len(p) - 1

# Cumulative weights (d[0] is not in use)
d = np.zeros(n+1, dtype=int)

for j in range(1, n + 1):
    d[j] = a[j] + d[j-1]

# After each iteration ps[s] = a_{1} X_{1} + ... + a_{n} X_{n} = s)
ps = np.zeros(d[n] + 1)

# Temporary storage of ps[s]
ph = np.zeros(d[n] + 1)
```



Example 1 (cont.)

```
# Calculate ps[0] = P(X_{1} = 0) and ps[1] = P(X_{1} = 1):
ps[0] = 1 - p[1]
ps[a[1]] = p[1]

# Save these values for the next iteration
ph[0] = ps[0]
ph[a[1]] = ps[a[1]]


for j in range(1, n):
    # Calculate P(a_{1} X_{1} + ... + a_{j} X_{j+1} = s), s = 0, 1, ... , (a[j+1]-1)
    for s in range(a[j+1]):
        ps[s] = ph[s] * (1 - p[j+1])

    # Calculate P(a_{1} X_{1} + ... + a_{j} X_{j+1} = s), s = a[j+1], ..., d[j]:
    for s in range(a[j+1], d[j] + 1):
        ps[s] = ph[s] * (1 - p[j+1]) + ph[s-a[j+1]] * p[j+1]

    # Calculate P(a_{1} X_{1} + ... + a_{j} X_{j+1} = s), s = d[j]+1 , ... , d[j+1]:
    for s in range(d[j] + 1, d[j+1] + 1):
        ps[s] = ph[s-a[j+1]] * p[j+1]

    # Save these values for the next iteration
    for s in range(d[j+1] + 1):
        ph[s] = ps[s]
```



Example 1 (cont.)

```
# At this stage ps[s] = P(a_{1} X_{1} + ... + a_{j} X_{j+1} = s), s = 0, 1, ... , d[n]

for s in range(d[n] + 1):
    print("P(S = ", s, ") = ", ps[s])

# Calculate the system reliability h = P(S = b) + ... + P(S = d[n])
h = 0

# for s in range(b, d[n] + 1):
#     h += ps[s]

for s in range(d[n] + 1):
    if s >= b:
        h += ps[s]


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



Example 2

```
P(S =  32 ) =  0.013970805888000003
P(S =  33 ) =  0.025047925488
P(S =  34 ) =  0.026273245423999995
P(S =  35 ) =  0.014366469936
P(S =  36 ) =  0.036533418432
P(S =  37 ) =  0.045261104095999996
P(S =  38 ) =  0.008367747024000002
P(S =  39 ) =  0.07544177555199999
P(S =  40 ) =  0.023167133391999998
P(S =  41 ) =  0.036486314591999996
P(S =  42 ) =  0.085146826336
P(S =  43 ) =  0.04245648825600001
P(S =  44 ) =  0.10910470771200002
P(S =  45 ) =  0.06136347817600001
P(S =  46 ) =  0.0
P(S =  47 ) =  0.050655757152
P(S =  48 ) =  0.06754100953600001
P(S =  49 ) =  0.0
P(S =  50 ) =  0.20262302860800002
h =  0.9238072356
```



Example 2 (cont.)

```
#####################################################################
#                                                                   #
#  This program calculates the reliability of a threshold system    #
#  by finding the distribution of:                                  #
#                                                                   #
#     S = (a_{1} X_{1} + ... + a_{n} X_{n}) / gcf.                  #
#                                                                   #
#  where gcf denotes the greatest common factor of the              #
#  weights a_{1}, ..., a_{n}.                                       #
#                                                                   #
#####################################################################

import numpy as np

# The following flag determines if the weights should be divided
# by their greatest common factor or not.
DIVIDE_BY_GCF = False


# Find the greatest common factor of a set of integers using
# the Euclidean algorithm.
# See:  https://en.wikipedia.org/wiki/Greatest_common_divisor
def greatest_common_factor(x):
    if len(x) == 1:
        for i in x:
            return i
    m = np.inf
    for i in x:
        if i < m and i > 0:
            m = i
    y = {m}
    for i in x:
        if i > m:
            y.add(i-m)
    return greatest_common_factor(y)
```



Example 2 (cont.)

```
# Component reliabilities (p[0] is not in use)
p = [0.0, 0.75, 0.80, 0.82, 0.65, 0.88, 0.91, 0.92, 0.86]

# Component weights (a[0] is not in use)
a = [0, 26, 39, 65, 78, 91, 104, 104, 143]

# Threshold
b = 410

gcf = 1

if DIVIDE_BY_GCF:
    gcf = greatest_common_factor(set(a))
    for i in range(len(a)):
        a[i] = int(a[i] / gcf)

# Number of components
n = len(p) - 1

# Cumulative weights (d[0] is not in use)
d = np.zeros(n+1, dtype=int)

for j in range(1, n + 1):
    d[j] = a[j] + d[j-1]

# After each iteration ps[s] = a_{1} X_{1} + ... + a_{n} X_{n} = s)
ps = np.zeros(d[n] + 1)

# Temporary storage of ps[s]
ph = np.zeros(d[n] + 1)
```



Example 2 (cont.)

```
# Calculate ps[0] = P(X_{1} = 0) and ps[1] = P(X_{1} = 1):
ps[0] = 1 - p[1]
ps[a[1]] = p[1]

# Save these values for the next iteration
ph[0] = ps[0]
ph[a[1]] = ps[a[1]]


for j in range(1, n):
    # Calculate P(a_{1} X_{1} + ... + a_{j} X_{j+1} = s), s = 0, 1, ... , (a[j+1]-1)
    for s in range(a[j+1]):
        ps[s] = ph[s] * (1 - p[j+1])

    # Calculate P(a_{1} X_{1} + ... + a_{j} X_{j+1} = s), s = a[j+1], ..., d[j]:
    for s in range(a[j+1], d[j] + 1):
        ps[s] = ph[s] * (1 - p[j+1]) + ph[s-a[j+1]] * p[j+1]

    # Calculate P(a_{1} X_{1} + ... + a_{j} X_{j+1} = s), s = d[j]+1 , ... , d[j+1]:
    for s in range(d[j] + 1, d[j+1] + 1):
        ps[s] = ph[s-a[j+1]] * p[j+1]

    # Save these values for the next iteration
    for s in range(d[j+1] + 1):
        ph[s] = ps[s]
```



Example 2 (cont.)

```
# At this stage ps[s] = P(a_{1} X_{1} + ... + a_{j} X_{j+1} = s), s = 0, 1, ... , d[n]

for s in range(d[n] + 1):
    print("P(S = ", s, ") = ", ps[s])

# Calculate the system reliability h = P(S = b) + ... + P(S = d[n])
h = 0

# for s in range(b, d[n] + 1):
#     h += ps[s]

for s in range(d[n] + 1):
    if gcf * s >= b:
        h += ps[s]

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

