



Fault Tolerant Computing

CS 530

Reliability Analysis Pt 2

Yashwant K. Malaiya

Colorado State University

Reliability Analysis: Continuation

- Note that a parallel configuration is an ideal redundant system. It assumes that a correctly operating unit can always be identified.
- Here we examine **Triple-Modular Redundancy**, a realistic scheme that has been used for both hardware and software. It is a special case of the combinatorial **k-out-of-n** system.
- We also examine use of switching to switch out a bad unit.

k-out-of-n Systems

- Assumption: we have n identical modules with statistically independent failures.
- **k-out-of-n system** is operational if k of the n modules are good.
- System reliability then is

$$R_{k/n} = \sum_{i=k}^n \binom{n}{i} p^i (1-p)^{n-i}$$

Where p is the probability that one unit

is good. $R_{k/n}$ is the summations of the probabilities of all good combinations.

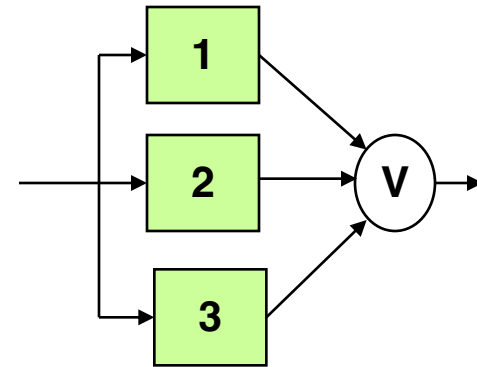
Note that you can choose i good system out of n in $\{n \text{ choose } i\}$ (i.e. nCi) ways.

Also recall that $nCi = n!/[i!(n-i)!]$

nCi is also written like the equation on the left.

Triple Modular Redundancy

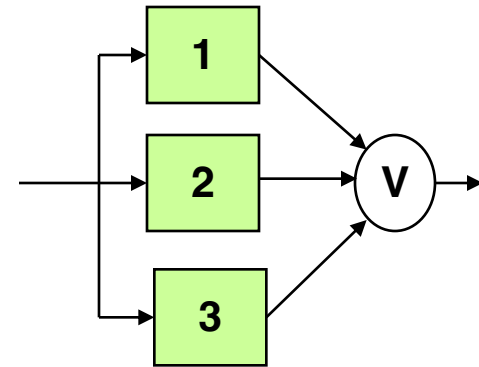
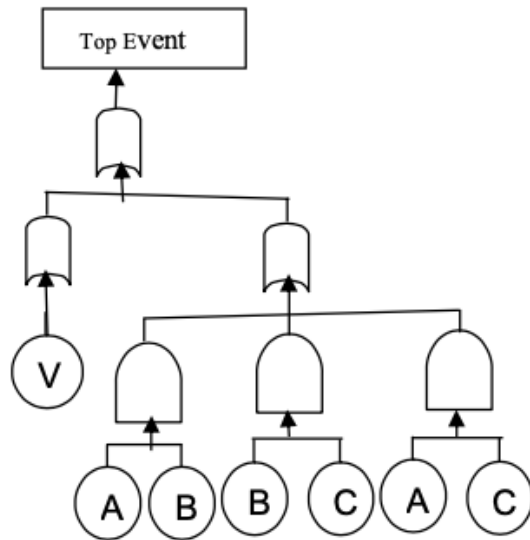
- Popular high-reliability scheme: 2-out-of-3 system
- Output is obtained using a majority voter



$$\begin{aligned} R_{TMR} &= \sum_{i=2}^3 \binom{3}{i} R^i (1-R)^{3-i} \\ &= 3R^2(1-R) + R^3 \\ &= 3R^2 - 2R^3 \end{aligned}$$

Where R is the reliability of a single module. This assumes that the voter is perfect, a reasonable assumption if the voter complexity is much less than an individual module.

TMR: Fault Tree Analysis



The system fails if

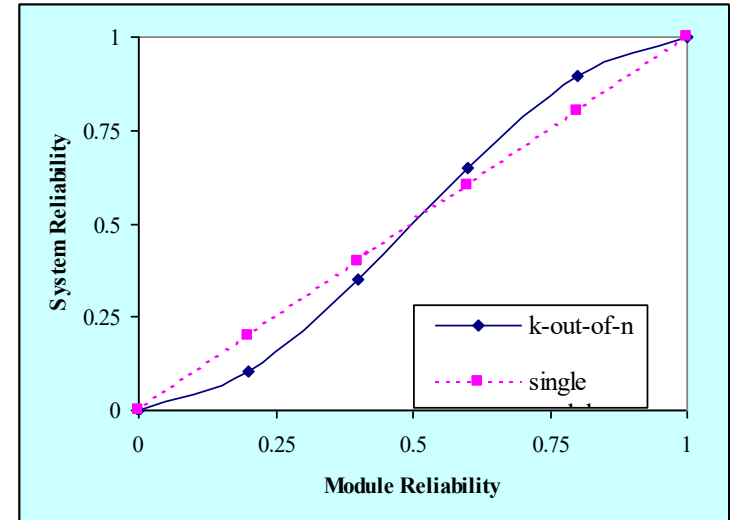
$$(\text{voter fails}) \cup (A f \cap B f) \cup (B f \cap C f) \cup (A f \cap C f)$$

Triple Modular Redundancy

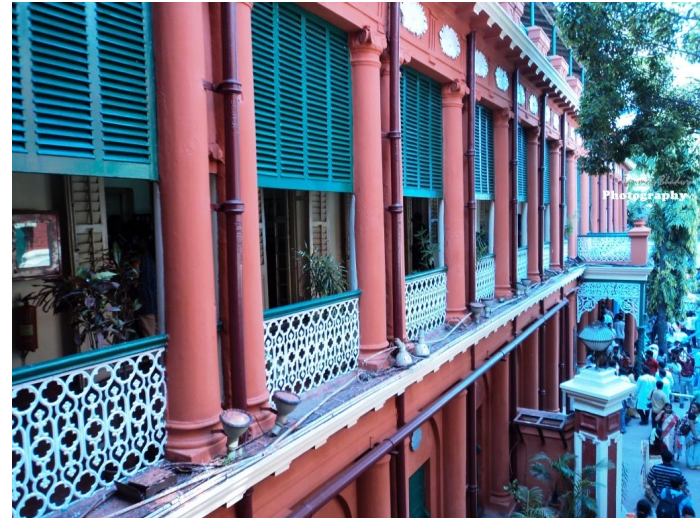
Here is a plot of the system reliability, when the individual module reliability varies between 0 to 1.

Note that if the reliability of an individual module is less than 0.5, it is more likely to be bad. Having several such modules, and taking majority vote, will actually make the system less reliable, as you see in the figure.

A political application of the principle: majority-based democracy works only if the individuals are more likely to make the right decision than wrong!



The Story of the Tagore Family



From Barnes and Noble Coffee Shop

Tagore: Nobel prize 1913: Gitanjali poems

TMR: Permanent Failures

Let $R = e^{-\lambda t}$

$$R_{TMR}(t) = 3e^{-2\lambda t} - 2e^{-3\lambda t}$$

$$MTTF = \int_0^{\infty} R_{TMR}(t) dt$$

$$= \int_0^{\infty} (3e^{-2\lambda t} - 2e^{-3\lambda t}) dt$$

$$= \frac{5}{6\lambda} \quad \left(\text{single module MTTF} : \frac{1}{\lambda} \right)$$

Thus TMR has a lower MTTF than a single module!

- When is a single module as reliable as TMR?

Solving $3R^2 - 2R^3 = R$

we get $R_{\text{cross}} = 0.5$.

TMR worse after $R < 0.5$!

MTTF may not be a good measure when very high reliability levels are maintained.

TMR

- Mission time

$$R_{Th} = 3e^{-2\lambda t_m} - 2e^{-3\lambda t_m}$$

A numerical solution for t_m can be obtained iteratively.

- Ex : $\lambda = 1/\text{year}$, $R_{Th} = 0.95$

	<i>MTTF</i>	t_m
single	1yr	0.05
TMR	0.83	0.145

Thus TMR mission time is much better.

- Temporary faults : steady state

$$A_{TMR} = 3A^2 - 2A^3, A = \frac{\mu}{\lambda + \mu}$$

- Ex : $\frac{\lambda}{\mu} = 0.01 \Rightarrow A = 0.9901$

$$\Rightarrow \bar{A} = 0.01$$

$$A_{TMR} = 0.9997 \Rightarrow \bar{A}_{TMR} = 0.0003$$

Thus TMR can greatly reduce down-time in presence of temporary faults.

TMR: implementation issues

- **Hardware:** 3 identical processors running either
 - Synchronized with each other
 - Not synchronized. Voter has to wait until comparable results from all 3 are available.
- **Software:** 3 independently implementations of the same specifications
 - The hope is that the failures are statistically independent. In practice, there is some correlation. We will examine the impact later.

Switching and “Coverage”

- If a module is suspected of being bad, it can be switched out, and the process is moved to a good module.
- Switching successfully requires
 - Identifying that the module is bad
 - By have some self-test capability in it
 - Or by comparing its output with that of an identical module
 - Successfully restarting the process on another module
 - “Coverage” is the probability that these two will be done successfully. (This coverage has nothing to do with test coverage. I wish they had used another term for this probability).

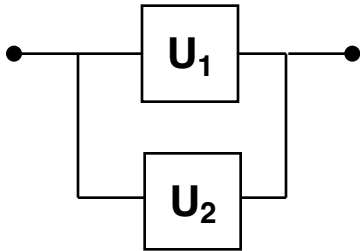
Course Notes

- **PSA1: Due Tues 2/27/24**
 - Understand, analyze, approach, solve problems
 - See canvas discussion group
 - No collaboration is permitted
- **Midterm: Wed, 3/6/24**
 - In the class-room, must have respondus lockdown browser installed and tested
 - Special seating requirements

Primary and Backup Units

- This popular scheme is an example of a **parallel configuration**.
 - There is a **primary** unit that participates in the process.
 - The **backup (spare)** is used if the primary has failed.
 - When primary fails, the uncorrupted **process** needs to be initiated on the backup.
 - Sometimes the backup runs a redundant **shadow process**, so that it is ready when the primary fails.
 - This is sometimes called “**dynamic redundancy**”. It is used when a brief interruption is acceptable.
- Next, we analyze such a system. U_1 is primary and U_2 is secondary, with reliabilities R_1 and R_2 .

Reliability of Primary/Backup system with imperfect “Coverage”

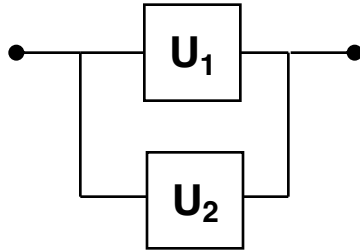


$$\begin{aligned} R_s &= P\{U_1 \text{ good}\} + \\ &\quad P\{U_2 \text{ has taken over} \mid U_1 \text{ failed}\}P\{U_1 \text{ failed}\} \\ &= R_1 + R_2 C(1 - R_1) \end{aligned}$$

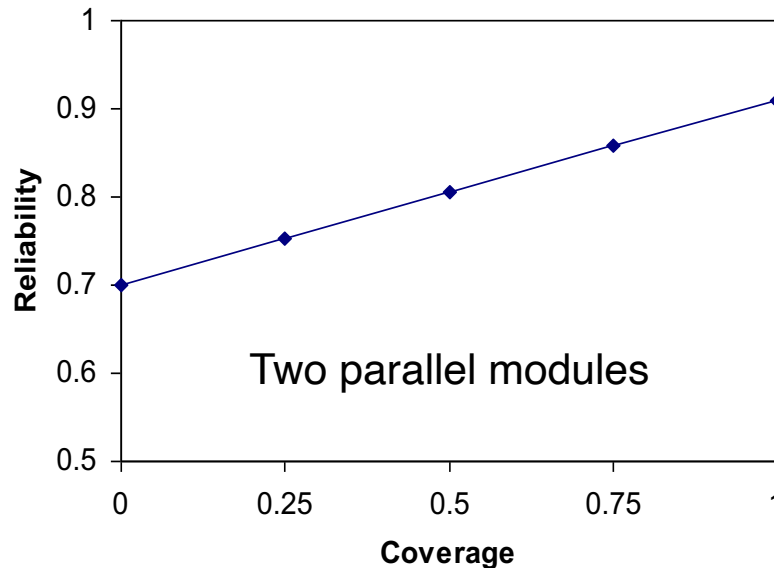
where $C = P\{\text{failure detected and successful switchover}\}$

- Failure detection: requires concurrent detection. This needs some kind of redundancy.
- Switchover requires:
 - good state loaded in U_2 .
 - Process restarted.

Imperfect Coverage: Example



- $R_s = R_1 + R_2 C(1 - R_1)$
- Assuming $R_1 = R_2 = 0.7$
- In general $R_s = R_m \sum_{i=0}^{n-1} C^i (1 - R_m)^i$



Note that better coverage improves the reliability. When coverage =1, full potential of the parallel configuration is achieved.

Disaster Recovery Metrics

- Disaster Recovery: recovery using a backup system
 - Backup can be “hot” or “cold”
- Recovery Time Objective (RTO): maximum acceptable length of time the system can be down.
 - How ready is the backup unit (Hot/Cold), and how fast the unit can switch.
 - Ex: Critical systems about 1 hour, non-critical 24 hours
- Recovery Point Objectives (RPO): age of the data you can afford to lose.
 - Depends on back-up update frequency.
 - Ex: few seconds to a few hours

Using TMR + Spares

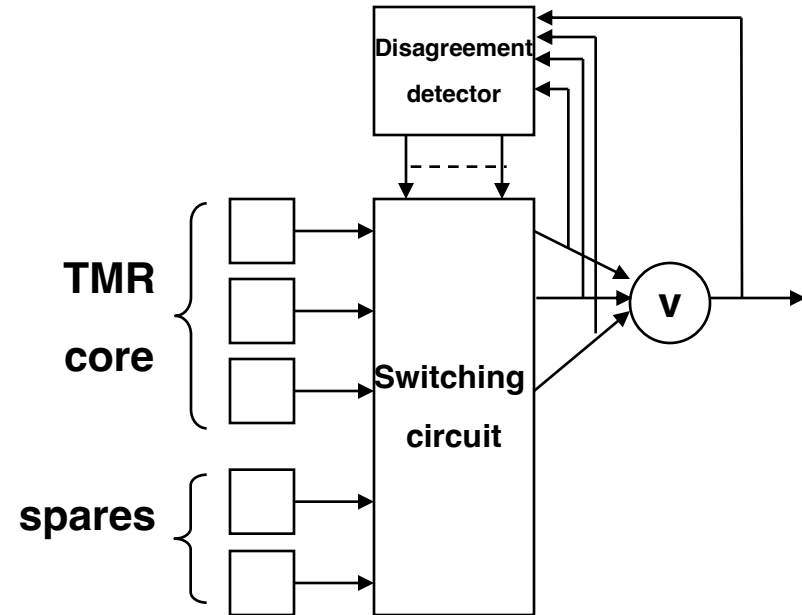
- A TMR will mask a single error, it will be “contained” by the voter. Thus it can be called a “**static redundancy**”. Used when no interruptions are desired.
- You can combine static and dynamic redundancy (“hybrid”) by having some **spares** in addition to the **TMR core** with three voting modules.
- A permanent failure in the core can be fixed by switching in a spare.
- The system will work as long as we have two good modules in the TMR core.

TMR+Spares

- TMR core, n-3 spares (assume same failure rate)
- **Scheme A:** System failure when all but one modules have failed. If we start with 3 in the core and 2 spares, the sequence will be
 $3+2 \rightarrow 3+1 \rightarrow 3+0 \rightarrow 2+0 \rightarrow \text{failure}$
- Reliability of the system then is

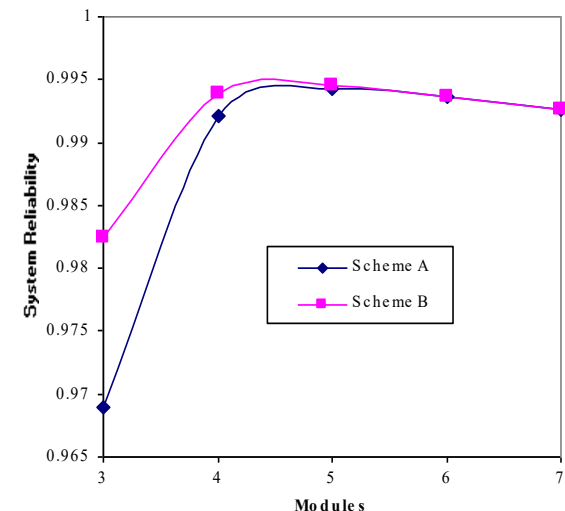
$$R_s = R_{sw} [1 - nR(1-R)^{n-1} - (1-R)^n]$$

Where R is reliability of a single module and R_{sw} is the reliability of the switching circuit overhead.
- R_{sw} should depend on total number of modules n, and relative complexity of the switching logic.
- Let us assume that $R_{sw} = (R^a)^n$, where a is measure of relative complexity, generally $a \ll 1$. Then
- $R_s = R^{an} [1 - nR(1-R)^{n-1} - (1-R)^n]$



Example: TMR+Spares

- Ex: $R=0.9$, $a=10^{-2}$. How many modules will give us the best reliability?
- $R_s = R^n [1 - nR(1-R)^{n-1} - (1-R)^n]$
- The plot of scheme A show that the optimal $n_{\max} \approx 4$, i.e. one spare in addition of a TMR core.
- Consider now **Scheme B**. When we have only two good modules left in the core, and if one of them fails, we still have one good module, but we don't know which one. If we arbitrarily choose one of them to continue, we will have a 50% chance of making the correct choice!



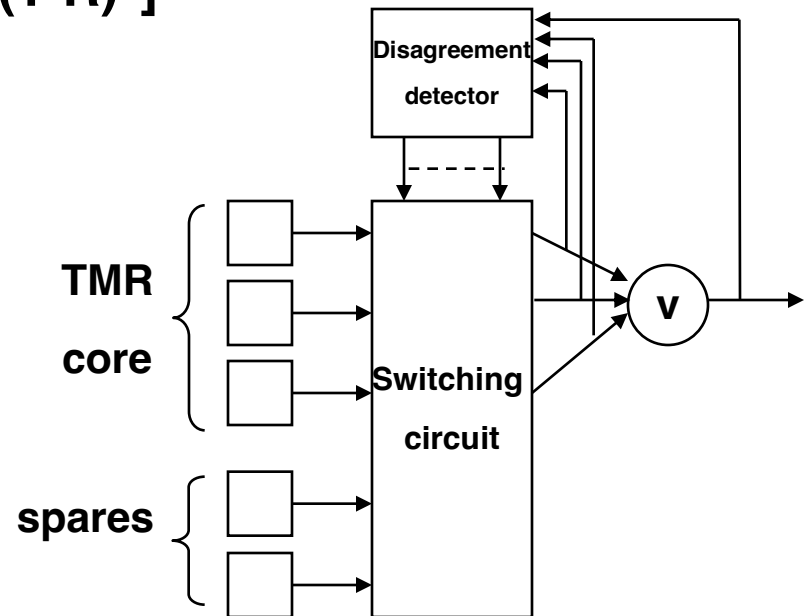
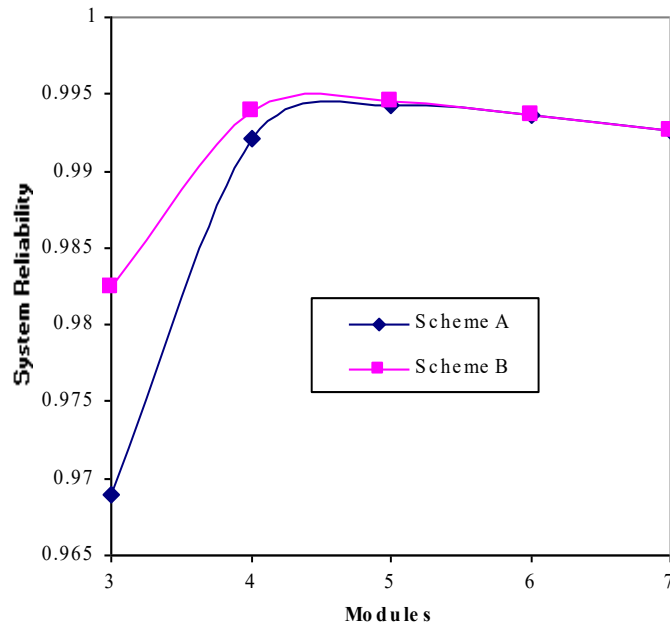
Scheme B is risky, but sometimes risk taking is needed for survival.

TMR+Spare: Scheme B

B: One of the last two fails, remove one arbitrarily.

3+2 → 3+1 → 3+0 → 2+0 → 1+0 → failure

$$R_s = R^n [1 - 0.5nR(1-R)^{n-1} - (1-R)^n]$$

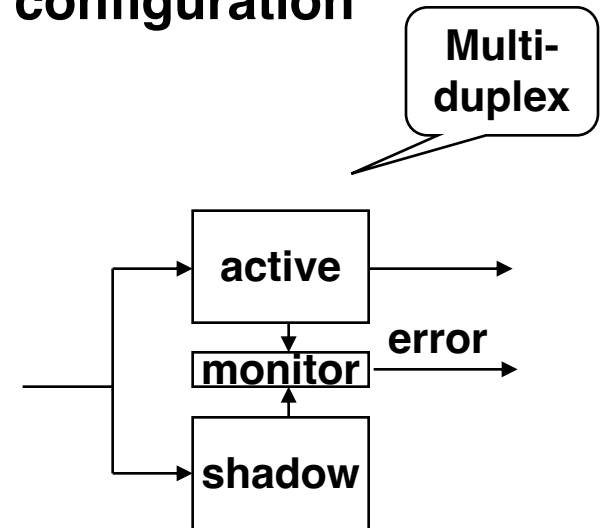
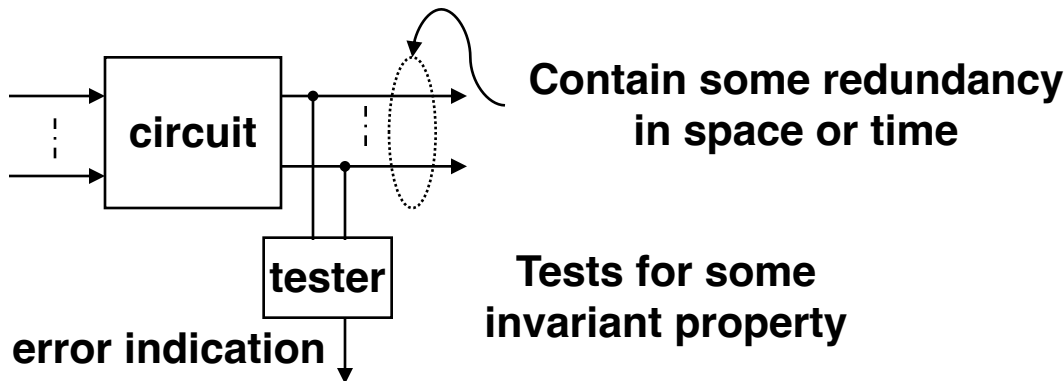


Redundancy for Testing and Fault-tolerance

- **Duplication for fault-tolerance:**
 - Primary and backup
- **Duplication for**
 - Self-Testing
- **Coding for self-testing**
 - Using an invariant property

On-line Testing

- On-line fault detection:
- Periodic scheduled testing
- Concurrent testing:
 - a. self-testing logic
 - b. duplex configuration



- Only selected nodes need to be compared.
- Active and Shadow need to stay synchronized

Variable Failure Rates

- We have assumed that the failure rate is constant. It greatly simplifies the calculations.
- It results in exponentially distributed time to failure.
- When the failure rate is varying, a generalization of the exponential distribution called Weibull distribution is used.
- The next slide is for your information only. For more details, consult the literature.

Generalization: Failure Rate

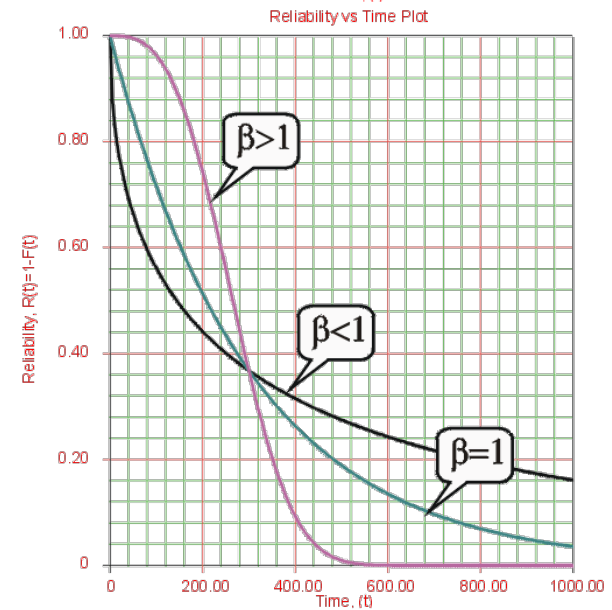
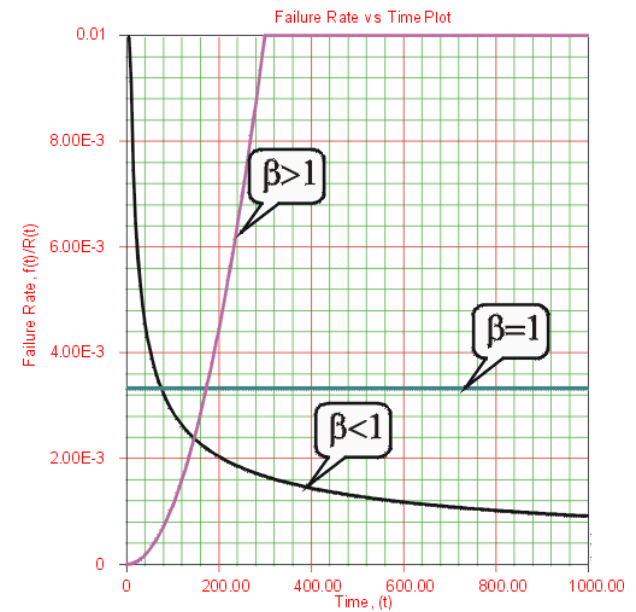
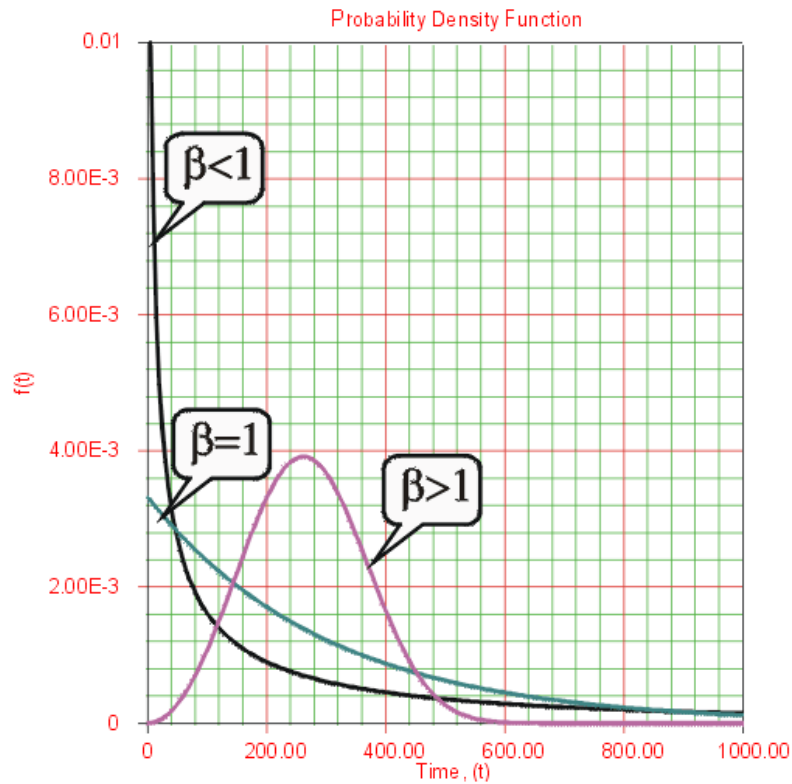
- $z(t) = \frac{\beta}{\eta} \left(\frac{t}{\eta}\right)^{\beta-1}$ where $\beta \geq 0, \eta \geq 0$

leads to Weibull Distribution

$$f_T(t) = \frac{\beta}{\eta} \left(\frac{t}{\eta}\right)^{\beta-1} e^{-\left(\frac{t}{\eta}\right)^\beta} \quad R(t) = e^{-\left(\frac{t}{\eta}\right)^\beta}$$

- β : shape parameter, 1 for constant rate
- η : scale parameter
- Often used to get a better fit, if needed,
for rising or falling failure rates.

Weibull Distribution



Q: Simple Voter Design

- Output is equal to majority
- Design this simple voter as a combinational circuit

I0	I1	I2	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	1	1	1

- Vote is more complex if the outputs are generated at slightly different times.