

Reliability Analysis with R: A Companion Guide

Paul Govan

2026-06-08

Table of contents

Preface	8
How to Use This Book and the Tutorials Together	8
The ReliaLearnR Ecosystem	8
Installation	9
Chapter Overview	9
About the Author	10
1 Reliability, Availability, and Maintainability (RAM)	11
1.1 Introduction	11
1.2 Learning Objectives	11
1.3 What is RAM?	11
1.4 Reliability	12
1.5 Availability	13
1.6 Mean Time to Repair (MTTR)	13
1.7 Mean Time to Failure (MTTF)	14
1.8 Mean Time Between Failures (MTBF)	14
1.9 Failure Rate	15
1.10 Probability of Failure	16
1.10.1 The Reliability Curve	16
1.11 B_n or L_n Life	18
1.12 From Exponential to Weibull	19
1.13 The Bathtub Curve	20
1.14 System Reliability	22
1.14.1 Series Systems	23
1.14.2 Parallel Systems	23
1.15 Summary	25
2 Reliability Block Diagrams	26
2.1 Introduction	26
2.2 Learning Objectives	26
2.3 What is a Reliability Block Diagram?	26
2.4 Series Systems	27
2.4.1 Example: Water Pumping System	27
2.5 Parallel Systems	28
2.5.1 Example: Backup Power System	28

2.6	Mixed Systems	31
2.6.1	Example: Safety System	31
2.7	k-out-of-n Systems	32
2.7.1	Example: Flight Control Computers	32
2.8	System MTTF	34
2.8.1	Example: Two Pumps in Parallel	34
2.9	Common Cause Failure	35
2.9.1	The Beta-Factor Model	35
2.10	Introduction to Fault Tree Analysis	36
2.10.1	Gates	37
2.10.2	RBD—Fault Tree Duality	37
2.11	Case Study: Industrial Cooling System	39
2.12	Summary	40
3	Life Data Analysis	41
3.1	Introduction	41
3.2	Learning Objectives	41
3.3	Life Distributions	41
3.3.1	The Weibull Distribution	41
3.3.2	The Reliability (Survival) Function	42
3.4	WeibullR	43
3.4.1	Getting Started	43
3.4.2	Fitting a Weibull Model	44
3.4.3	Reading the Probability Plot	44
3.5	Data Censoring	45
3.5.1	Right-Censored Data (Suspensions)	46
3.5.2	Interval-Censored Data	47
3.5.3	Grouped Data	48
3.6	Parameter Estimation Methods	49
3.6.1	MRR vs MLE	49
3.7	Goodness of Fit	50
3.8	Choosing a Distribution	51
3.9	Sample Size Considerations	53
3.10	Other Weibull Models	55
3.10.1	The Weibayes Model	55
3.10.2	The 3-Parameter Weibull	56
3.11	Multi-Plots	57
3.12	Competing Failure Modes	58
3.13	Contour Plots	59
3.13.1	Comparing Contour Plots	60
3.14	Summary	61

4	Reliability Growth Analysis	62
4.1	Introduction	62
4.2	Learning Objectives	62
4.3	Reliability Growth Analysis	62
4.4	Test Planning	63
4.4.1	Reliability Demonstration Tests: <code>rdt()</code>	63
4.5	The Duane Model	65
4.5.1	Fitting with <code>ReliaGrowR</code>	66
4.6	The Crow-AMSAA Model	68
4.6.1	Fitting with <code>ReliaGrowR</code>	70
4.6.2	The Piecewise NHPP Model	72
4.6.3	Change Point Detection	73
4.7	Forecasting Growth: <code>predict_rga()</code>	74
4.8	Summary	76
5	Accelerated Life Testing	77
5.1	Introduction	77
5.2	Learning Objectives	77
5.3	The Acceleration Factor	77
5.4	The Arrhenius Model	78
5.4.1	<code>WeibullR.ALT</code> : Arrhenius-Lognormal Example	79
5.5	The Power Law Model	80
5.5.1	<code>WeibullR.ALT</code> : Power Law Example	81
5.6	Multiple Stress Factors	82
5.7	Parallelization	82
5.8	Relationship Plots	84
5.9	Important Limitations of ALT	86
5.10	Summary	87
6	Repairable Systems Analysis	88
6.1	Introduction	88
6.2	Learning Objectives	88
6.3	Repairable vs Non-Repairable Systems	88
6.4	Failure Processes: HPP and NHPP	89
6.4.1	Homogeneous Poisson Process (HPP)	89
6.4.2	Non-Homogeneous Poisson Process (NHPP)	89
6.5	The Power Law Process	91
6.6	Exposure Analysis	92
6.7	Mean Cumulative Function	93
6.8	NHPP Analysis with <code>nhpp()</code>	96
6.9	Piecewise NHPP	98
6.10	Interpreting Results and Predicting Future Failures	99
6.10.1	Reading $\hat{\beta}$	99

6.10.2	Cumulative vs Instantaneous MTBF	99
6.10.3	Predicting Future Failures	99
6.11	Summary	101
7	Interactive Plots with ReliaPlotR	102
7.1	Introduction	102
7.2	Installation	103
7.3	Life Data Analysis Plots	103
7.3.1	Weibull Probability Plot: <code>plotly_wblr()</code>	103
7.3.2	Contour Plot: <code>plotly_contour()</code>	104
7.4	Reliability Growth Plots	105
7.4.1	Duane Plot: <code>plotly_duane()</code>	105
7.4.2	Crow-AMSAA Plot: <code>plotly_rga()</code>	105
7.5	Accelerated Life Testing Plots	105
7.5.1	ALT Probability Plot: <code>plotly_alt()</code>	106
7.5.2	Life-Stress Relationship Plot: <code>plotly_rel()</code>	106
7.5.3	Power Law–Weibull ALT Plots	106
7.6	Repairable Systems Plots	107
7.6.1	Exposure Plot: <code>plotly_exposure()</code>	107
7.6.2	MCF Plot: <code>plotly_mcf()</code>	108
7.6.3	NHPP Plot: <code>plotly_nhpp()</code>	108
7.7	Embedding in Reports and Dashboards	108
7.8	When to Use ReliaPlotR	109
7.9	Getting Help	109
7.10	Summary	110
8	Web-Based Analysis with ReliaShiny	111
8.1	Introduction	111
8.2	Installation and Launch	111
8.3	App Overview	111
8.4	Module Walkthroughs	113
8.4.1	Life Data Analysis Module	113
8.4.2	Reliability Growth Module	114
8.4.3	Accelerated Life Testing Module	115
8.4.4	Repairable Systems Module	116
8.5	Data Upload Tips	117
8.6	Downloading Results	117
8.7	When to Use ReliaShiny vs Scripted R	118
8.8	Getting Help	118
8.9	Summary	118
9	Reproducible Reliability Analysis with Quarto	119
9.1	Learning Objectives	119

9.2	Why Reproducibility Matters	119
9.2.1	Audits and Regulatory Compliance	119
9.2.2	Re-Analysis and Collaboration	120
9.2.3	Version Control	120
9.3	Introduction to Quarto	120
9.3.1	Installing Quarto	121
9.3.2	Anatomy of a .qmd File	121
9.3.3	Rendering	121
9.4	A Basic Reliability Report in Quarto	122
9.4.1	YAML Front Matter for Reliability Reports	122
9.4.2	Embedding Equations	122
9.4.3	Code Chunk Options	123
9.4.4	A Complete Weibull Report Block	123
9.5	Parameterized Reports	125
9.5.1	What Are Parameters?	125
9.5.2	Declaring Parameters in YAML	125
9.5.3	Using Parameters in Code	125
9.5.4	Rendering Multiple Reports	125
9.6	Integration with Reliability R Packages	126
9.6.1	WeibullR: Life Data Analysis	126
9.6.2	ReliaGrowR: Reliability Growth	127
9.6.3	ReliaLearnR: RAM Metrics	128
9.6.4	Saving Plots	129
9.7	This Book Is Built with Quarto	130
9.8	Quarto Books and Multi-File Projects	130
9.9	When to Use Quarto vs. ReliaShiny vs. AI/MCP	131
9.10	Getting Help	131
9.11	Summary	131
9.12	Review Questions	132
10	AI-Assisted Reliability Analysis with MCP	134
10.1	Introduction	134
10.2	Prerequisites	134
10.3	MCP Tools for ReliaGrowR	135
10.4	MCP Tools for ReliaPlotR	139
10.5	Launching the MCP Server	142
10.6	Connecting to Claude Desktop	143
10.7	Connecting to Claude Code	143
10.8	Example AI Conversations	143
10.9	When to Use AI vs Scripted R	144
10.10	Getting Help	145
10.11	Summary	145

Appendix: R Quick Reference	146
ReliaLearnR	146
WeibullR	148
WeibullR.ALT	148
ReliaGrowR	149
DiagrammeR	150
ReliaPlotR	151
ReliaShiny	152
Glossary and Formula Reference	154
Key Symbols	154
Key Formulas	156
RAM (Chapter 1)	156
Weibull Distribution (Chapter 3)	156
Reliability Block Diagrams (Chapter 2)	157
Accelerated Life Testing (Chapter 5)	157
Reliability Growth Analysis (Chapter 4)	158
Repairable Systems (Chapter 6)	158

Preface

This book is a companion reference guide to the **ReliaLearnR** interactive learning modules. Where the tutorials offer hands-on practice with immediate feedback and interactive widgets, this book provides readable, searchable, printable chapters that cover the same material in a narrative format, plus two additional chapters on the wider ReliaLearnR ecosystem.

How to Use This Book and the Tutorials Together

The interactive tutorials and this book cover the same core content but serve different purposes:

Feature	Interactive Tutorials	This Book
Instant code feedback	Yes	No
Quiz questions	Yes	Review questions
Interactive sliders	Yes	Static equivalents
Searchable / printable	No	Yes
Copy-paste code	Limited	Full
Ecosystem chapters (ReliaPlotR, ReliaShiny)	No	Yes

Recommended workflow: Read a chapter here first to understand the concepts and see the code, then open the corresponding tutorial to practice. The chapter order mirrors the tutorial progression.

The ReliaLearnR Ecosystem

ReliaLearnR is one part of a broader suite of R packages for reliability analysis:

Package	Role
ReliaLearnR	Interactive learning modules
WeibullR	Weibull life data analysis
ReliaGrowR	Reliability growth and repairable systems

Package	Role
WeibullR.ALT	Accelerated life testing
ReliaPlotR	Interactive reliability plots
ReliaShiny	Web-based reliability analysis app
Quarto	Reproducible document and book publishing system

Installation

Install the core packages needed to run the code in this book:

```
install.packages(c(
  "ReliaLearnR",
  "WeibullR",
  "ReliaGrowR",
  "WeibullR.ALT",
  "DiagrammeR",
  "ReliaPlotR",
  "ReliaShiny",
  "quarto"
))
```

Chapter Overview

1. **RAM:** Foundational reliability, availability, and maintainability metrics.
2. **Reliability Block Diagrams:** Series, parallel, mixed, k-out-of-n systems, and fault tree analysis.
3. **Life Data Analysis:** Weibull distribution, censoring, MRR vs. MLE, and contour plots.
4. **Reliability Growth Analysis:** Duane and Crow-AMSAA growth models, test planning, and forecasting.
5. **Accelerated Life Testing:** Arrhenius and Power Law acceleration models for compressed test time.
6. **Repairable Systems:** Power Law Process, MCF, and fleet-level analysis.
7. **Interactive Plots with ReliaPlotR:** Publishable interactive reliability charts.
8. **Web-Based Analysis with ReliaShiny:** Point-and-click reliability analysis.
9. **Reproducible Reports with Quarto:** Literate programming, parameterized reports, and version-controlled reliability analysis.
10. **AI-Assisted Analysis with MCP:** Conversational reliability analysis using Model Context Protocol.

The book also includes an **Appendix** (helper function reference) and a **Glossary** (key symbols and formulas).

About the Author

Paul Govan is the author of ReliaLearnR, ReliaGrowR, ReliaPlotR, and ReliaShiny. He can be reached at paul.govan2@gmail.com.

1 Reliability, Availability, and Maintainability (RAM)

1.1 Introduction

Reliability, Availability, and Maintainability (RAM) are the foundational metrics of reliability engineering. They describe how well a system performs its intended function, how often it is usable, and how quickly it can be restored when it fails. This chapter introduces these concepts and builds toward calculating probability of failure, B_n life, and system-level reliability.

1.2 Learning Objectives

By the end of this chapter, you will be able to:

- Define key reliability metrics: reliability, availability, and failure rate.
- Describe the significance of MTTR, MTTF, and MTBF.
- Calculate probability of failure using reliability data.
- Interpret B_n (or L_n) life values.
- Explain the bathtub curve and identify which failure region a component occupies given its β value.
- Compute system reliability for series and parallel configurations.

1.3 What is RAM?

Reliability: The ability of an item to perform its intended function without failure over a specified period.

Availability: The proportion of time an item is in a functioning condition.

Maintainability: The ease and speed with which an item can be restored to operational status after a failure.

These three concepts are interrelated: a highly reliable item will be available more often, and an item that is easy to maintain can be restored to service quickly, further improving availability.

1.4 Reliability

Reliability is the probability that an item will not fail under defined conditions for a specified period:

$$\text{Reliability} = \left(1 - \frac{\text{Failed Time}}{\text{Total Time}}\right) \times 100$$

Example: A motor ran for 5 years total and was failed for 10 of those days.

$$\text{Reliability} = \left(1 - \frac{10}{5 \times 365}\right) \times 100 = 94.5\%$$

Unreliability is the complement:

$$\text{Unreliability} = 1 - \text{Reliability} = \frac{\text{Failed Time}}{\text{Total Time}} \times 100$$

The `ReliaLearnR` package includes helper functions for these calculations:

```
library(ReliaLearnR)

# Motor: 5 years total, 10 days failed
rel(outageTime = 10, totalTime = 5 * 365)
```

```
[1] 0.9945205
```

Review

A machine ran for 3 years total and was failed for 5 of those days. What are the reliability and unreliability?

Answer

Reliability = $(1 - 5/(3 \times 365)) \times 100 = 99.5\%$. Unreliability = 0.5%.

1.5 Availability

Availability accounts for both failed time **and** scheduled maintenance: any time the item is unavailable for service:

$$\text{Availability} = \left(1 - \frac{\text{Unavailable Time}}{\text{Total Time}}\right) \times 100$$

Example: A motor ran for 5 years, was failed for 10 days, and had 15 days of scheduled maintenance.

$$\text{Availability} = \left(1 - \frac{10 + 15}{5 \times 365}\right) \times 100 = 93.2\%$$

```
# Motor: 5 years, 10 days failed + 15 days maintenance = 25 unavailable
avail(unavailTime = 25, totalTime = 5 * 365)
```

```
[1] 0.9863014
```

Note: standby time (available but not in use) does not count as unavailable time.

Review

What is the difference between reliability and availability?

Answer

Availability includes scheduled maintenance downtime; reliability does not. Availability \leq Reliability (all failed time reduces both, but only availability is also reduced by planned maintenance).

1.6 Mean Time to Repair (MTTR)

MTTR measures maintainability: the average time required to repair a failed item:

$$\text{MTTR} = \frac{\sum_{i=1}^n \text{RepairTime}_i}{\text{RepairCount}}$$

Example: 5 failures with repair times of 5, 10, 15, 8, and 12 days.

$$\text{MTTR} = \frac{5 + 10 + 15 + 8 + 12}{5} = 10 \text{ days}$$

1.7 Mean Time to Failure (MTTF)

MTTF is a reliability measure for *non-repairable* items. Once the item fails, it is replaced rather than repaired.

$$\text{MTTF} = \frac{\text{Total Time}}{\text{FailureCount}}$$

Example: 100 motors run for 5 years total; 5 failures occur.

$$\text{MTTF} = \frac{5 \times 100}{5} = 100 \text{ years}$$

```
mttf(failures = 5, totalTime = 5 * 100)
```

```
[1] 100
```

1.8 Mean Time Between Failures (MTBF)

MTBF is the reliability measure for *repairable systems*, where the item is restored to service after each failure.

$$\text{MTBF} = \frac{\text{Total Time}}{\text{FailureCount}}$$

The formula is identical to MTTF, but the interpretation differs: MTBF is the average time *between* successive failures on the same system.

Example: A motor fails 5 times during 10,000 total operating hours.

$$\text{MTBF} = \frac{10,000}{5} = 2,000 \text{ hours}$$

```
mtbf(failures = 5, totalTime = 10000)
```

```
[1] 2000
```

Try It

A fleet of 50 generators runs for 8 years. During that time, 20 failures occur. Calculate the MTBF in years.

```
fleet_size <- 50
years      <- 8
failures   <- 20
# Calculate MTBF = (fleet_size * years) / failures
```

Solution

```
fleet_size <- 50
years      <- 8
failures   <- 20
MTBF <- (fleet_size * years) / failures
MTBF # 20 years
```

```
[1] 20
```

Review

What is the difference between MTTF and MTBF?

Answer

MTTF is for **non-repairable** items (replaced after failure); MTBF is for **repairable** systems (restored to service after failure). Both use the same formula, but MTBF represents the average time *between* successive failures of the same system.

1.9 Failure Rate

The **failure rate** λ is the inverse of MTBF:

$$\lambda = \frac{1}{\text{MTBF}} = \frac{\text{FailureCount}}{\text{Total Time}}$$

A key assumption of MTBF is a **constant** failure rate throughout the item's life. While this is not always true in practice, it is a reasonable approximation for many systems during their useful life period.

Example: 100 motors run for 10 years; 20 failures occur.

$$\lambda = \frac{20}{10 \times 100} = 0.02 \text{ failures per year}$$

```
fr(failures = 20, totalTime = 10 * 100)
```

```
[1] 0.02
```

1.10 Probability of Failure

Under the constant failure rate assumption, failure times follow an **exponential distribution**. The cumulative probability of failure prior to time t is:

$$F(t) = 1 - e^{-\lambda t}$$

And the cumulative probability of **survival** (reliability function) is:

$$R(t) = 1 - F(t) = e^{-\lambda t}$$

Example: A motor has a failure rate of 0.1 failures/year. Probability of surviving 10 years:

$$R(10) = e^{-10 \times 0.1} = e^{-1} \approx 36.8\%$$

```
lambda <- 0.1  
t       <- 10  
R <- exp(-lambda * t)  
R
```

```
[1] 0.3678794
```

1.10.1 The Reliability Curve

The shape of $R(t) = e^{-\lambda t}$ depends entirely on λ . Below are curves for three different failure rates:

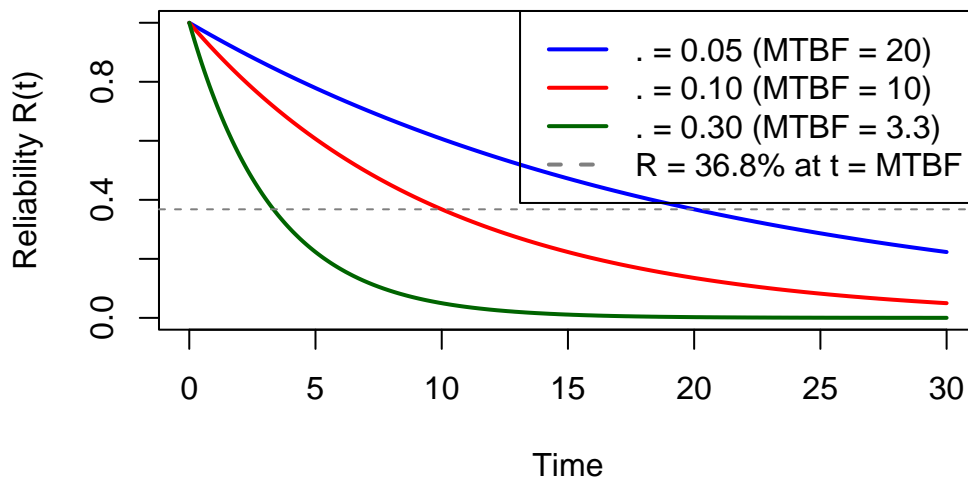
```

t <- seq(0, 30, by = 0.1)

plot(t, exp(-0.05 * t), type = "l", col = "blue", lwd = 2,
      xlab = "Time", ylab = "Reliability R(t)",
      main = "Exponential Reliability for Different Failure Rates",
      ylim = c(0, 1))
lines(t, exp(-0.1 * t), col = "red", lwd = 2)
lines(t, exp(-0.3 * t), col = "darkgreen", lwd = 2)
abline(h = exp(-1), col = "gray50", lty = 2)
legend("topright",
       legend = c("λ = 0.05 (MTBF = 20)", "λ = 0.10 (MTBF = 10)", "λ = 0.30
↪ (MTBF = 3.3)",
                 "R = 36.8% at t = MTBF"),
       col = c("blue", "red", "darkgreen", "gray50"),
       lty = c(1, 1, 1, 2), lwd = 2)

```

Exponential Reliability for Different Failure Rates



Note that at $t = \text{MTBF} = 1/\lambda$, reliability is always $e^{-1} \approx 36.8\%$ regardless of λ . This is a universal property of the exponential distribution.

i Try It

A pump has a failure rate of 0.05 failures/year. Calculate the probability of survival at $t = 10$ years.

```
lambda <- 0.05
t      <- 10
# R(t) = exp(-lambda * t)
```

Solution

```
lambda <- 0.05
t      <- 10
R <- exp(-lambda * t)
R # ~0.607 (60.7% survival)
```

```
[1] 0.6065307
```

1.11 B_n or L_n Life

The B_n life is the time at which $n\%$ of a population is expected to have failed. Some industries use L_n instead, but the concept is identical.

To find B_n , set $F(t) = n/100$ and solve for t :

$$B_n = -\frac{\ln(1 - n/100)}{\lambda}$$

Example: Motor with $\lambda = 0.2$ failures/year. B10 life:

$$B_{10} = -\frac{\ln(1 - 0.1)}{0.2} = 0.526 \text{ years}$$

```
lambda <- 0.2
B10 <- -log(1 - 0.10) / lambda
B10 # 0.526 years
```

```
[1] 0.5268026
```

Try It

A component has a failure rate of 0.05 failures/year. Calculate the B10 life.

```
lambda <- 0.05
# B10 = -log(1 - 0.10) / lambda
```

Solution

```
lambda <- 0.05
B10 <- -log(1 - 0.10) / lambda
B10 # ~2.1 years
```

```
[1] 2.10721
```

Review

What is the relationship between failure rate and MTBF?

Answer

Failure rate is the **inverse** of MTBF: $\lambda = 1/\text{MTBF}$, and $\text{MTBF} = 1/\lambda$.

1.12 From Exponential to Weibull

The exponential model assumes a **constant** failure rate. This is appropriate for electronic components that fail at random, but many mechanical components have failure rates that change over time.

The **Weibull distribution** (Abernethy 2004) generalizes the exponential by adding a shape parameter β :

$$R(t) = e^{-(t/\eta)^\beta}$$

β	Failure rate	Typical cause
< 1	Decreasing	Infant mortality
= 1	Constant	Random failures, exponential applies
> 1	Increasing	Wear-out (fatigue, corrosion, aging)

When $\beta = 1$, the Weibull reduces to the exponential with $\lambda = 1/\eta$.

1.13 The Bathtub Curve

The failure rate of a population of items over its lifetime traces a characteristic shape known as the **bathtub curve**, named for its distinctive U-profile. Rather than drawing it as a stylized cartoon, we can build it analytically by compositing the three Weibull failure modes:

$$h(t) = \underbrace{h_{\text{infant}}(t)}_{\beta < 1} + \underbrace{h_{\text{random}}(t)}_{\beta = 1} + \underbrace{h_{\text{wear-out}}(t)}_{\beta > 1}$$

where the Weibull hazard rate is $h(t) = (\beta/\eta)(t/\eta)^{\beta-1}$. Each term represents a physically distinct failure mechanism; the bathtub shape is not assumed, it *emerges* from their sum.

```
t <- seq(0.01, 15, by = 0.01)

# Three Weibull hazard rate components
h_infant <- function(t, b = 0.4, e = 2) (b/e) * (t/e)^(b-1) # infant
  ↪ mortality
h_random <- function(t, b = 1.0, e = 5) (b/e) * (t/e)^(b-1) # random
  ↪ failures
h_wearout <- function(t, b = 3.5, e = 10) (b/e) * (t/e)^(b-1) # wear-out

h_total <- h_infant(t) + h_random(t) + h_wearout(t)

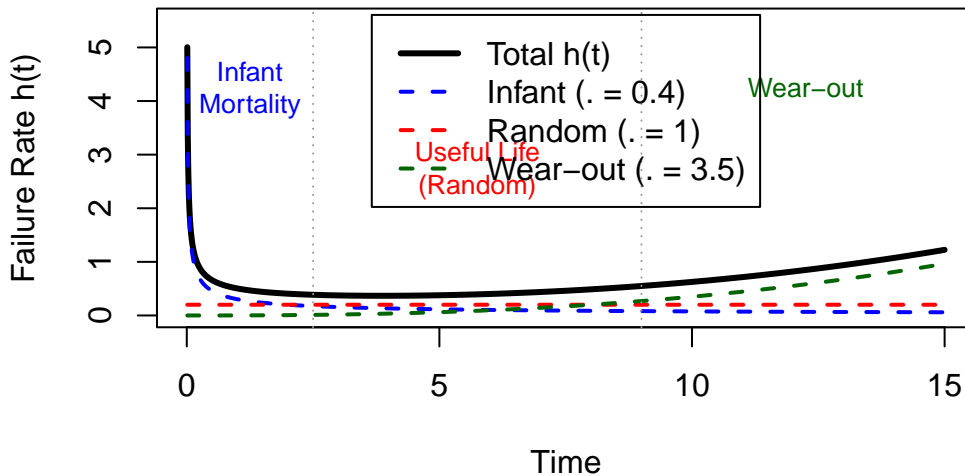
plot(t, h_total, type = "l", col = "black", lwd = 3,
     xlab = "Time", ylab = "Failure Rate h(t)",
     main = "The Bathtub Curve: Three Composited Failure Modes",
     ylim = c(0, max(h_total) * 1.1))
lines(t, h_infant(t), col = "blue", lwd = 2, lty = 2)
lines(t, h_random(t), col = "red", lwd = 2, lty = 2)
lines(t, h_wearout(t), col = "darkgreen", lwd = 2, lty = 2)

abline(v = c(2.5, 9), col = "gray60", lty = 3)
text(1.25, max(h_total) * 0.85, "Infant\nMortality", col = "blue",
     ↪ cex = 0.85)
text(5.75, max(h_total) * 0.55, "Useful Life\n(Random)", col = "red",
     ↪ cex = 0.85)
text(12.25, max(h_total) * 0.85, "Wear-out", col = "darkgreen",
     ↪ cex = 0.85)

legend("top", inset = 0.02,
     legend = c("Total h(t)", "Infant ( = 0.4)", "Random ( = 1)",
     ↪ "Wear-out ( = 3.5)"),
```

```
col = c("black", "blue", "red", "darkgreen"), lwd = c(3, 2, 2, 2), lty
↵ = c(1, 2, 2, 2))
```

The Bathtub Curve: Three Composited Failure Modes



Each region maps to a distinct engineering strategy:

- **Infant mortality** ($\beta < 1$): manufacturing defects, assembly errors, material flaws. Mitigated by *burn-in testing*, running units briefly before shipment to screen out defective items.
- **Useful life** ($\beta \approx 1$): random external shocks, load spikes, operator error. The constant failure rate means failures are unrelated to age; mitigated by *redundancy*.
- **Wear-out** ($\beta > 1$): fatigue, corrosion, material aging. Mitigated by *preventive replacement* scheduled before the B or B life is reached.

💡 Review

A pump has $\beta = 2.8$. Which region of the bathtub curve does it occupy, and what maintenance strategy is most appropriate?

Answer

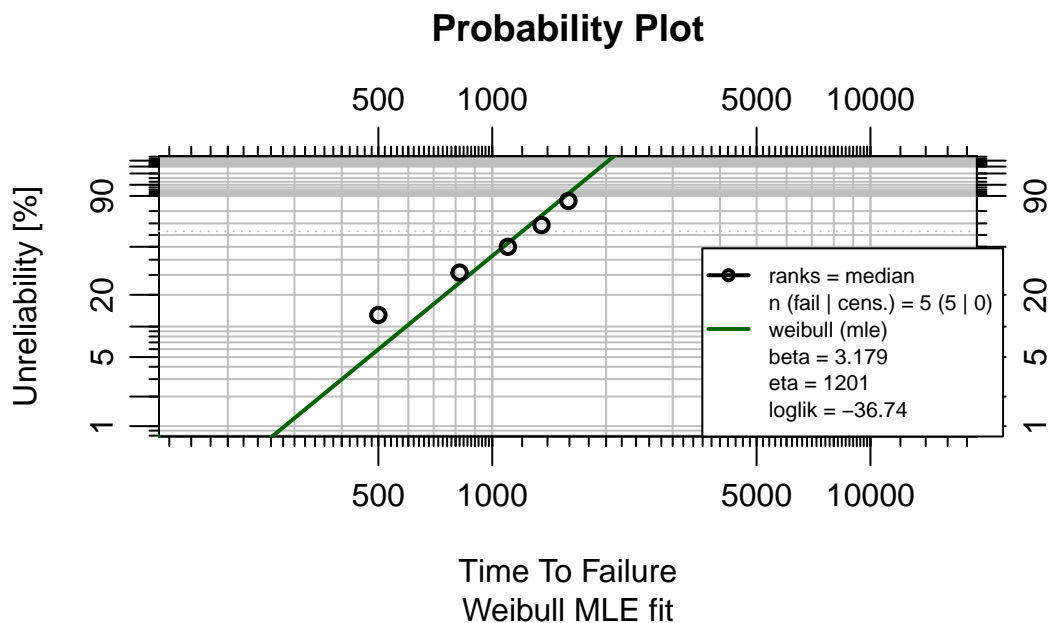
$\beta > 1$ places it in the **wear-out** region. The failure rate is increasing with age, so a time-based **preventive replacement** strategy, replacing the pump before it reaches its B or B life, is most appropriate.

The bathtub curve is a population-level concept: individual units fail at random times, but the aggregate failure rate of a large fleet traces this shape. Fitting a separate Weibull to each failure mode is covered in Chapter 3.

The `WeibullR` package (Silkworth and Symynck 2022) provides functions for Weibull fitting in R.

```
library(WeibullR)

failures <- c(500, 820, 1100, 1350, 1590)
fit <- MLEw2p(failures, show = TRUE)
```



A β close to 1 indicates random failures consistent with the exponential model. Higher values indicate wear-out behavior. The Life Data Analysis chapter covers Weibull analysis in depth.

1.14 System Reliability

Real systems combine many components. System reliability depends on how those components are arranged.

1.14.1 Series Systems

In a **series** configuration, every component must function for the system to function:

$$R_{\text{sys}} = R_1 \times R_2 \times \dots \times R_n = \prod_{i=1}^n R_i$$

A series system is always *less* reliable than its weakest component.

```
R_components <- c(0.90, 0.90, 0.90)
R_series <- prod(R_components)
R_series # 72.9%
```

```
[1] 0.729
```

1.14.2 Parallel Systems

In a **parallel** configuration, only one component needs to function, providing redundancy. The system fails only when *all* components fail:

$$R_{\text{sys}} = 1 - (1 - R_1)(1 - R_2) \dots (1 - R_n)$$

```
R_parallel <- 1 - prod(1 - R_components)
R_parallel # 99.9%
```

```
[1] 0.999
```

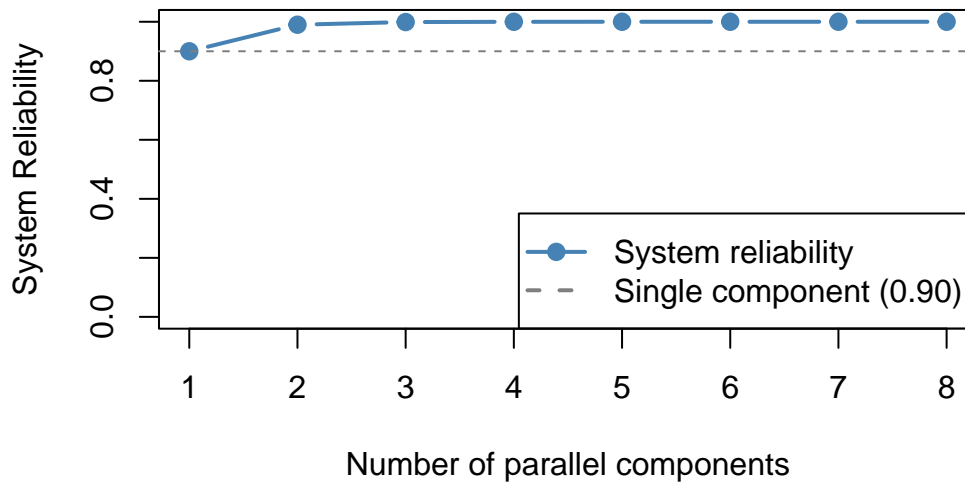
The effect of redundancy increases dramatically with the number of parallel components:

```
n_vals <- 1:8
R_c <- 0.90
R_par <- 1 - (1 - R_c)^n_vals

plot(n_vals, R_par, type = "b", col = "steelblue", lwd = 2, pch = 19,
     xlab = "Number of parallel components",
     ylab = "System Reliability",
     main = "Effect of Redundancy on System Reliability (R_component =
     ↪ 0.90)",
     ylim = c(0, 1))
```

```
abline(h = R_c, col = "gray50", lty = 2)
legend("bottomright",
      legend = c("System reliability", "Single component (0.90)"),
      col = c("steelblue", "gray50"), lty = c(1, 2), pch = c(19, NA), lwd =
      ↪ 2)
```

Effect of Redundancy on System Reliability ($R_{\text{component}} =$



i Try It

Four components each have reliability $R = 0.85$. Calculate the series and parallel system reliability.

```
R_comp <- 0.85
# Series: R_sys = prod(R_components)
# Parallel: R_sys = 1 - prod(1 - R_components)
```

Solution

```
R_comp <- rep(0.85, 4)
R_series <- prod(R_comp)
R_parallel <- 1 - prod(1 - R_comp)
R_series # ~0.522 (52.2%)
```

```
[1] 0.5220062
```

```
R_parallel # ~0.9995 (99.95%)
```

```
[1] 0.9994937
```

Review

Three components each have 80% reliability. What is the series system reliability?

Answer

$$R_{\text{sys}} = 0.8 \times 0.8 \times 0.8 = 0.512 = 51.2\%$$

1.15 Summary

Key takeaways:

- $R(t) = e^{-\lambda t}$, exponential reliability under a constant failure rate.
- At $t = \text{MTBF} = 1/\lambda$, reliability is always 36.8%.
- MTTF is for non-repairable items; MTBF is for repairable systems.
- Series systems: $R_{\text{sys}} = \prod R_i$, reliability decreases with more components.
- Parallel systems: $R_{\text{sys}} = 1 - \prod(1 - R_i)$, redundancy increases reliability.
- When $\beta \neq 1$, use the Weibull distribution (see Chapter 3).

2 Reliability Block Diagrams

2.1 Introduction

In Chapter 1, you learned to calculate reliability metrics for individual components. This chapter extends those concepts to *systems*, collections of components whose arrangement determines whether the system succeeds or fails.

A **Reliability Block Diagram (RBD)** is a graphical tool for modeling how component reliabilities combine to produce system-level reliability. RBDs are widely used in reliability engineering to analyze designs, identify vulnerabilities, and evaluate the benefit of redundancy (Billinton and Allan 1992).

2.2 Learning Objectives

By the end of this chapter, you will be able to:

- Define reliability block diagrams and relate them to physical systems.
- Compute system reliability for series, parallel, and mixed configurations.
- Interpret k-out-of-n (voting) systems and calculate their reliability.
- Describe the relationship between RBDs and Fault Tree Analysis.
- Apply RBD calculations to a realistic multi-component system.

2.3 What is a Reliability Block Diagram?

An RBD represents each component as a **block** with a known reliability value. Blocks are connected by lines showing logical dependencies, not physical connections. A path from left (input) to right (output) through functioning blocks represents a working system.

Each block has a reliability $R_i(t)$ derived from the exponential model (Chapter 1) or a Weibull fit (Chapter 3).

```
library(DiagrammeR)
```

2.4 Series Systems

In a **series** system, all components must function for the system to succeed:

$$R_{\text{sys}} = R_1 \times R_2 \times \dots \times R_n = \prod_{i=1}^n R_i$$

A series system is *always less reliable* than its weakest component. Adding more components in series can only reduce system reliability.

2.4.1 Example: Water Pumping System

A pumping system requires a motor ($R = 0.95$), a pump ($R = 0.90$), and a control valve ($R = 0.98$) to all be operational.

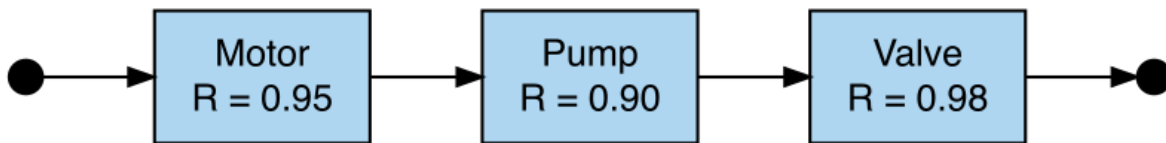


Figure 2.1: Series RBD: Motor → Pump → Valve (all must work)

```
R_motor <- 0.95
R_pump <- 0.90
R_valve <- 0.98

R_series <- R_motor * R_pump * R_valve
R_series # ~0.838
```

```
[1] 0.8379
```

The system reliability is approximately 83.8%, lower than any individual component.

i Try It

A conveyor belt system has 5 components in series with reliabilities 0.98, 0.96, 0.99, 0.94, and 0.97. Calculate the system reliability.

```
R_components <- c(0.98, 0.96, 0.99, 0.94, 0.97)
# R_series <- prod(R_components)
```

Solution

```
R_components <- c(0.98, 0.96, 0.99, 0.94, 0.97)
R_series <- prod(R_components)
R_series # ~0.845
```

```
[1] 0.8492432
```

 Review

In a series system, which component has the greatest influence on system reliability?

Answer

The **least reliable** component. It is the bottleneck; improving it produces the largest gain in system reliability.

2.5 Parallel Systems

In a **parallel** system, only one component needs to function, through **active redundancy** (all components operate simultaneously). The system fails *only* if all components fail at the same time. Since component failures are assumed to be independent:

$$P(\text{all fail}) = \prod_{i=1}^n (1 - R_i)$$

$$R_{\text{sys}} = 1 - \prod_{i=1}^n (1 - R_i)$$

Adding parallel components always increases system reliability.

2.5.1 Example: Backup Power System

A primary generator ($R = 0.90$) and a standby generator ($R = 0.85$); either alone keeps the system running.

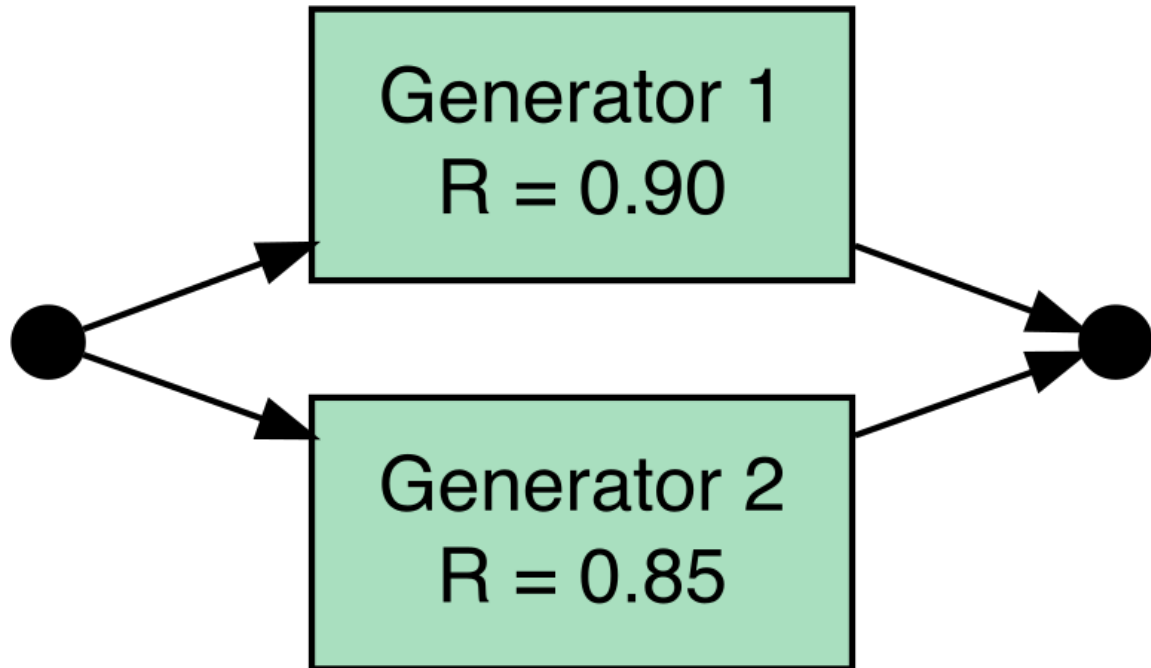


Figure 2.2: Parallel RBD: Generator 1 and Generator 2 (either can work)

```
R_gen1 <- 0.90 # Generator 1
R_gen2 <- 0.85 # Generator 2

R_parallel <- 1 - (1 - R_gen1) * (1 - R_gen2)
R_parallel # 0.985
```

```
[1] 0.985
```

The parallel system reliability is 98.5%, much higher than either generator alone.

The benefit of redundancy grows with each additional component:

```
n_vals <- 1:8
Rc      <- 0.90
R_par   <- 1 - (1 - Rc)^n_vals

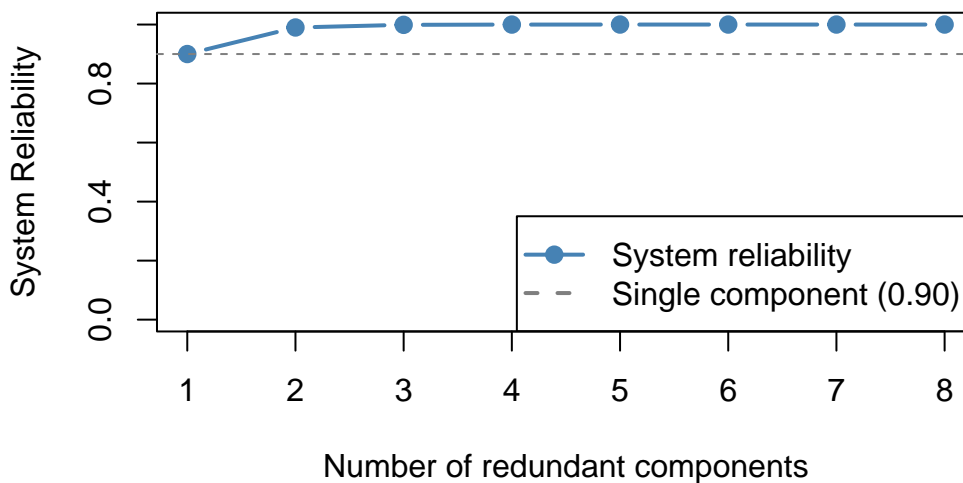
plot(n_vals, R_par, type = "b", col = "steelblue", lwd = 2, pch = 19,
     xlab = "Number of redundant components",
     ylab = "System Reliability",
```

```

main = "Parallel System Reliability vs. Redundancy (R_component =
↪ 0.90)",
ylim = c(0, 1))
abline(h = Rc, col = "gray50", lty = 2)
legend("bottomright",
      legend = c("System reliability", "Single component (0.90)"),
      col = c("steelblue", "gray50"), lty = c(1, 2), pch = c(19, NA), lwd =
↪ 2)

```

Parallel System Reliability vs. Redundancy (R_component =



i Try It

A critical pump station has 3 pumps in parallel. Each pump has reliability 0.88. What is the system reliability?

```

R_components <- c(0.88, 0.88, 0.88)
# R_parallel <- 1 - prod(1 - R_components)

```

Solution

```

R_components <- c(0.88, 0.88, 0.88)
R_parallel <- 1 - prod(1 - R_components)
R_parallel # ~0.9983

```

```
[1] 0.998272
```

2.6 Mixed Systems

Most real systems combine series and parallel blocks. To analyze a **mixed** system, decompose it into subsystems and apply series and parallel rules step by step, working from the innermost blocks outward.

2.6.1 Example: Safety System

- **Subsystem A:** a sensor ($R = 0.95$) and a transmitter ($R = 0.97$) in series.
- **Subsystem B:** two redundant actuators (each $R = 0.90$) in parallel.
- Subsystems A and B in series (both must work).

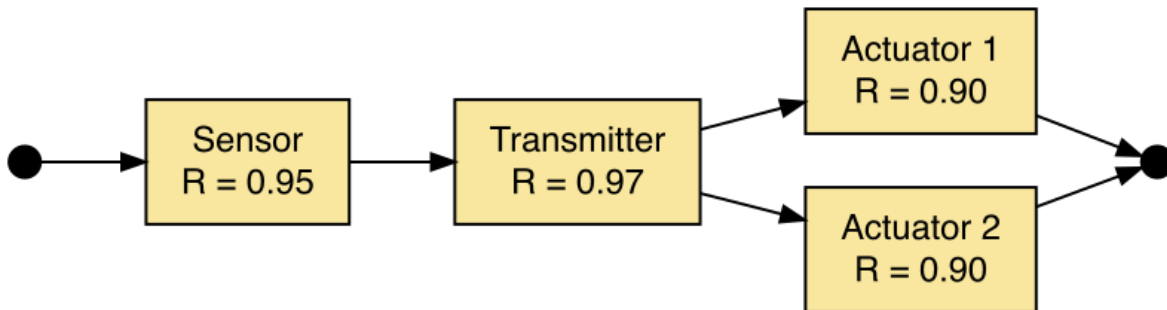


Figure 2.3: Mixed RBD: Sensor–Transmitter in series, Actuators 1 & 2 in parallel

```
# Subsystem A (series)
R_A <- 0.95 * 0.97
R_A
```

```
[1] 0.9215
```

```
# Subsystem B (parallel)
R_B <- 1 - (1 - 0.90) * (1 - 0.90)
R_B
```

```
[1] 0.99
```

```
# Overall system (A and B in series)
R_system <- R_A * R_B
R_system
```

```
[1] 0.912285
```

💡 Review

A system has two subsystems in series. Subsystem 1 has two components in parallel (each $R = 0.85$); Subsystem 2 is a single component with $R = 0.92$. What is the system reliability?

Answer

$$R_{\text{sub1}} = 1 - (1 - 0.85)^2 = 0.9775. \quad R_{\text{sys}} = 0.9775 \times 0.92 \approx 0.899.$$

2.7 k-out-of-n Systems

A **k-out-of-n** system succeeds if at least k of its n identical components function:

- $k = n$: all must work \rightarrow series.
- $k = 1$: at least one works \rightarrow parallel.
- $1 < k < n$: voting or load-sharing system.

The reliability of a k-out-of-n system (identical components, each with reliability p) follows the binomial distribution:

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

The compact R expression counts the probability that *at least* k components function: $\text{pbinom}(k-1, n, 1-p)$ gives the probability that $k-1$ or fewer components work (i.e., the system fails), and we subtract that from 1.

2.7.1 Example: Flight Control Computers

A 2-out-of-3 voter: at least 2 of 3 computers must agree. Each computer has $R = 0.99$.

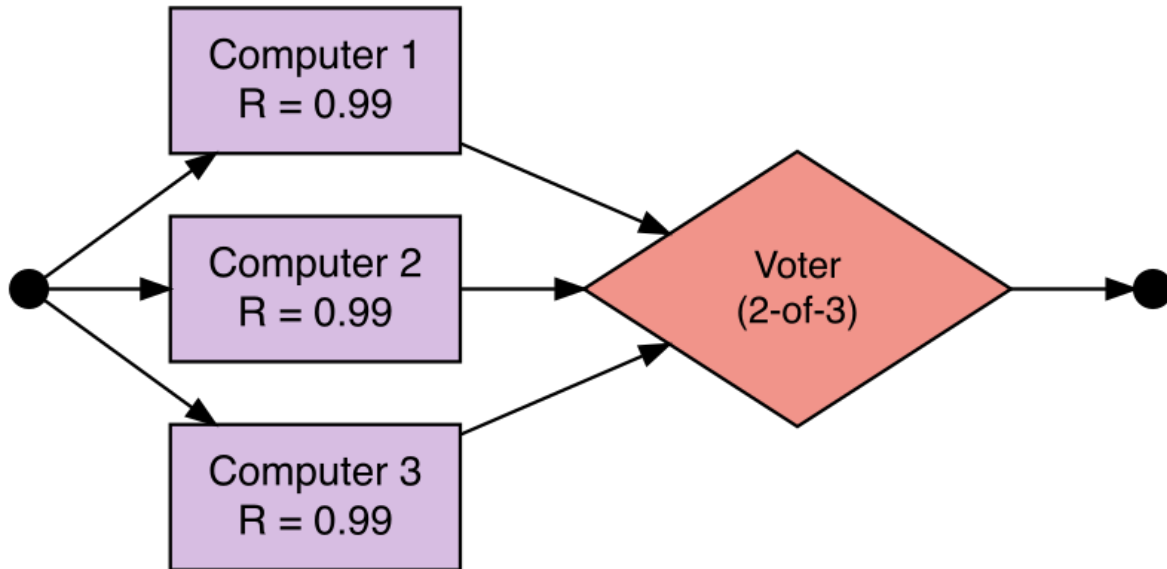


Figure 2.4: k-out-of-n RBD: 2-of-3 voting system with three computers

```
n <- 3      # total components
k <- 2      # minimum required
p <- 0.99   # individual reliability

R_voting <- 1 - pbinom(k - 1, n, 1 - p)
R_voting # 0.9997
```

[1] 0.000298

i Try It

A 3-out-of-5 redundant sensor array. Each sensor has reliability $p = 0.95$. Calculate the system reliability.

```
n <- 5
k <- 3
p <- 0.95
# R_sys <- 1 - pbinom(k - 1, n, 1 - p)
```

Solution

```
n <- 5
k <- 3
p <- 0.95
R_sys <- 1 - pbinom(k - 1, n, 1 - p)
R_sys # ~0.9988
```

```
[1] 0.001158125
```

2.8 System MTTF

For a **series** system of components with constant failure rates, the system failure rate equals the sum of component failure rates:

$$\lambda_{\text{sys}} = \sum_{i=1}^n \lambda_i \quad \text{MTTF}_{\text{sys}} = \frac{1}{\lambda_{\text{sys}}}$$

For n identical parallel components each with failure rate λ :

$$\text{MTTF}_{\text{parallel}} = \frac{1}{\lambda} \left(1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} \right)$$

2.8.1 Example: Two Pumps in Parallel

```
lambda <- 0.01 # failures per hour

MTTF_single <- 1 / lambda
MTTF_parallel <- (1 / lambda) * (1 + 1/2)

MTTF_single # 100 hours
```

```
[1] 100
```

```
MTTF_parallel # 150 hours - 50% longer with one spare
```

```
[1] 150
```

💡 Review

Three components in series have failure rates 0.02, 0.03, and 0.05 failures/hour. What is the system MTTF?

Answer

$\lambda_{\text{sys}} = 0.02 + 0.03 + 0.05 = 0.10$, so $\text{MTTF} = 1/0.10 = 10$ hours.

2.9 Common Cause Failure

All the analyses above assume that component failures are **statistically independent**: if Component A fails, it tells us nothing about Component B's probability of failing. In practice, this assumption can break down.

Common Cause Failure (CCF) occurs when a single event simultaneously causes multiple components to fail. Typical sources include:

- A shared environment (vibration, temperature, contamination) that degrades all components together
- A common design flaw or manufacturing batch defect
- A maintenance error that affects multiple redundant units at once

CCF is especially dangerous because **redundancy does not protect against it**: if both pumps in a parallel system can be destroyed by the same flooding event, adding a third pump does not help.

2.9.1 The Beta-Factor Model

The simplest CCF model splits each component's total failure rate λ into two parts:

$$\lambda_{\text{ind}} = (1 - \beta_{\text{CCF}}) \cdot \lambda \quad (\text{independent failures})$$

$$\lambda_{\text{CCF}} = \beta_{\text{CCF}} \cdot \lambda \quad (\text{common-cause failures})$$

The **beta-factor** β_{CCF} is the fraction of failures attributable to common causes. Typical values range from 0.01 (1%) for well-separated, dissimilar components to 0.20 (20%) for closely coupled identical units.

System reliability for a two-component parallel system with CCF:

$$R_{\text{sys}} = \underbrace{[1 - (1 - R_{\text{ind}})^2]}_{\text{parallel, independent part}} \times \underbrace{e^{-\lambda_{\text{CCF}} t}}_{\text{common-cause part}}$$

```

lambda <- 0.001 # total component failure rate (per hour)
t       <- 1000 # mission time (hours)

beta_vals <- c(0, 0.05, 0.10, 0.20)

results <- sapply(beta_vals, function(b) {
  R_ind <- exp(-(1 - b) * lambda * t) # independent reliability
  R_ccf <- exp(-b * lambda * t)      # common-cause reliability
  R_sys <- (1 - (1 - R_ind)^2) * R_ccf
  round(R_sys, 4)
})

data.frame(beta_CCF = beta_vals, R_system = results)

```

	beta_CCF	R_system
1	0.00	0.6004
2	0.05	0.5935
3	0.10	0.5862
4	0.20	0.5705

Even a modest $\beta_{CCF} = 0.10$ visibly reduces the system reliability that redundancy would otherwise provide.

Review

Why doesn't adding more parallel components help when β_{CCF} is large?

Answer

When β_{CCF} is large, a significant fraction of failures are **common cause**, simultaneously affecting all components regardless of how many there are. Adding more identical, coupled components may even increase the common-cause failure rate if they share the same environment or design flaw. Effective mitigation requires **diversity** (different designs, manufacturers, or locations) rather than simple redundancy.

CCF data and beta-factor values for different equipment types are tabulated in IEC 61508 (functional safety) and MIL-HDBK-217F (electronic reliability). For nuclear applications, the NUREG/CR-5497 handbook provides detailed CCF databases.

2.10 Introduction to Fault Tree Analysis

Fault Tree Analysis (FTA) (Vesely et al. 1981) is a top-down approach that starts with an undesirable event (the **top event**) and works backward to identify combinations of component

failures that could cause it.

While an RBD asks “What must work for the system to succeed?”, a fault tree asks “What can cause the system to fail?”

2.10.1 Gates

- **AND gate:** top event occurs only if *all* inputs occur, corresponding to **parallel** RBD components (all must fail).
- **OR gate:** top event occurs if *any* input occurs, corresponding to **series** RBD components (any failure causes system failure).

2.10.2 RBD—Fault Tree Duality

RBD configuration	Fault tree gate for system failure
Series (all must work)	OR gate (any failure causes system failure)
Parallel (any can work)	AND gate (all must fail for system failure)

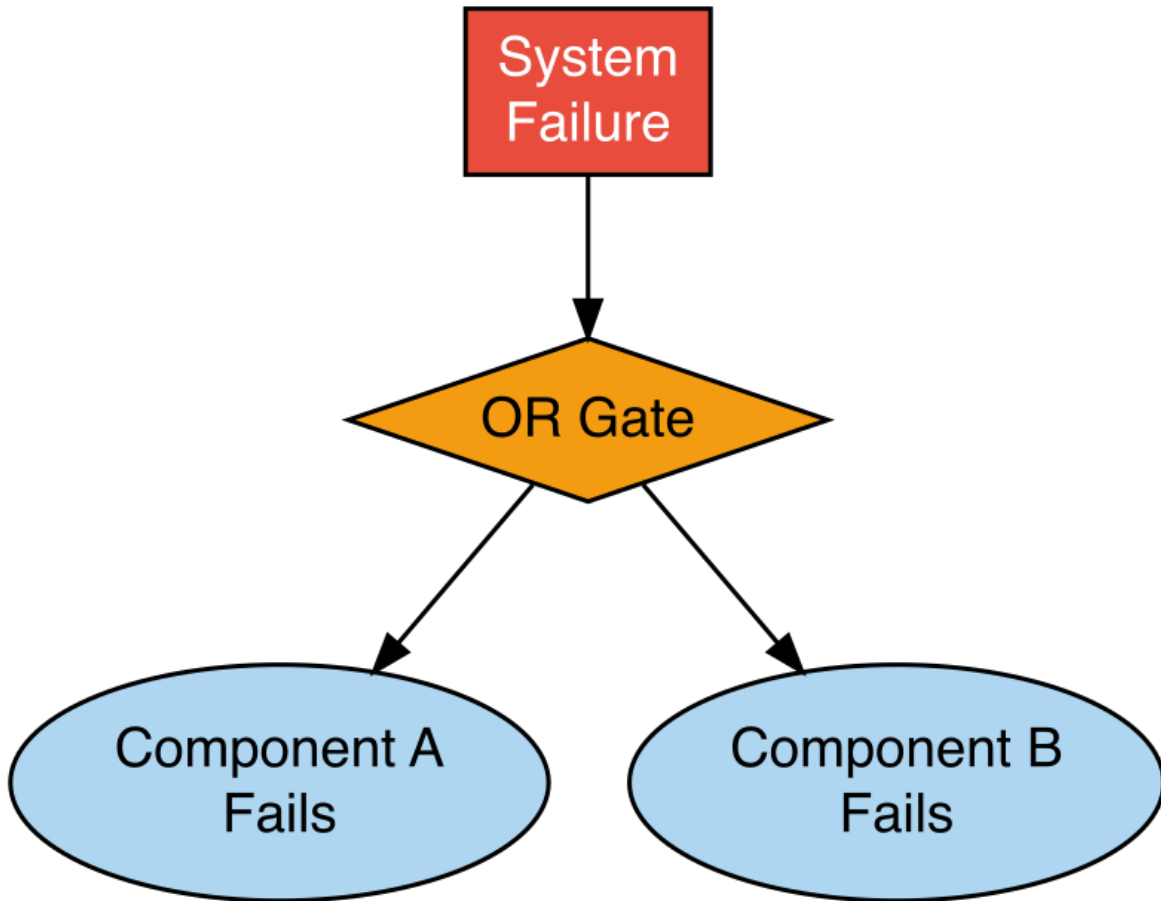


Figure 2.5: Fault Tree: OR gate — system failure if Component A or B fails

This OR gate corresponds to a two-component series RBD: any single failure causes system failure. For complex fault trees, the R package `FaultTree` on CRAN provides computational support.

 Review

A fault tree has an AND gate combining two component failures. Which RBD configuration does this correspond to?

Answer

Parallel: an AND gate means the system fails only if both components fail simultaneously, which is exactly the parallel (redundant) configuration.

2.11 Case Study: Industrial Cooling System

An industrial cooling system has the following architecture:

1. **Water supply subsystem:** two pumps in parallel (each $R = 0.92$), followed by a filter in series ($R = 0.99$).
2. **Control subsystem:** a primary controller ($R = 0.97$) and a backup controller in parallel ($R = 0.95$).
3. Both subsystems must work (series at system level).

```
# Step 1 - Water supply subsystem
R_pump_parallel <- 1 - (1 - 0.92)^2
R_water <- R_pump_parallel * 0.99
R_water
```

```
[1] 0.983664
```

```
# Step 2 - Control subsystem
R_control <- 1 - (1 - 0.97) * (1 - 0.95)
R_control
```

```
[1] 0.9985
```

```
# Step 3 - Overall system (both subsystems in series)
R_system <- R_water * R_control
R_system
```

```
[1] 0.9821885
```

i Try It

Modify the case study: the filter is upgraded to $R = 0.999$. Recalculate the system reliability.

```
R_pump1 <- 0.92
R_pump2 <- 0.92
R_filter <- 0.999 # upgraded from 0.99
R_ctrl1 <- 0.97
R_ctrl2 <- 0.95
```

Solution

```
R_pump1 <- 0.92
R_pump2 <- 0.92
R_filter <- 0.999
R_ctrl1 <- 0.97
R_ctrl2 <- 0.95

R_pump_parallel <- 1 - (1 - R_pump1) * (1 - R_pump2)
R_water <- R_pump_parallel * R_filter
R_control <- 1 - (1 - R_ctrl1) * (1 - R_ctrl2)
R_system <- R_water * R_control
R_system # ~0.985
```

```
[1] 0.9911175
```

2.12 Summary

Key takeaways:

- **Series:** $R_{\text{sys}} = \prod R_i$, every component must work; reliability decreases with more components.
- **Parallel:** $R_{\text{sys}} = 1 - \prod(1 - R_i)$, redundancy; only one component needs to work.
- **Mixed:** decompose into subsystems, apply formulas step by step.
- **k-out-of-n:** $1 - \text{pbinom}(k-1, n, 1-p)$ for voting/load-sharing systems.
- **Series MTTF:** $1/\sum \lambda_i$; **Parallel MTTF:** $(1/\lambda) \sum_{i=1}^n (1/i)$.
- **Common Cause Failure:** redundancy requires *diversity* to be effective; the β_{CCF} factor quantifies the fraction of failures caused by shared events, and adding more identical, coupled components does not protect against CCF.
- **FTA duality:** series RBD OR gate; parallel RBD AND gate.

3 Life Data Analysis

3.1 Introduction

Life Data Analysis is the study of how systems function over time. Life data tells us how long a system will last, when it will likely fail, and how often it will need maintenance. In reliability engineering it is often called **Weibull analysis** because the Weibull distribution is the dominant model (Abernethy 2004).

3.2 Learning Objectives

By the end of this chapter, you will be able to:

- Describe the purpose of Weibull analysis in reliability engineering.
- Differentiate between types of data censoring: right-censored and interval-censored.
- Differentiate between Weibull models: 2-parameter, 3-parameter, and Weibayes.
- Apply Median Rank Regression (MRR) and Maximum Likelihood Estimation (MLE).
- Compare Weibull and lognormal fits using goodness-of-fit statistics.
- Determine how many failures are needed for reliable Weibull estimates.
- Interpret results using probability plots and contour plots.

3.3 Life Distributions

3.3.1 The Weibull Distribution

The **Weibull distribution** is a flexible continuous probability distribution widely used in life data analysis. Its flexibility comes from a shape parameter β that allows it to mimic the normal, exponential, and log-normal distributions.

3.3.2 The Reliability (Survival) Function

The Weibull **reliability** (survival) function is:

$$R(t) = e^{-(t/\eta)^\beta}$$

where $R(t)$ is the probability of *surviving beyond* time t , β is the shape parameter, and η is the scale parameter. The **CDF** (cumulative probability of failure by time t) is its complement: $F(t) = 1 - R(t)$.

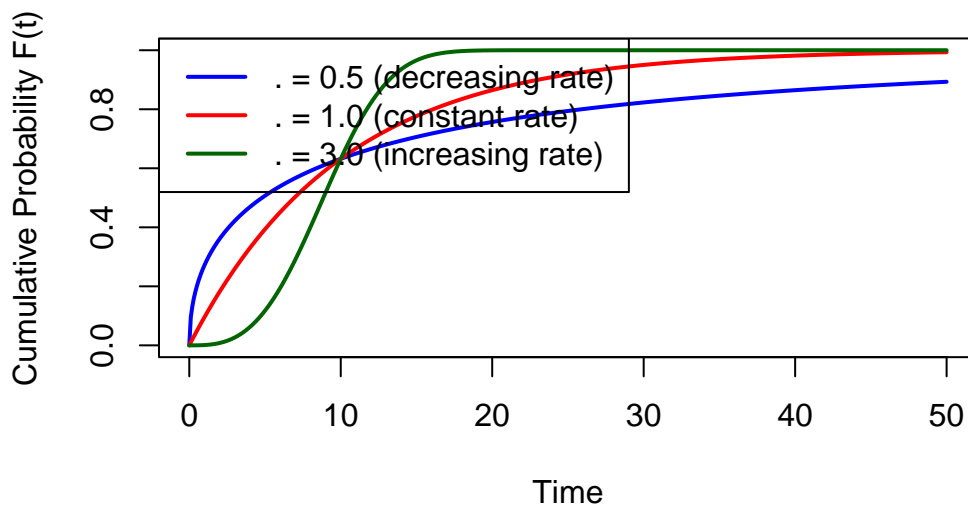
The shape parameter β has a direct physical interpretation:

β	Failure rate	Typical cause
< 1	Decreasing	Infant mortality
$= 1$	Constant	Random failures (exponential)
> 1	Increasing	Wear-out (fatigue, corrosion, aging)

```
t <- seq(0, 50, by = 0.1)
etas <- 10

# Plot CDFs for three values of beta
plot(t, pweibull(t, shape = 0.5, scale = etas), type = "l", col = "blue",
     ↪ lwd = 2,
     xlab = "Time", ylab = "Cumulative Probability F(t)",
     main = "Weibull CDF for Different Shape Parameters ( = 10)",
     ylim = c(0, 1))
lines(t, pweibull(t, shape = 1.0, scale = etas), col = "red", lwd = 2)
lines(t, pweibull(t, shape = 3.0, scale = etas), col = "darkgreen", lwd = 2)
legend("topleft",
     legend = c(" = 0.5 (decreasing rate)", " = 1.0 (constant rate)",
               " = 3.0 (increasing rate)"),
     col = c("blue", "red", "darkgreen"), lwd = 2)
```

Weibull CDF for Different Shape Parameters ($\eta = 10$)



Key property: at $t = \eta$, $F(\eta) = 1 - e^{-1} \approx 63.2\%$ for any value of β . This is why η is called the **characteristic life**.

💡 Review

Which value of β indicates an increasing failure rate?

Answer

$\beta > 1$. A shape parameter greater than 1 indicates a wear-out failure mode where the failure rate increases over time.

3.4 WeibullR

The `WeibullR` package (Silkworth and Symynck 2022; Silkworth 2020) is the primary R tool for Weibull analysis.

3.4.1 Getting Started

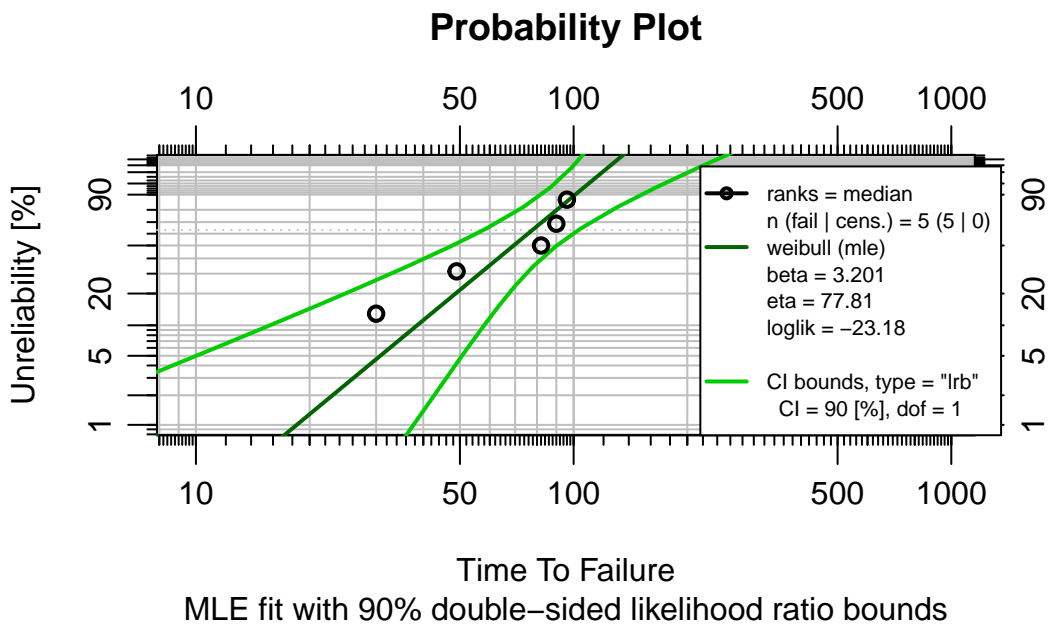
```
install.packages("WeibullR")
```

```
library(WeibullR)
```

3.4.2 Fitting a Weibull Model

A factory has 5 machines that failed at times 30, 49, 82, 90, and 96. Use `MLEw2p()` to fit a 2-parameter Weibull model using Maximum Likelihood Estimation and display the probability plot.

```
failures <- c(30, 49, 82, 90, 96)
fit <- MLEw2p(failures, bounds = TRUE, show = TRUE)
```



3.4.3 Reading the Probability Plot

The **Weibull probability plot** shows time on the horizontal axis (log scale) and cumulative unreliability $F(t)$ on the vertical axis (log-log scale). This double-log transformation linearizes Weibull data; the result is a straight line when the Weibull model fits.

- Read off $F(t)$ at any time to get the probability of failure by that time.
- Reliability at time t is $R(t) = 1 - F(t)$.
- β (shape) and η (scale) are shown in the legend.

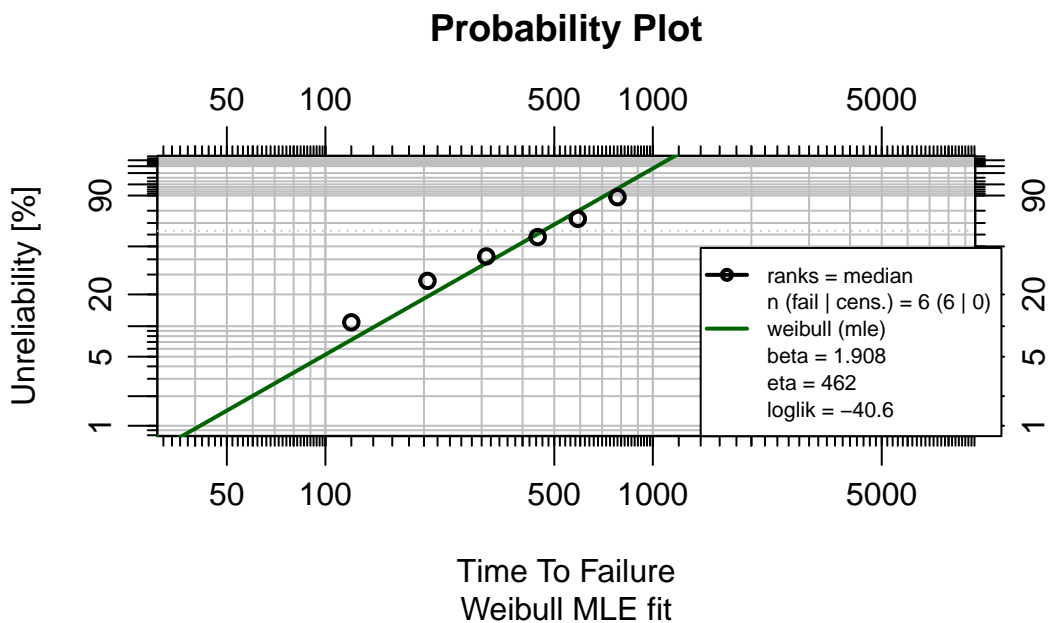
i Try It

A batch of industrial pumps failed at times (hours): 120, 205, 310, 445, 590, 780. Use `MLEw2p()` to fit a 2-parameter Weibull model.

```
failures <- c(120, 205, 310, 445, 590, 780)
# fit <- MLEw2p(failures, show = TRUE)
```

Solution

```
failures <- c(120, 205, 310, 445, 590, 780)
fit <- MLEw2p(failures, show = TRUE)
```



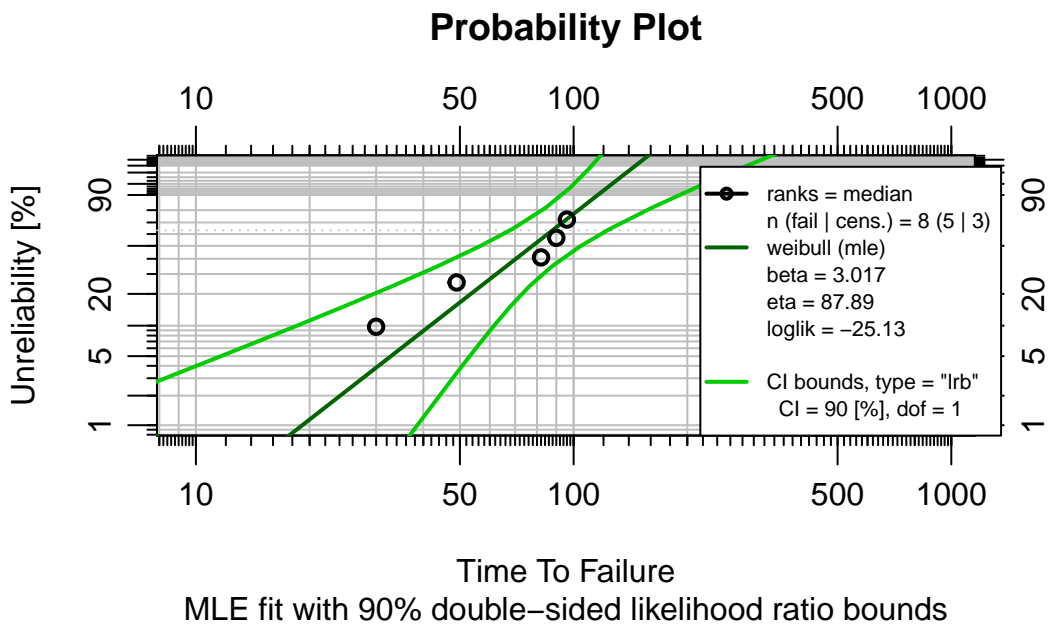
3.5 Data Censoring

In life data analysis, data is often **censored**: the exact time to failure is not known for all units.

3.5.1 Right-Censored Data (Suspensions)

Right censoring occurs when a unit has operated for a period of time without failing (a **suspension**). The unit was removed from service before failing.

```
failures <- c(30, 49, 82, 90, 96)
suspensions <- c(100, 45, 10)
fit <- MLEw2p(failures, suspensions, bounds = TRUE, show = TRUE)
```



Adding suspensions changes the fit: the model accounts for units that survived longer, adjusting β and η accordingly.

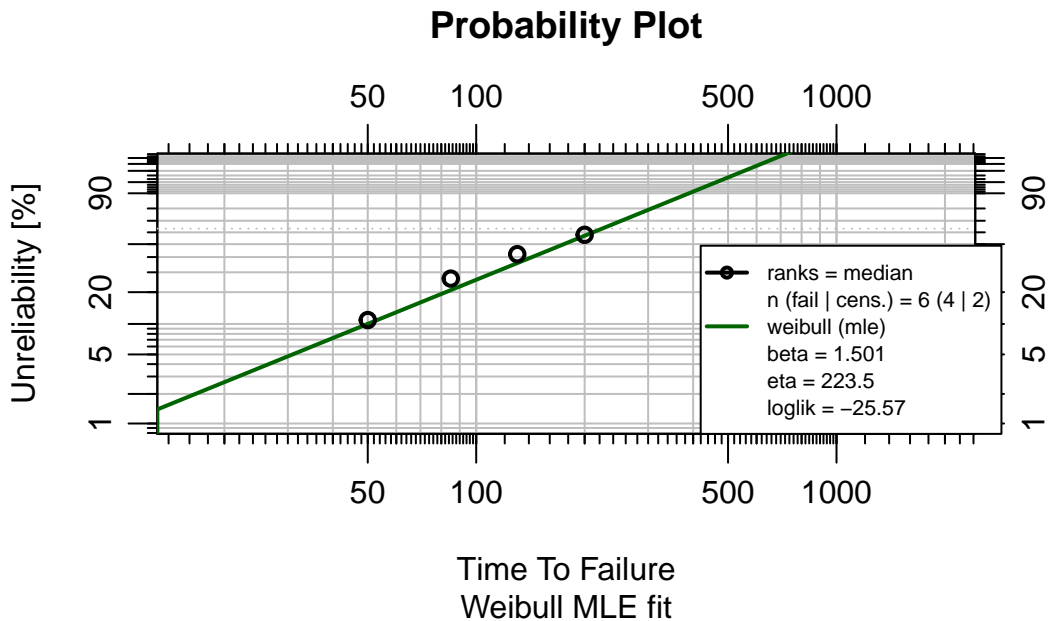
i Try It

Four compressors failed at 50, 85, 130, and 200 hours. Two additional units were removed at 250 hours without failing. Fit a Weibull model with suspensions.

```
failures <- c(50, 85, 130, 200)
suspensions <- c(250, 250)
# fit <- MLEw2p(failures, suspensions, show = TRUE)
```

Solution

```
failures <- c(50, 85, 130, 200)
suspensions <- c(250, 250)
fit <- MLEw2p(failures, suspensions, show = TRUE)
```

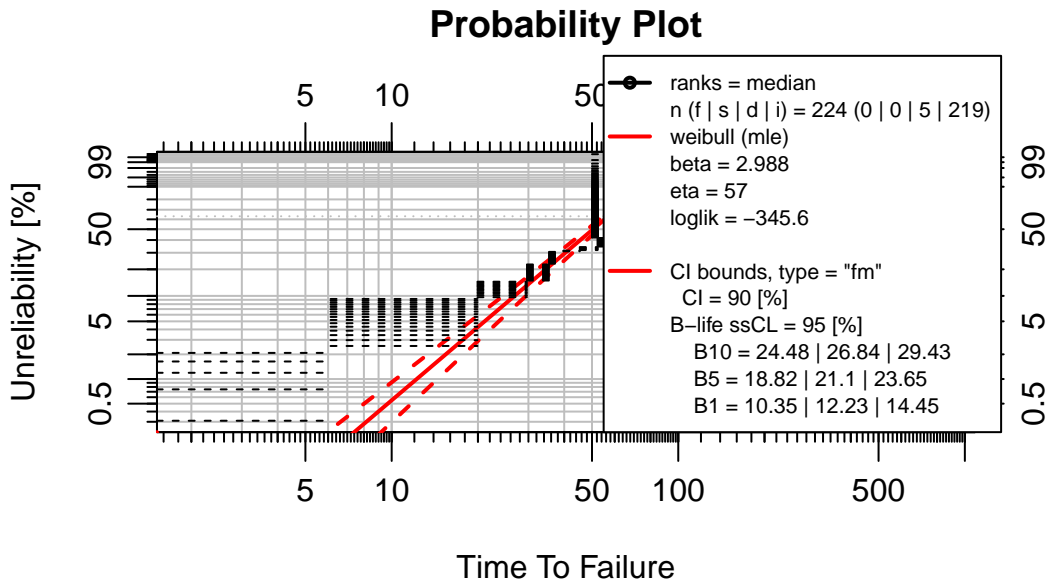


3.5.2 Interval-Censored Data

Interval censoring occurs when failures are only detected during periodic inspections; the exact failure time is unknown, but it is known to have occurred within an interval.

```
inspection_data <- data.frame(
  left = c(0, 6.12, 19.92, 29.64, 35.4, 39.72, 45.32, 52.32),
  right = c(6.12, 19.92, 29.64, 35.4, 39.72, 45.32, 52.32, 63.48),
  qty = c(5, 16, 12, 18, 18, 2, 6, 17)
)
suspensions <- data.frame(time = 63.48, event = 0, qty = 73)

obj1 <- wblr(suspensions, interval = inspection_data)
obj1 <- wblr.fit(obj1, method.fit = "mle", col = "red")
obj1 <- wblr.conf(obj1, method.conf = "fm", lty = 2)
plot(obj1)
```



Black horizontal lines represent the inspection intervals; the solid red line is the model fit; dashed red lines are confidence bounds.

3.5.3 Grouped Data

Life data is often collected in groups. The `qty` column in the inspection data above indicates how many units failed in each interval: 5 failed between time 0 and 6.12, 16 between 6.12 and 19.92, and so on.

💡 Review

In the interval-censored example above, how many total failures occurred?

Answer

$5 + 16 + 12 + 18 + 18 + 2 + 6 + 17 = 94$ failures. The 73 suspensions were units that did not fail by the last inspection time (63.48). Total units in the dataset: $94 + 73 = 167$.

3.6 Parameter Estimation Methods

3.6.1 MRR vs MLE

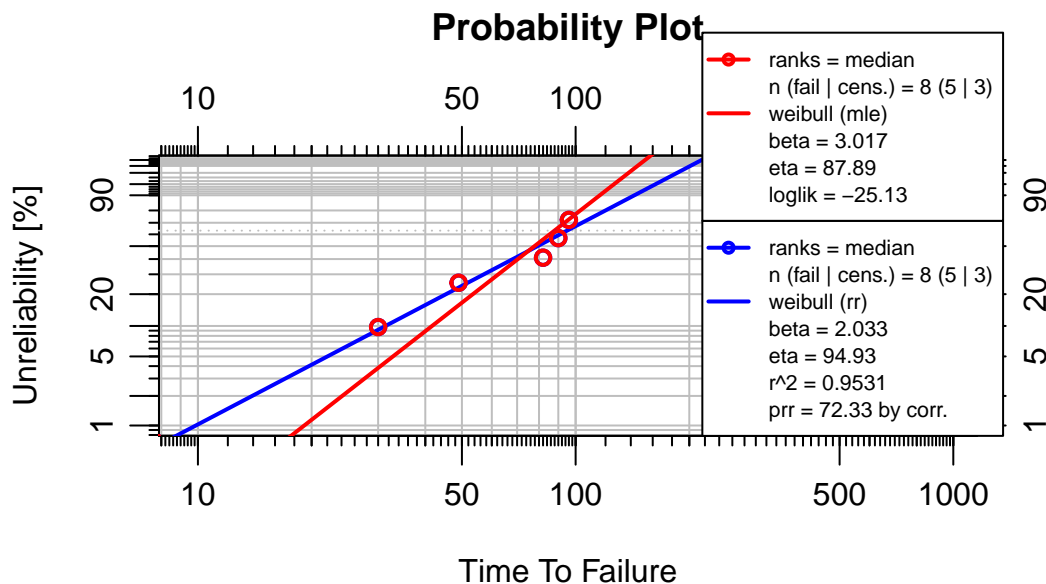
Median Rank Regression (MRR) estimates parameters by minimizing the sum of squared errors between observed and predicted values (least squares).

Maximum Likelihood Estimation (MLE) finds the parameters that make the observed data most probable: it maximizes the likelihood function.

Comparing both methods on the same data:

```
failures <- c(30, 49, 82, 90, 96)
suspensions <- c(100, 45, 10)

MRRfit <- wblr.fit(wblr(failures, suspensions, col = "blue"), method.fit =
  ↪ "rr")
MLEfit <- wblr.fit(wblr(failures, suspensions, col = "red"), method.fit =
  ↪ "mle")
plot.wblr(list(MRRfit, MLEfit))
```



Both methods fit the same data. Slight differences in β and η are visible; the better fit can be determined objectively using the Anderson-Darling statistic. In practice, **MLE is preferred**

for small samples and censored data (it uses all the information in the likelihood); MRR is adequate for larger, complete datasets and is computationally simpler.

3.7 Goodness of Fit

The **Anderson-Darling (AD) statistic** measures how closely the fitted CDF matches the empirical distribution. A **lower AD** value indicates a better fit:

AD Value	Interpretation
< 0.3	Good fit
0.3 – 0.6	Acceptable fit
> 0.6	Poor fit

WeibullR displays the AD statistic in the plot legend. Use it to compare fitting methods or distributions.

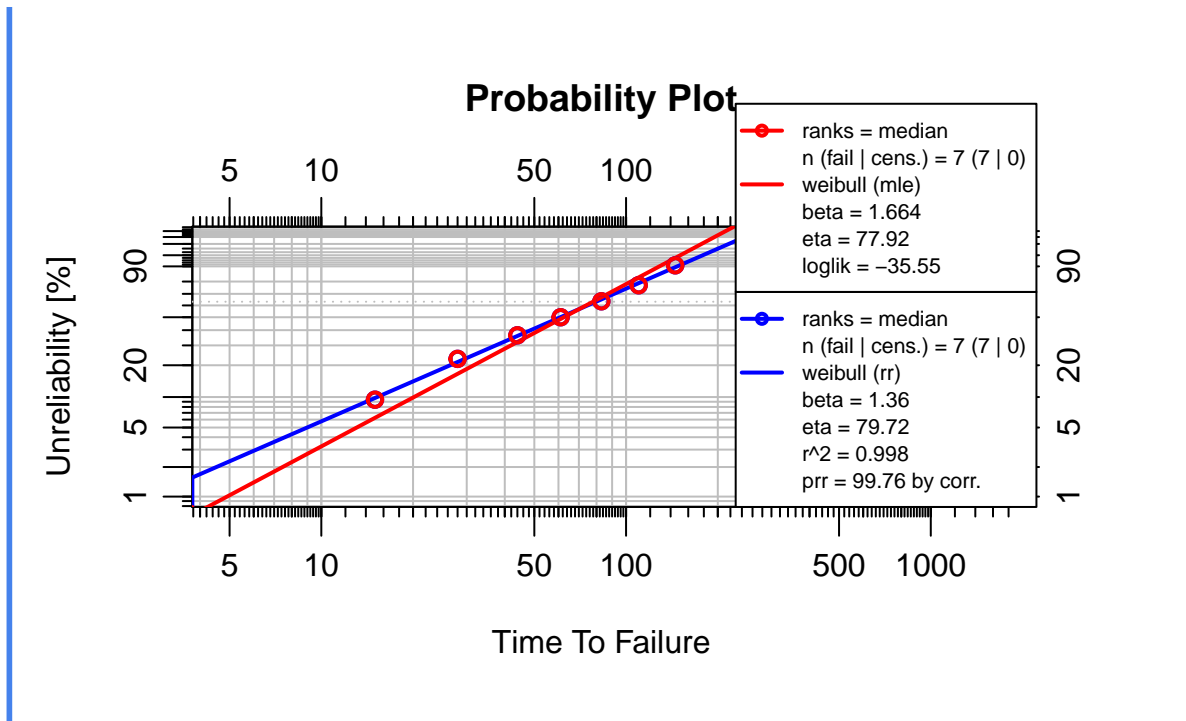
i Try It

Fit the following failure data with both MRR and MLE, then compare their AD statistics on a combined probability plot.

```
failures <- c(15, 28, 44, 61, 83, 110, 145)
```

Solution

```
failures <- c(15, 28, 44, 61, 83, 110, 145)
obj_mrr <- wblr.fit(wblr(failures, col = "blue"), method.fit = "rr")
obj_mle <- wblr.fit(wblr(failures, col = "red"), method.fit = "mle")
plot.wblr(list(obj_mrr, obj_mle))
```



3.8 Choosing a Distribution

The Weibull distribution is the default in reliability engineering, but it is not always the best fit. The **lognormal** distribution often fits failure data driven by fatigue crack growth, corrosion, or cumulative damage, mechanisms where the logarithm of time-to-failure is normally distributed.

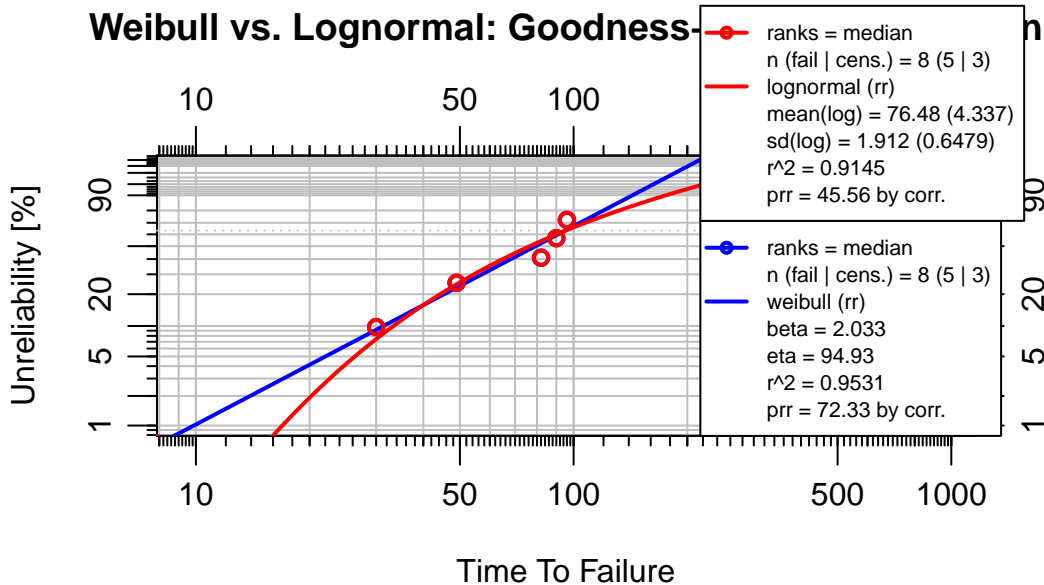
Distribution	When it tends to fit best
Weibull	Wear-out, mechanical fatigue, early-life failures
Lognormal	Corrosion, fatigue crack propagation, electronic degradation

`wblr.fit()` supports both via the `dist` argument. The goodness-of-fit metric in the plot legend (r^2 for rank regression) is the tiebreaker: **higher r^2 indicates a better fit**:

```
failures <- c(30, 49, 82, 90, 96)
suspensions <- c(100, 45, 10)
```

```
obj_wb <- wblr.fit(wblr(failures, suspensions, col = "blue"),
  method.fit = "rr")
obj_ln <- wblr.fit(wblr(failures, suspensions, col = "red"),
  method.fit = "rr", dist = "lognormal")

plot.wblr(list(obj_wb, obj_ln),
  main = "Weibull vs. Lognormal: Goodness-of-Fit Comparison")
```



For this dataset, Weibull achieves $r^2 \approx 0.953$ vs. lognormal at $r^2 \approx 0.914$; Weibull fits better. The choice should be driven by the data, not by habit.

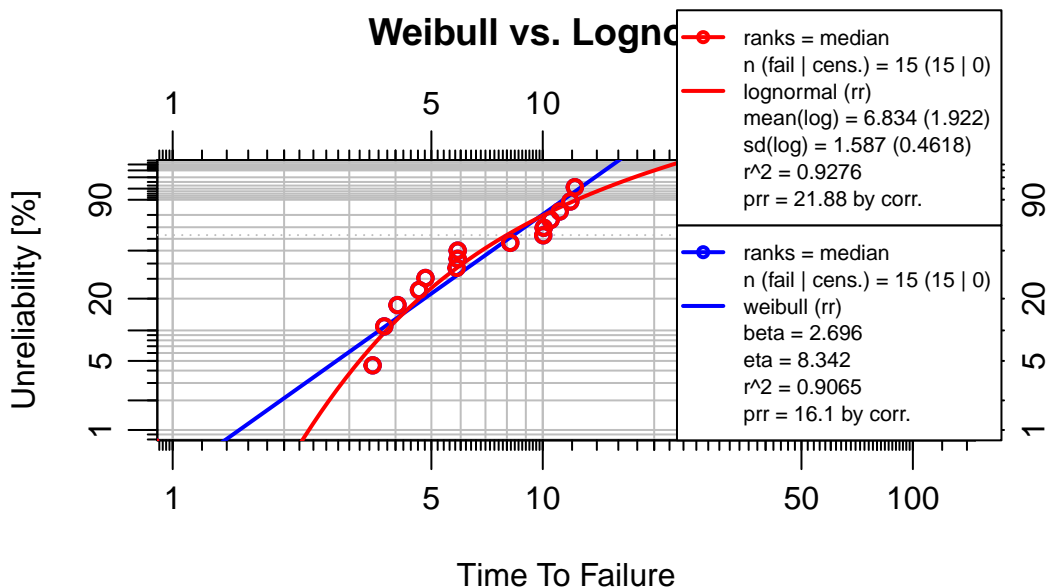
i Try It

Fit the 3-parameter Weibull failure dataset below with both Weibull and lognormal distributions. Which fits better?

```
failures_3p <- c(3.47, 3.73, 4.05, 4.63, 4.82, 5.85, 5.89, 5.89,
  8.17, 10.03, 10.06, 10.50, 11.11, 11.87, 12.21)
# obj_wb <- wblr.fit(wblr(failures_3p, col = "blue"), method.fit = "rr")
# obj_ln <- wblr.fit(wblr(failures_3p, col = "red"),
#   method.fit = "rr", dist = "lognormal")
# plot.wblr(list(obj_wb, obj_ln))
```

Solution

```
failures_3p <- c(3.47, 3.73, 4.05, 4.63, 4.82, 5.85, 5.89, 5.89,  
                8.17, 10.03, 10.06, 10.50, 11.11, 11.87, 12.21)  
obj_wb <- wblr.fit(wblr(failures_3p, col = "blue"), method.fit = "rr")  
obj_ln <- wblr.fit(wblr(failures_3p, col = "red"),  
                  method.fit = "rr", dist = "lognormal")  
plot.wblr(list(obj_wb, obj_ln),  
          main = "Weibull vs. Lognormal")
```



Compare the r^2 values in the legend. For this dataset the lognormal often achieves a slightly higher r^2 , suggesting cumulative-damage behavior, where the logarithm of time-to-failure is approximately normally distributed.

3.9 Sample Size Considerations

Weibull parameter estimates are most reliable when based on at least 10–15 **observed failures**. With fewer failures, uncertainty grows substantially: confidence bounds widen, r^2 loses discriminating power, and estimated β and η can deviate significantly from the true values.

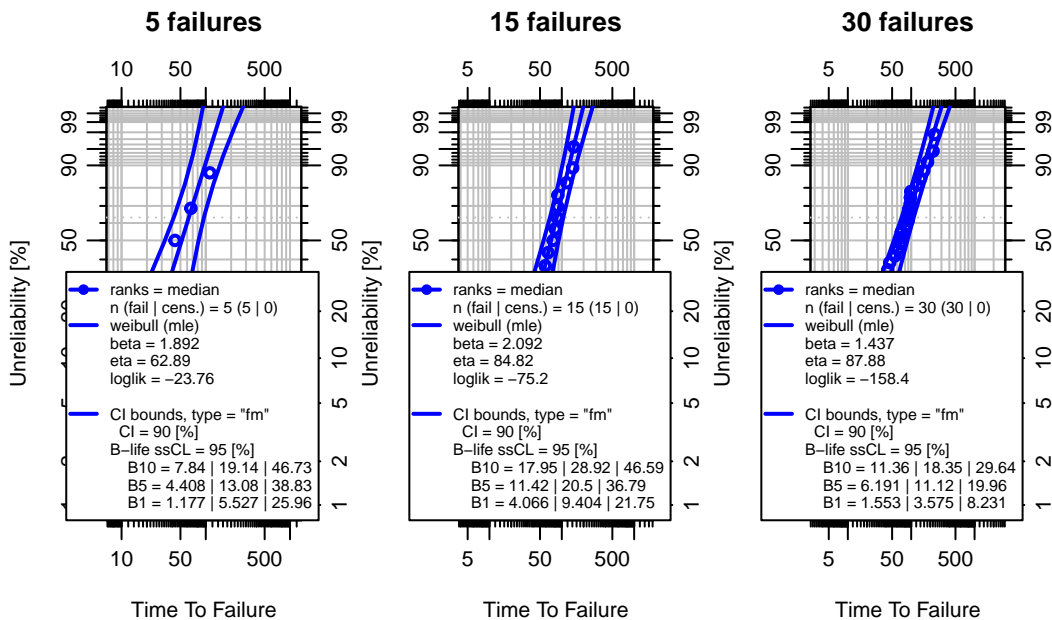
The following code simulates datasets of different sizes from the same Weibull ($\beta = 2$, $\eta = 100$) and fits each, showing how confidence bounds tighten as sample size increases:

```

set.seed(42)
eta_true <- 100; beta_true <- 2

par(mfrow = c(1, 3))
for (n in c(5, 15, 30)) {
  f <- sort(rweibull(n, shape = beta_true, scale = eta_true))
  obj <- wblr.conf(wblr.fit(wblr(f, col = "blue"), method.fit = "mle"),
                  method.conf = "fm")
  plot(obj, main = paste0(n, " failures"))
}

```



```

par(mfrow = c(1, 1))

```

With only 5 failures the confidence bounds span most of the probability axis. At 30 failures they are narrow enough for confident decision-making.

Practical guidance:

Failures observed	Guidance
< 5	Estimates unreliable; use Weibayes (fix β from prior knowledge)

Failures observed	Guidance
5–10	Usable with caution; widen safety margins
10–20	Acceptable for most engineering decisions
> 20	Reliable estimates; confidence bounds are tight

When fewer than 5 failures are available, the Weibayes model (next section) provides a structured way to incorporate prior knowledge about β .

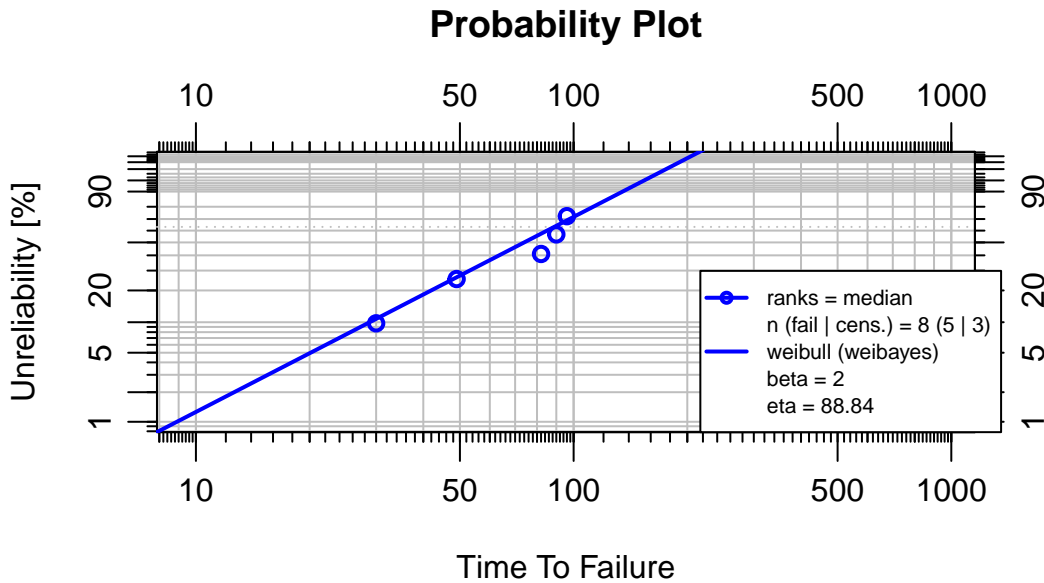
3.10 Other Weibull Models

3.10.1 The Weibayes Model

A **Weibayes** (one-parameter Weibull) has a fixed β based on prior knowledge or experience. This is appropriate when failure data are scarce.

```
failures    <- c(30, 49, 82, 90, 96)
suspensions <- c(100, 45, 10)

obj <- wblr.fit(wblr(failures, suspensions, col = "blue"),
               method.fit = "weibayes", weibayes.beta = 2)
plot(obj)
```



Confidence bounds are not shown for Weibayes because β is fixed by assumption.

3.10.2 The 3-Parameter Weibull

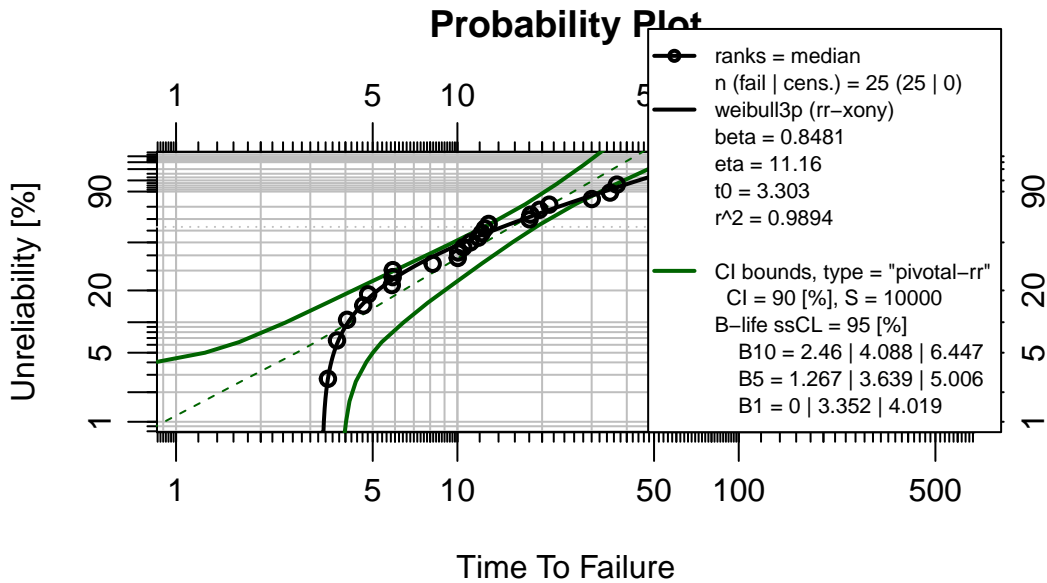
The **3-parameter Weibull** adds a failure-free period t_0 , the time before which no failures can occur:

$$R(t) = e^{-((t-t_0)/\eta)^\beta}$$

This is appropriate for components that must age, wear, or fatigue before they can fail.

```
failures_3p <- c(
  3.46623, 3.73271, 4.05300, 4.62870, 4.81570,
  5.84517, 5.88831, 5.89297, 8.16836, 10.02799,
  10.06062, 10.49785, 11.11493, 11.87369, 12.21122,
  12.51854, 12.91357, 18.04246, 18.20712, 19.57305,
  21.20873, 30.03917, 34.88001, 36.87355, 53.91168
)

fit_3p <- wblr.conf(wblr.fit(wblr(failures_3p), dist = "weibull3p"), col =
  ↪ "darkgreen")
plot(fit_3p)
```



The failure-free period is approximately 3.3 time units, visible as the curve flattening at low probabilities.

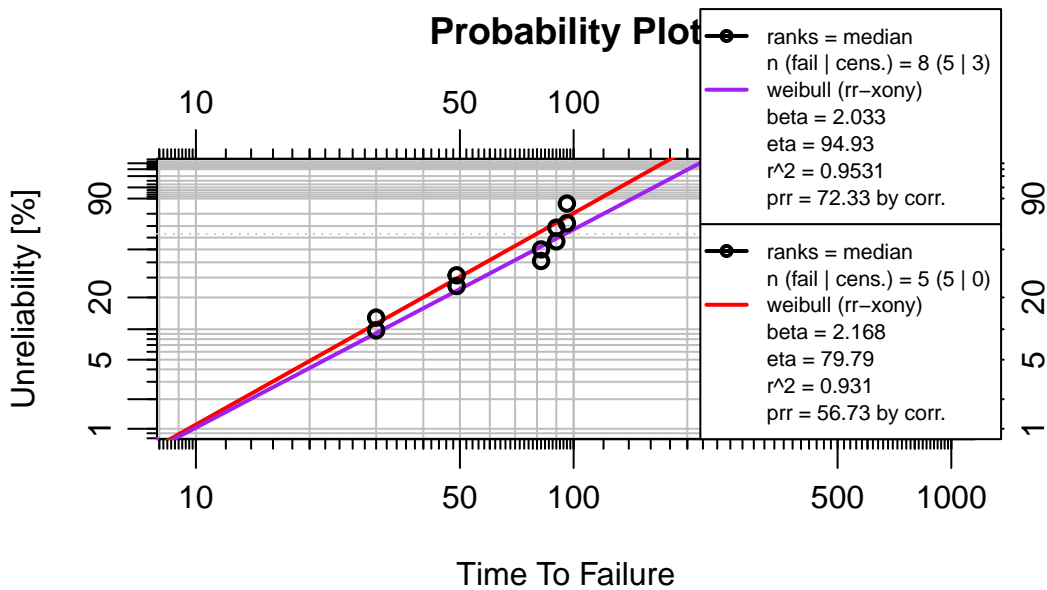
3.11 Multi-Plots

Multi-plots overlay multiple Weibull fits on one chart, making comparisons easy.

```
failures <- c(30, 49, 82, 90, 96)
suspensions <- c(100, 45, 10)

obj1 <- wblr.fit(wblr(failures), col = "red")
obj2 <- wblr.fit(wblr(failures, suspensions), col = "purple")

plot.wblr(list(obj1, obj2))
```



The red model (without suspensions) has a higher beta and lower eta than the purple model (with suspensions). Adding suspensions shifts the fitted distribution to account for units that survived longer than the observed failures.

3.12 Competing Failure Modes

When a component can fail in several distinct ways (corrosion, fatigue, overload), fitting a single Weibull to the combined data produces a poor fit because each mode has its own distribution.

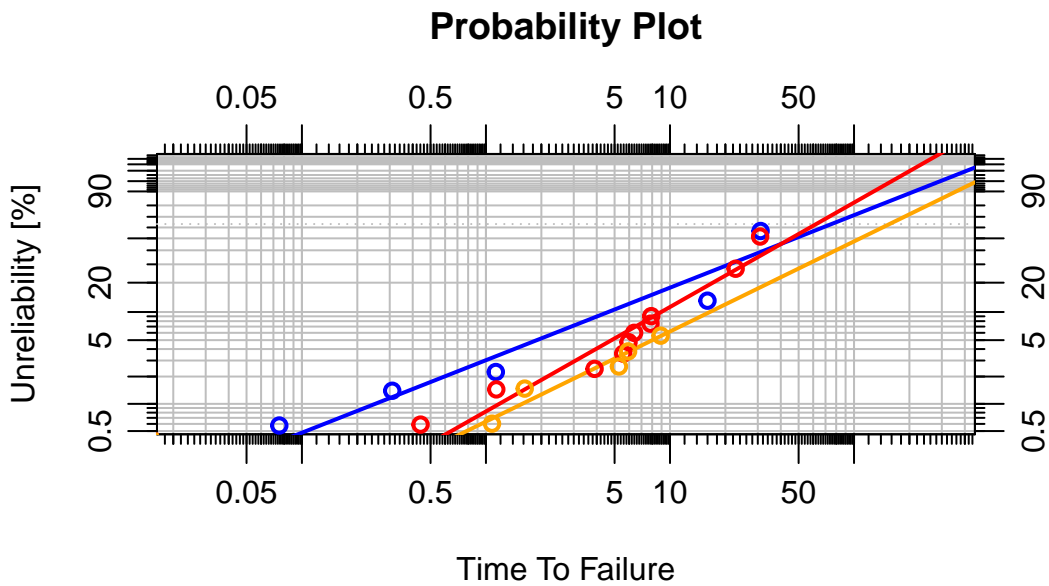
The approach: fit a separate Weibull to each failure mode, treating failures from other modes as suspensions.

```
set.seed(123)
data <- data.frame(
  time = c(
    rweibull(5, 0.5, 20), # Failure Mode A
    rweibull(10, 1, 10), # Failure Mode B
    rweibull(5, 2, 5), # Failure Mode C
    rweibull(100, 2, 10) # Suspensions
  ),
  event = c(rep(1, 20), rep(0, 100)),
  failure_mode = c(rep("A", 5), rep("B", 10), rep("C", 5), rep("", 100))
)
```

```
)

# Separate dataset per failure mode (others treated as suspensions)
dat1 <- dat2 <- dat3 <- data
dat1$event[dat1$failure_mode != "A"] <- 0
dat2$event[dat2$failure_mode != "B"] <- 0
dat3$event[dat3$failure_mode != "C"] <- 0

obj1 <- wblr.fit(wblr(dat1, col = "blue"))
obj2 <- wblr.fit(wblr(dat2, col = "red"))
obj3 <- wblr.fit(wblr(dat3, col = "orange"))
plot.wblr(list(obj1, obj2, obj3), is.plot.legend = FALSE)
```



Each failure mode has a distinct β : Mode A ($\beta < 1$, decreasing rate, infant mortality), Mode B ($\beta \approx 1$, random), Mode C ($\beta > 1$, wear-out).

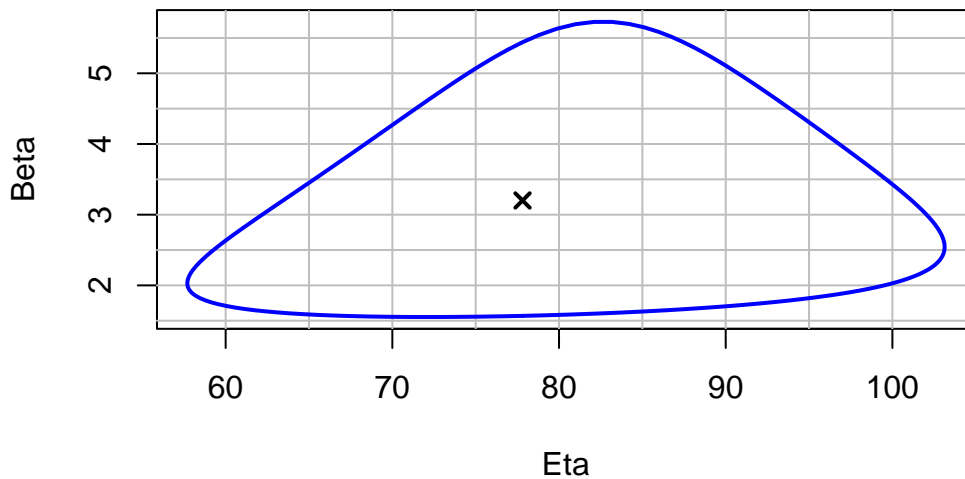
3.13 Contour Plots

A **contour plot** shows the joint confidence region for β and η . Overlapping contours indicate that two distributions are not significantly different at the chosen confidence level.

```

failures <- c(30, 49, 82, 90, 96)
obj <- wblr.conf(
  wblr.fit(wblr(failures, col = "blue"), method.fit = "mle"),
  method.conf = "lrb"
)
plot_contour(obj, CL = 0.9)

```



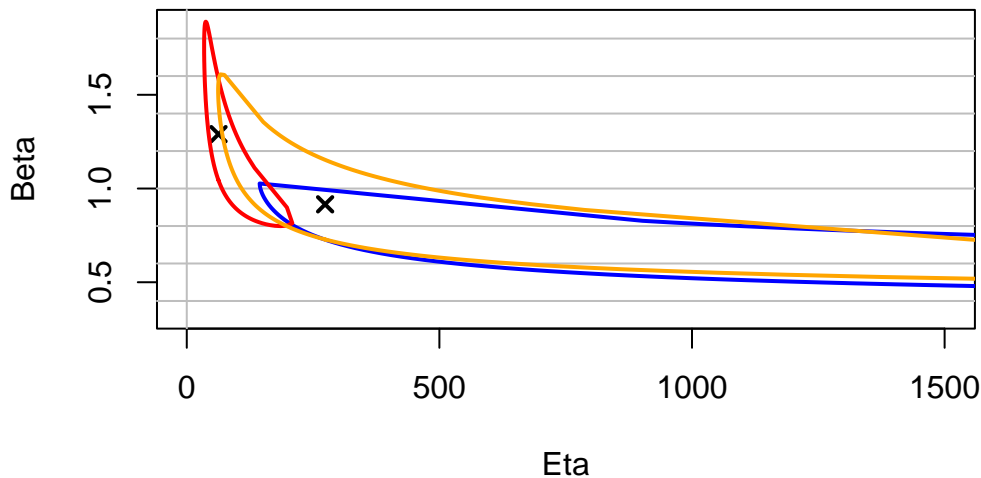
3.13.1 Comparing Contour Plots

```

obj1 <- wblr.conf(
  wblr.fit(wblr(dat1, col = "blue"), method.fit = "mle"),
  method.conf = "lrb"
)
obj2 <- wblr.conf(
  wblr.fit(wblr(dat2, col = "red"), method.fit = "mle"),
  method.conf = "lrb"
)
obj3 <- wblr.conf(
  wblr.fit(wblr(dat3, col = "orange"), method.fit = "mle"),
  method.conf = "lrb"
)

```

```
)
plot_contour(list(obj1, obj2, obj3), CL = 0.9, xlim = c(1, 1500))
```



Overlapping contours at 90% confidence level indicate that the three failure modes are not statistically distinguishable at this confidence level.

3.14 Summary

Key takeaways:

- The Weibull CDF: $R(t) = e^{-(t/\eta)^\beta}$.
- $\beta < 1$: decreasing rate (infant mortality); $\beta = 1$: constant (exponential); $\beta > 1$: increasing (wear-out).
- At $t = \eta$: $F(\eta) = 63.2\%$ for any β .
- Suspensions are units that did not fail; always include them in the analysis.
- MRR and MLE often give slightly different fits; use the AD statistic to choose.
- Weibayes is appropriate for scarce data with prior knowledge of β .
- 3-parameter Weibull handles failure-free periods.
- Multi-plots and contour plots are powerful tools for comparing distributions.

4 Reliability Growth Analysis

4.1 Introduction

Reliability Growth Analysis (RGA) monitors and improves reliability over time by analyzing failure data collected during testing. It identifies trends in the failure rate, confirming whether corrective actions are having the intended effect.

4.2 Learning Objectives

By the end of this chapter, you will be able to:

- Define key reliability growth concepts: Crow-AMSAA and Duane models.
- Use `rdt()` to plan a reliability demonstration test (required test time and sample size).
- Fit reliability growth models using R.
- Apply the Duane model to assess reliability growth on a log-log plot.
- Apply the Crow-AMSAA model and interpret its shape parameter β .
- Use piecewise NHPP models to detect design changes during testing.
- Forecast future failures from a fitted growth model using `predict_rga()` and visualize the result.
- Interpret reliability growth plots and make decisions from the results.

4.3 Reliability Growth Analysis

The `ReliaGrowR` package (Govan 2024) provides functions for reliability growth analysis in R.

```
library(ReliaGrowR)
```

4.4 Test Planning

Before analyzing test data, it helps to answer two design questions: *How long must we test?* and *How many failures should we expect?* ReliaGrowR provides tools for both.

4.4.1 Reliability Demonstration Tests: `rdt()`

A **Reliability Demonstration Test (RDT)** is a pass/fail test designed to demonstrate that a product meets a reliability target at a specified confidence level. `rdt()` calculates the required test time given the sample size (or vice versa).

```
# How long must we test 10 units to demonstrate R(1000 hrs) >= 0.90
# at 80% confidence? Assume Weibull shape beta = 2, zero allowable failures.
plan <- rdt(
  target      = 0.90,    # target reliability
  mission_time = 1000,   # mission time (hours)
  conf_level  = 0.80,   # confidence level
  beta        = 2,      # assumed Weibull shape parameter
  n           = 10      # number of test units
)
print(plan)
```

Reliability Demonstration Test (RDT) Plan

```
-----
Distribution: Weibull
Weibull Shape Parameter (Beta): 2
Allowed Failures (f): 0
Target Reliability: 0.9
Mission Time: 1000
Input Sample Size (n): 10
Required Test Time (T): 1235.94
```

Each unit must be tested for approximately 1,236 hours without failure. If the test passes, we have demonstrated $R(1000) \geq 0.90$ with 80% confidence.

i Try It

Compare the required test time for sample sizes of 5, 10, and 20 units (keep all other parameters the same). What is the trade-off?

```
# Vary n to see the effect on required test time
for (n in c(5, 10, 20)) {
  # plan <- rdt(target = 0.90, mission_time = 1000, conf_level = 0.80,
  #             beta = 2, n = n)
  # print(plan)
}
```

Solution

```
for (n in c(5, 10, 20)) {
  plan <- rdt(target = 0.90, mission_time = 1000, conf_level = 0.80,
             beta = 2, n = n)
  print(plan)
}
```

Reliability Demonstration Test (RDT) Plan

```
-----
Distribution: Weibull
Weibull Shape Parameter (Beta): 2
Allowed Failures (f): 0
Target Reliability: 0.9
Mission Time: 1000
Input Sample Size (n): 5
Required Test Time (T): 1747.89
Reliability Demonstration Test (RDT) Plan
```

```
-----
Distribution: Weibull
Weibull Shape Parameter (Beta): 2
Allowed Failures (f): 0
Target Reliability: 0.9
Mission Time: 1000
Input Sample Size (n): 10
Required Test Time (T): 1235.94
Reliability Demonstration Test (RDT) Plan
```

```
-----
Distribution: Weibull
Weibull Shape Parameter (Beta): 2
Allowed Failures (f): 0
Target Reliability: 0.9
Mission Time: 1000
Input Sample Size (n): 20
```

Required Test Time (T): 873.94

More test units means shorter required test time per unit, but total test effort ($n \times T$) stays roughly constant. Larger fleets reduce calendar time at the cost of more physical units.

4.5 The Duane Model

The **Duane Model** (Duane 1964) is one of the earliest and most widely used models for reliability growth. It is a log-log plot of cumulative MTBF vs cumulative time, where the MTBF is the total operating time divided by the number of failures up to that time.

$$\text{CMTBF}(t) = K \cdot t^{\beta-1}$$

where $K > 0$ is a scale parameter estimated from the data and β is the growth slope. The slope of the line on the log-log plot indicates the rate of reliability growth:

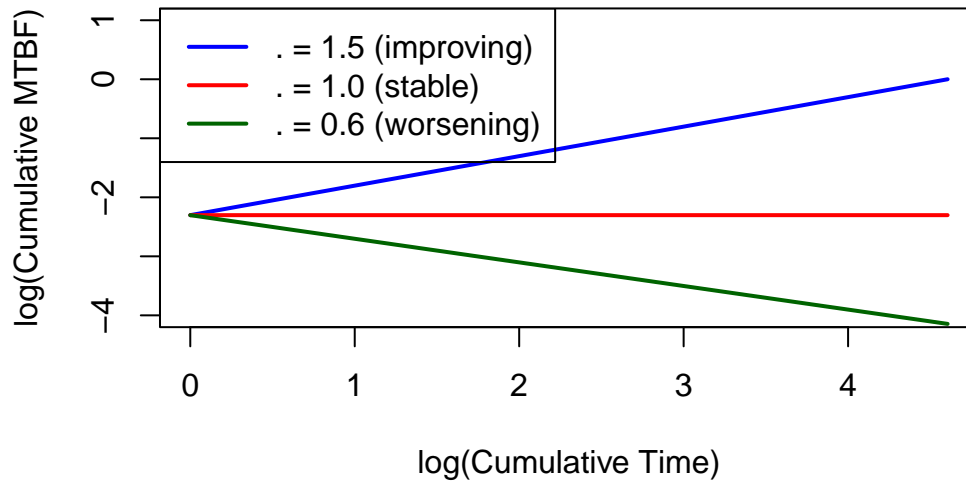
Slope (β)	Meaning
> 1	Reliability improving (failure rate decreasing)
$= 1$	No change (stable)
< 1	Reliability worsening (failure rate increasing)

Plotting the Duane model for three beta values:

```
t <- seq(1, 100, by = 0.1)

# Three scenarios
plot(log(t), log(0.1 * t^(1.5 - 1)), type = "l", col = "blue", lwd = 2,
     xlab = "log(Cumulative Time)", ylab = "log(Cumulative MTBF)",
     main = "Duane Plot: Three Reliability Growth Scenarios",
     ylim = c(-4, 1))
lines(log(t), log(0.1 * t^(1.0 - 1)), col = "red", lwd = 2)
lines(log(t), log(0.1 * t^(0.6 - 1)), col = "darkgreen", lwd = 2)
legend("topleft",
      legend = c(" = 1.5 (improving)", " = 1.0 (stable)", " = 0.6
        ↪ (worsening)"),
      col = c("blue", "red", "darkgreen"), lwd = 2)
```

Duane Plot: Three Reliability Growth Scenarios

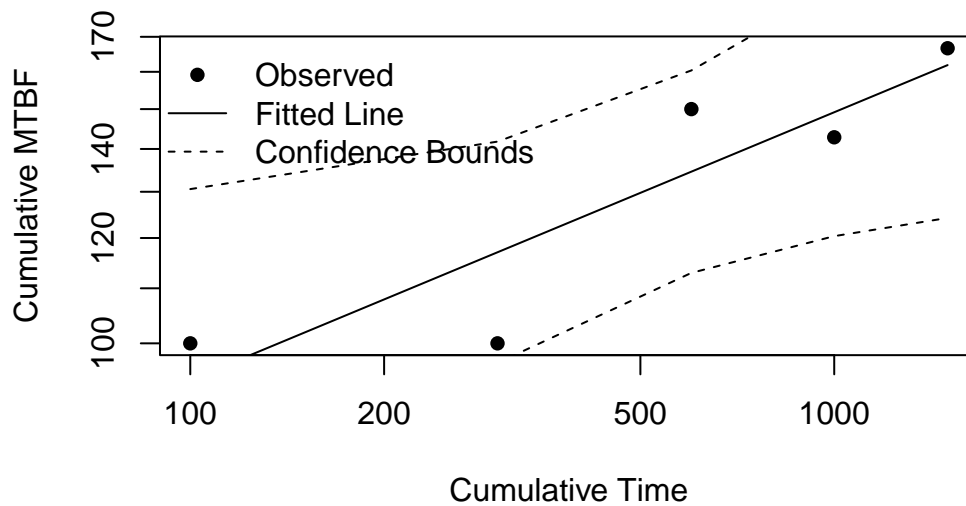


4.5.1 Fitting with ReliaGrowR

```
times    <- c(100, 200, 300, 400, 500)
failures <- c(1, 2, 1, 3, 2)

fit <- duane(times, failures)
plot(fit, main = "Duane Model Example",
     xlab = "Cumulative Time", ylab = "Cumulative MTBF")
```

Duane Model Example



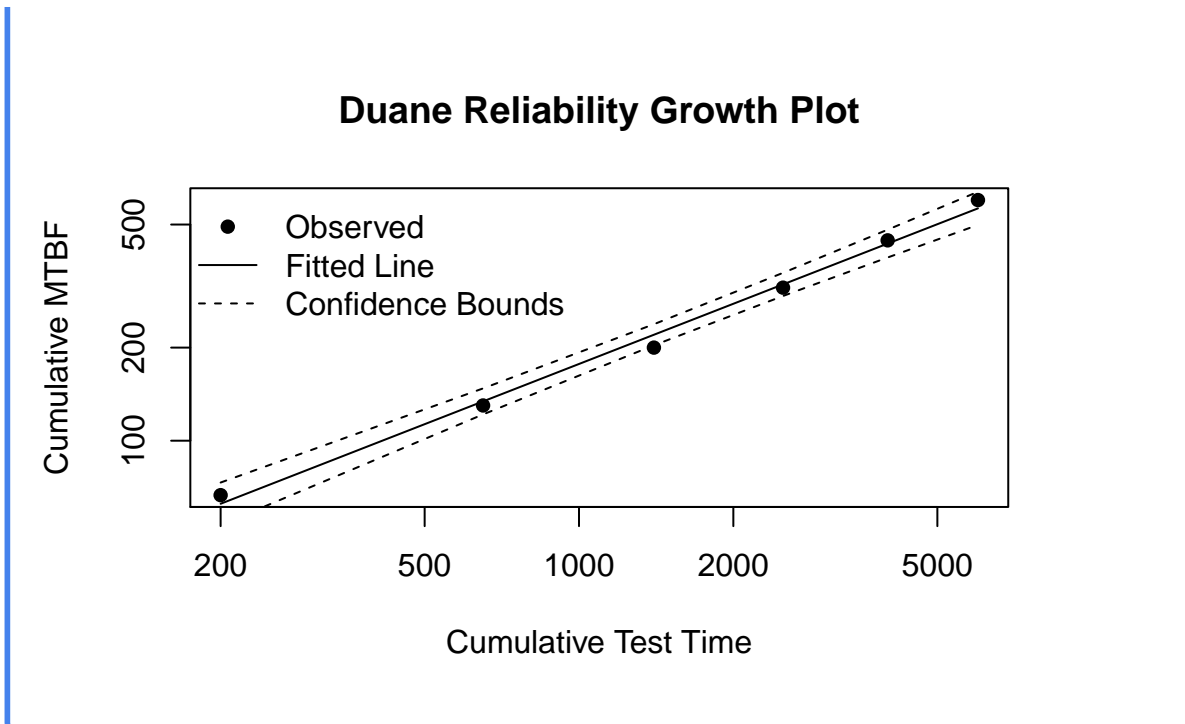
Try It


A new system was tested and the following cumulative failure counts were recorded. Fit a Duane model and plot the result.

```
times    <- c(200, 450, 750, 1100, 1500, 2000)
failures <- c(3, 2, 2, 1, 1, 1)
# fit <- duane(times, failures)
# plot(fit)
```

Solution

```
times    <- c(200, 450, 750, 1100, 1500, 2000)
failures <- c(3, 2, 2, 1, 1, 1)
fit <- duane(times, failures)
plot(fit, main = "Duane Reliability Growth Plot",
     xlab = "Cumulative Test Time", ylab = "Cumulative MTBF")
```



 Review

What does a Duane plot with a slope greater than 1 indicate?
 Answer
 The system's reliability is **improving**: the failure rate is decreasing over time.

4.6 The Crow-AMSAA Model

The **Crow-AMSAA Model** (Crow 1975) models cumulative failures vs cumulative time using a Non-Homogeneous Poisson Process (NHPP):

$$N(t) = \lambda_0 \cdot t^\beta$$

where $\lambda_0 > 0$ is the scale parameter and β is the shape parameter.

The shape parameter β interpretation:

β	Meaning
> 1	Failures increasing (reliability worsening)
$= 1$	Constant rate (stable)

β	Meaning
< 1	Failures decreasing (reliability improving)

⚠ Interpretation is Reversed Between Models

The Crow-AMSAA β means the **opposite** of the Duane β :

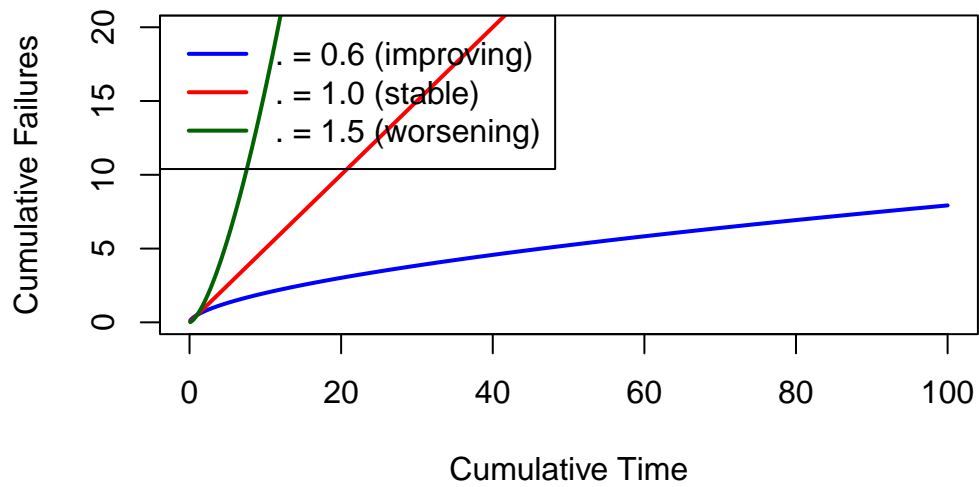
- **Duane:** $\beta > 1$ = reliability *improving* (MTBF is rising).
- **Crow-AMSAA:** $\beta > 1$ = reliability *worsening* (cumulative failures are accelerating).

This reversal occurs because Duane plots MTBF (higher is better), while Crow-AMSAA plots cumulative failures (a lower slope means fewer failures, so *improving*). Always check which model produced your β before interpreting the result.

```
t <- seq(0.1, 100, by = 0.1)

# Three beta scenarios for cumulative failures
plot(t, 0.5 * t^0.6, type = "l", col = "blue", lwd = 2,
      xlab = "Cumulative Time", ylab = "Cumulative Failures",
      main = "Crow-AMSAA: Three Reliability Scenarios",
      ylim = c(0, 20))
lines(t, 0.5 * t^1.0, col = "red", lwd = 2)
lines(t, 0.5 * t^1.5, col = "darkgreen", lwd = 2)
legend("topleft",
      legend = c(" = 0.6 (improving)", " = 1.0 (stable)", " = 1.5
        ↪ (worsening)"),
      col = c("blue", "red", "darkgreen"), lwd = 2)
```

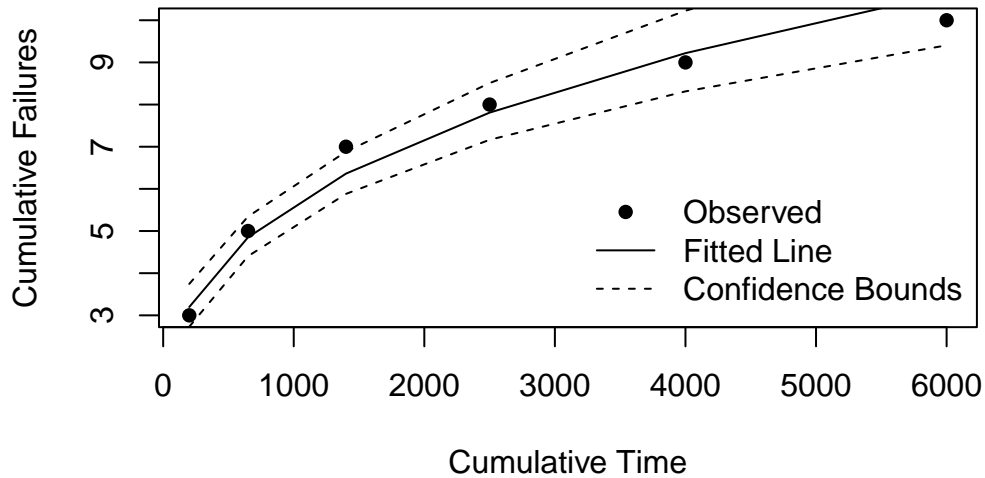
Crow-AMSAA: Three Reliability Scenarios



4.6.1 Fitting with ReliaGrowR

```
result <- rga(times, failures)
plot(result, main = "Crow-AMSAA Model Example",
      xlab = "Cumulative Time", ylab = "Cumulative Failures")
```

Crow-AMSAA Model Example



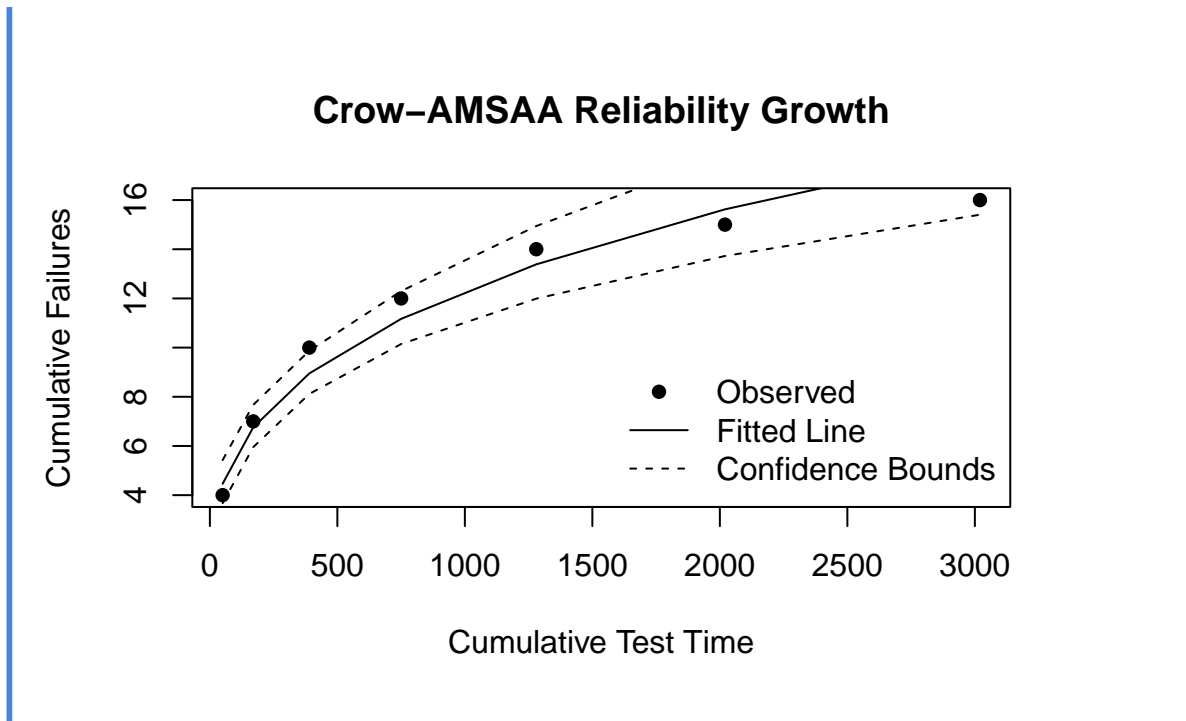
Try It

A development test recorded these cumulative failure counts. Use `rga()` with `model_type = "Crow-AMSAA"` to fit the model.

```
times    <- c(50, 120, 220, 360, 530, 740, 1000)
failures <- c(4, 3, 3, 2, 2, 1, 1)
# fit <- rga(times, failures, model_type = "Crow-AMSAA")
# plot(fit)
```

Solution

```
times    <- c(50, 120, 220, 360, 530, 740, 1000)
failures <- c(4, 3, 3, 2, 2, 1, 1)
fit <- rga(times, failures, model_type = "Crow-AMSAA")
plot(fit, main = "Crow-AMSAA Reliability Growth",
     xlab = "Cumulative Test Time", ylab = "Cumulative Failures")
```



4.6.2 The Piecewise NHPP Model

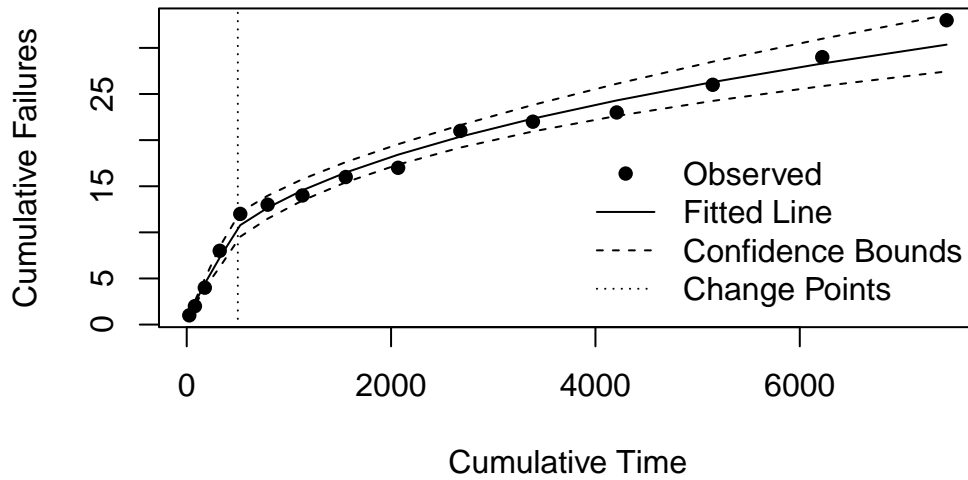
The **Piecewise NHPP** fits a separate Power Law to each segment of the time axis, separated by breakpoints. This is useful when a design change occurs during testing.

$$N_i(t) = N(t_{i-1}) + \lambda_i(t - t_{i-1})^{\beta_i}, \quad t_{i-1} < t \leq t_i$$

```
times    <- c(25, 55, 97, 146, 201, 268, 341, 423, 513, 609, 710, 820, 940,
  ↪ 1072, 1217)
failures <- c(1, 1, 2, 4, 4, 1, 1, 2, 1, 4, 1, 1, 3, 3, 4)
breaks   <- 500

result <- rga(times, failures, model_type = "Piecewise NHPP", breaks =
  ↪ breaks)
plot(result, main = "Piecewise NHPP Model",
  xlab = "Cumulative Time", ylab = "Cumulative Failures")
```

Piecewise NHPP Model



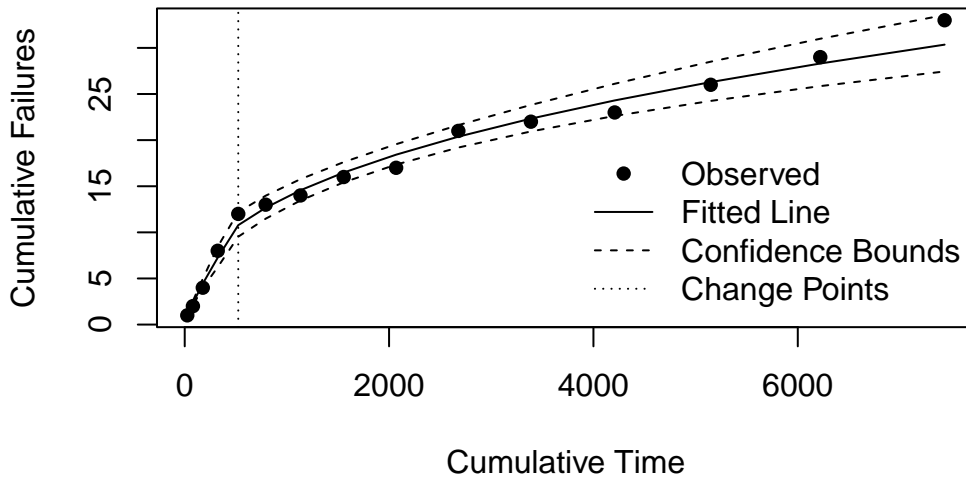
The two segments have different slopes; the first segment (pre-change) has a higher slope than the second (post-change), confirming the design change reduced the failure rate.

4.6.3 Change Point Detection

Without a known breakpoint, `rga()` can detect the change point automatically:

```
result_auto <- rga(times, failures, model_type = "Piecewise NHPP")
plot(result_auto, main = "Piecewise NHPP with Change Point Detection",
      xlab = "Cumulative Time", ylab = "Cumulative Failures")
```

Piecewise NHPP with Change Point Detection



The automatically detected change point is near time 500, consistent with the known design change.

4.7 Forecasting Growth: `predict_rga()`

Once a growth test is underway, `predict_rga()` projects cumulative failures beyond the current endpoint, useful for deciding whether to continue testing or whether a reliability target will be met by a program deadline.

```
times    <- c(100, 200, 300, 400, 500)
failures <- c(1, 2, 1, 3, 2)

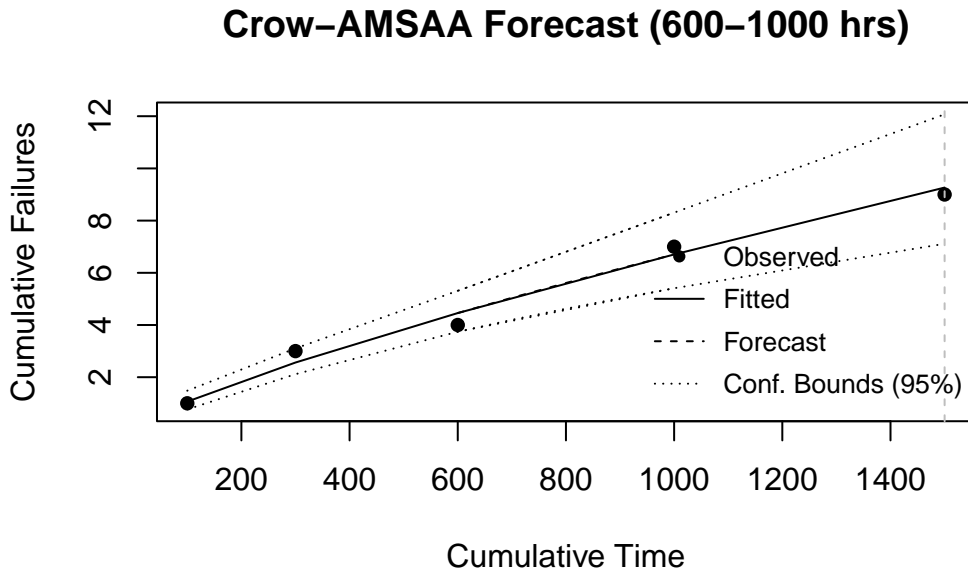
fit <- rga(times, failures)
pred <- predict_rga(fit, times = seq(600, 1000, by = 100))
print(pred)
```

Reliability Growth Forecast (Crow-AMSAA)

Time	Cum.Failures	Lower (95%)	Upper (95%)
600	4.5	3.7	5.3
700	5.0	4.2	6.1

800	5.6	4.6	6.8
900	6.2	5.0	7.6
1000	6.7	5.4	8.3

```
plot(pred, main = "Crow-AMSAA Forecast (600-1000 hrs)",
      xlab = "Cumulative Time", ylab = "Cumulative Failures")
```



The confidence band reflects the uncertainty in the Crow-AMSAA parameter estimates. A narrow band indicates the growth trend is well characterized; a wide band suggests more test time is needed.

💡 Review

The fitted model gives $\hat{\beta} < 1$. Is the system still improving at time 1,000?

Answer

Yes, for the Crow-AMSAA model, $\beta < 1$ means the cumulative failure rate is **decreasing** (the growth curve is concave), so reliability is still improving at time 1,000. Testing should continue until β stabilizes near the growth target.

4.8 Summary

Key takeaways:

- **Duane model:** log-log of MTBF vs time; positive slope ($\beta > 1$) = improving.
- **Crow-AMSAA:** log of cumulative failures vs time; concave downward = improving ($\beta < 1$).
- **Piecewise NHPP:** fits multiple segments; use when a known change occurs during testing.
- **Forecasting:** `predict_rga()` projects cumulative failures with confidence bounds beyond the current test endpoint.
- For accelerated life testing (Arrhenius, Power Law, relationship plots), see Chapter 5.

5 Accelerated Life Testing

5.1 Introduction

Accelerated Life Testing (ALT) (Meeker and Escobar 1998; Nelson 1990) subjects products to elevated stress levels: temperature, voltage, mechanical load, humidity; to induce failures more quickly than would occur under normal operating conditions. The acceleration relationship is then used to translate accelerated-test results into normal-use life predictions.

This approach is essential when products are designed for long service lives (years or decades), making it impractical to wait for failures at use conditions. ALT compresses that time by running tests at higher stress, then mathematically projecting back to the use environment.

5.2 Learning Objectives

By the end of this chapter, you will be able to:

- Explain the purpose of accelerated life testing and the role of the Acceleration Factor.
- Apply the Arrhenius model for temperature-accelerated tests.
- Apply the Power Law model for non-thermal stress acceleration.
- Fit ALT models using the `WeibullR.ALT` package.
- Interpret probability plots and relationship plots from ALT analyses.
- Use parallelization to constrain the shape parameter across stress levels.

5.3 The Acceleration Factor

The **Acceleration Factor (AF)** quantifies how much faster a product fails under elevated stress compared to use conditions:

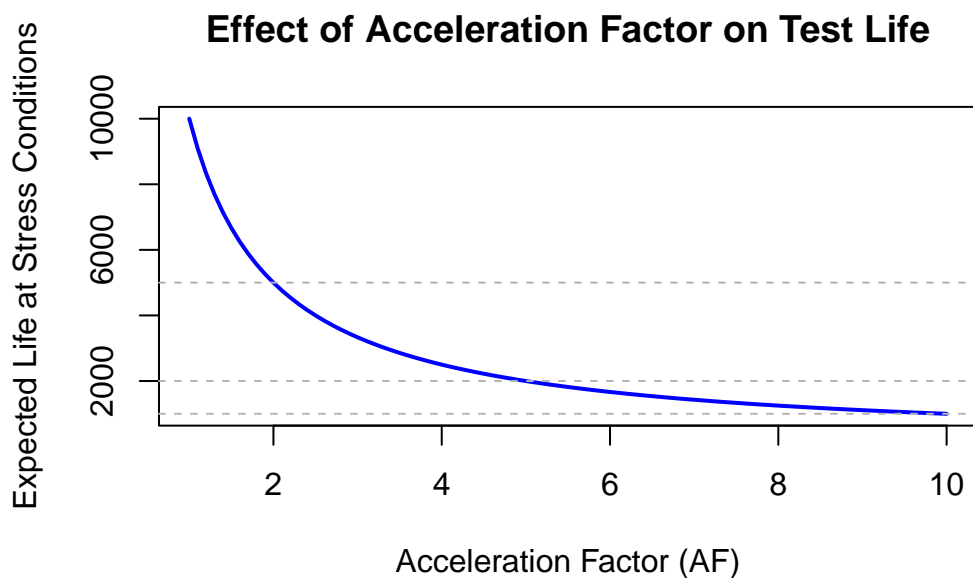
$$\text{Life at use conditions} = AF \times \text{Life at stress conditions}$$

The impact of AF on expected test life:

```
library(WeibullR.ALT)
```

```
AF_vals <- seq(1, 10, by = 0.1)
life_normal <- 10000 # assumed normal-use life

plot(AF_vals, life_normal / AF_vals, type = "l", col = "blue", lwd = 2,
     xlab = "Acceleration Factor (AF)",
     ylab = "Expected Life at Stress Conditions",
     main = "Effect of Acceleration Factor on Test Life")
abline(h = life_normal / c(2, 5, 10), col = "gray70", lty = 2)
```



An AF of 5 means the product fails five times faster under test stress: a 10,000-hour life in the field becomes a 2,000-hour test.

5.4 The Arrhenius Model

The **Arrhenius Model** describes the relationship between temperature and reaction rate, which maps directly to material degradation and failure:

$$AF = \exp\left(\frac{E_a}{k} \left(\frac{1}{T_{\text{use}}} - \frac{1}{T_{\text{stress}}}\right)\right)$$

Where: - E_a = activation energy in eV (electron volts) - k = Boltzmann constant (8.617×10^{-5} eV/K) - T_{use} , T_{stress} = temperatures in Kelvin

Higher activation energies mean the failure mechanism is more sensitive to temperature; a small temperature increase produces a large change in failure rate.

i Choosing Activation Energy

E_a is typically drawn from published literature values for the specific failure mechanism, or estimated from pilot tests at multiple temperatures. Representative ranges:

Failure mechanism	Typical E_a (eV)
Corrosion / electromigration	0.5 – 1.0
Electrical insulation degradation	0.4 – 1.5
Semiconductor oxide breakdown	0.3 – 0.7
Thermal fatigue (solder)	0.2 – 0.5

If E_a is unknown, use a sensitivity analysis across a plausible range before drawing conclusions.

5.4.1 WeibullR.ALT: Arrhenius-Lognormal Example

The `WeibullR.ALT` package (Silkworth 2022) extends `WeibullR` with ALT models. Data: Class B insulation tested at three oven temperatures (170, 190, 220°C), from Nelson (1990) Table 4.1.

```
t41 <- NelsonData("table4.1")
set170 <- alt.data(t41$C170f, s = t41$C170s, stress = 170)
set190 <- alt.data(t41$C190f, s = t41$C190s, stress = 190)
set220 <- alt.data(t41$C220f, s = t41$C220s, stress = 220)

arrobj <- alt.make(list(set170, set190, set220),
                   dist = "lognormal", alt.model = "arrhenius",
                   method.fit = "mle", view_dist_fits = FALSE)
```

The fitted curves shift left as temperature increases; higher temperature leads to shorter life.

Calculating the AF at 220°C vs 170°C with $E_a = 1.0$ eV:

```
E_a <- 1.0
k <- 8.617e-5
```

```
T_use    <- 170 + 273 # 443 K
T_stress <- 220 + 273 # 493 K

AF <- exp(E_a / k * (1 / T_use - 1 / T_stress))
cat("Acceleration Factor at 220°C vs 170°C:", round(AF, 1), "\n")
```

Acceleration Factor at 220°C vs 170°C: 14.3

As printed above, the insulation degrades approximately 14 times faster at 220°C than at 170°C, a substantial acceleration that compresses a multi-year test into a much shorter duration.

i Try It

A component is tested at 85°C to accelerate failures. The use temperature is 25°C. The activation energy is 0.65 eV. Compute the AF.

```
E_a    <- 0.65
k      <- 8.617e-5
T_use  <- 25 + 273.15
T_stress <- 85 + 273.15
# AF = exp((E_a / k) * (1/T_use - 1/T_stress))
```

Solution

```
E_a    <- 0.65
k      <- 8.617e-5
T_use  <- 25 + 273.15
T_stress <- 85 + 273.15
AF <- exp((E_a / k) * (1 / T_use - 1 / T_stress))
cat("Acceleration Factor:", round(AF, 1), "\n")
```

Acceleration Factor: 69.3

5.5 The Power Law Model

The **Power Law Model** describes how a product's life is affected by non-thermal stresses (voltage, mechanical load, pressure):

$$AF = \left(\frac{S_{\text{stress}}}{S_{\text{use}}} \right)^n$$

Where n is the **stress exponent**, the sensitivity of the failure mechanism to the applied stress. Typical values range from 1–3 for low-sensitivity mechanisms to 4–8 for voltage-driven dielectric breakdown. n is estimated from the ALT data by fitting the model at multiple stress levels.

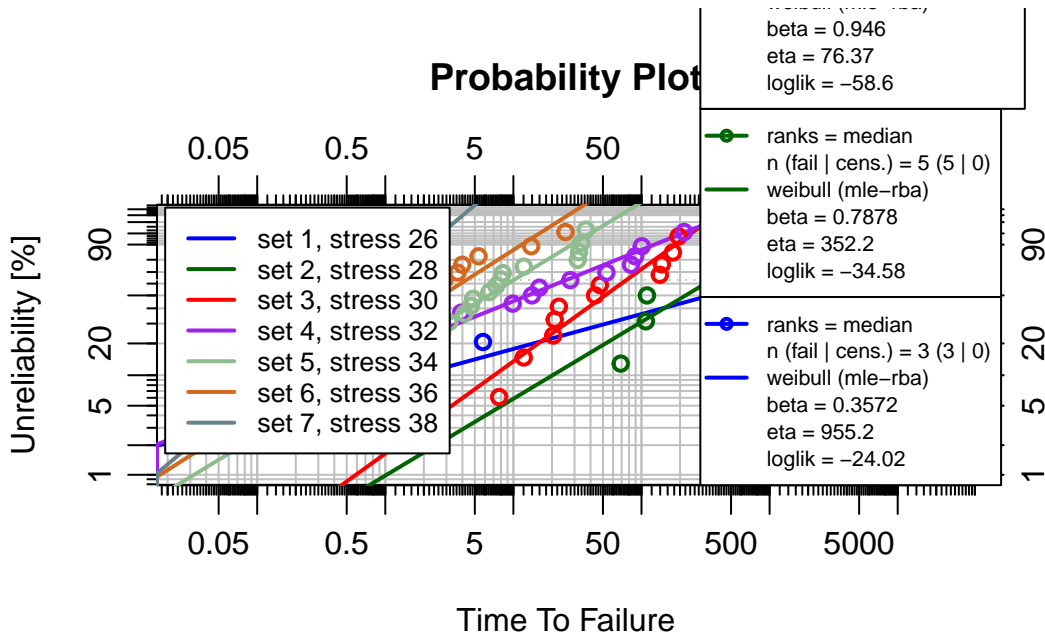
5.5.1 WeibullR.ALT: Power Law Example

Data: electrical insulating fluids tested at 7 voltage levels (26–38 kV).

```
data_nelson <- NelsonData("table3.1")

data26 <- alt.data(data_nelson$kV26, stress = 26)
data28 <- alt.data(data_nelson$kV28, stress = 28)
data30 <- alt.data(data_nelson$kV30, stress = 30)
data32 <- alt.data(data_nelson$kV32, stress = 32)
data34 <- alt.data(data_nelson$kV34, stress = 34)
data36 <- alt.data(data_nelson$kV36, stress = 36)
data38 <- alt.data(data_nelson$kV38, stress = 38)

pwrapj <- alt.make(list(data26, data28, data30, data32, data34, data36,
  ↪ data38),
  dist = "weibull", alt.model = "power")
```



Curves shift left as voltage increases; higher voltage stress leads to shorter life.

5.6 Multiple Stress Factors

The **Eyring Model** extends the Arrhenius approach to account for multiple stress factors (temperature, voltage, humidity):

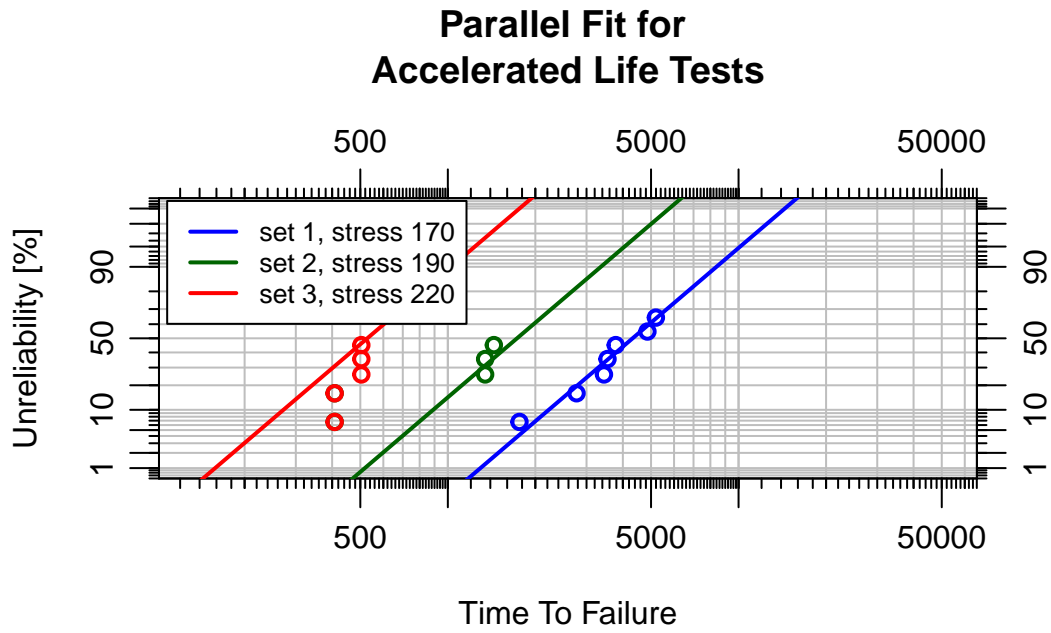
$$\text{Life}(S) = A \cdot \exp\left(\frac{B}{S} + C \cdot S\right)$$

The formula above assumes a single primary stress S . For truly multiple independent stresses (e.g., temperature *and* humidity *and* voltage), additional terms are added to the exponent. The `WeibullR.ALT` package supports the Eyring model via `alt.make(..., alt.model = "eyring")`. See the package documentation for examples.

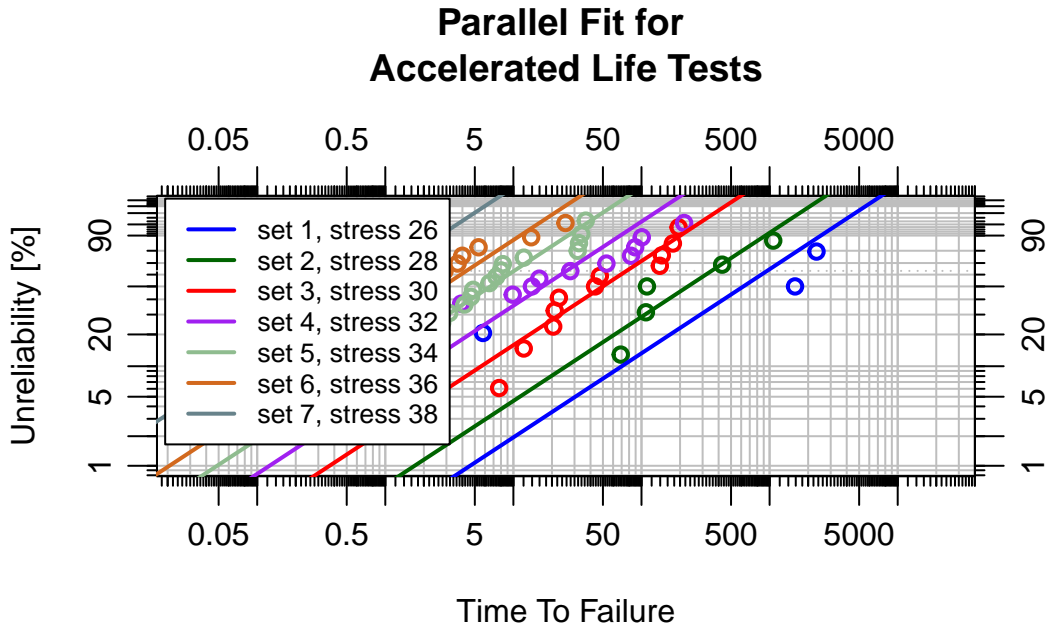
5.7 Parallelization

Parallelization constrains the shape parameter (slope) to be equal across all stress levels, simplifying the analysis:

```
prlobj <- alt.parallel(arobj)
```



```
expobj <- alt.parallel(pwobj)
```



With parallel slopes, the relationship between stress and life can be characterized by a single line (the relationship plot).

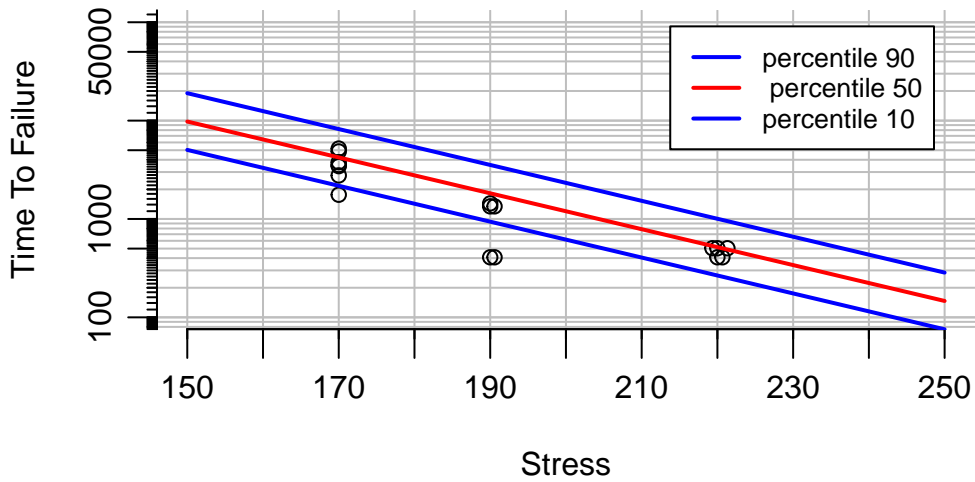
5.8 Relationship Plots

The **relationship plot** shows how life changes as stress increases (log-linear, stress on x-axis, log-life on y-axis). The slope characterizes the sensitivity of life to stress:

- **Negative slope:** higher stress → shorter life (typical).
- **Flat slope:** stress has little effect.
- **Positive slope:** higher stress → longer life (rare).

```
lnrobj <- alt.fit(prlobj)
plot(lnrobj, suppress.dev.new = TRUE)
```

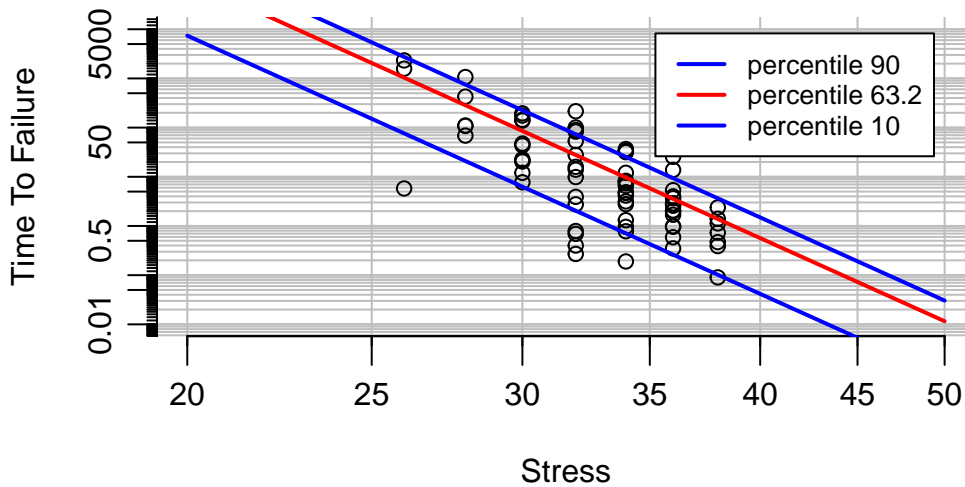
ARRHENIUS – LOGNORMAL RELATIONSHIP PLOT



Each line corresponds to a different probability of failure (e.g., P10 = 10% failure probability, P63.2 = 63.2% failure probability = characteristic life).

```
lnrobj2 <- alt.fit(expobj)
plot(lnrobj2, suppress.dev.new = TRUE)
```

POWER – WEIBULL RELATIONSHIP PLOT



Both relationship plots confirm a negative slope; higher stress leads to shorter life in both datasets.

💡 Review

What does a negative slope in a relationship plot indicate?

Answer

Higher stress leads to **shorter life**. This is the typical finding in ALT; stress accelerates the failure mechanism.

5.9 Important Limitations of ALT

! When ALT Predictions May Be Invalid

ALT is a powerful tool, but it rests on a critical assumption: **the same failure mechanism operates at elevated stress as at use conditions**. If high stress activates a different failure mode, one that would not appear during normal operation, the accelerated results will not transfer to real-world life predictions.

Common pitfalls:

- **New failure modes at high stress**, e.g., thermal cracking from extreme tempera-

ture that would never occur at use temperature.

- **Mechanism change**, e.g., ductile failure at low stress transitions to brittle fracture at high stress; the Weibull shape and the AF model no longer apply.
- **Unknown activation energy or stress exponent**: small errors in E_a or n produce large errors in predicted AF; always perform a sensitivity analysis.

When in doubt: run tests at the *lowest* stress level possible that still produces observable failures in a reasonable time, and verify that failures look the same as field failures.

5.10 Summary

Key takeaways:

- **Acceleration Factor**: $\text{Life}_{\text{use}} = AF \times \text{Life}_{\text{stress}}$.
- **Arrhenius**: temperature-based acceleration; $AF = \exp(E_a/k \cdot (1/T_{\text{use}} - 1/T_{\text{stress}}))$.
- **Power Law**: non-thermal stress acceleration; $AF = (S_{\text{stress}}/S_{\text{use}})^n$.
- **Parallelization**: constrains slope to be equal across stress levels; required before fitting the relationship.
- **Relationship plots**: visualize life vs stress; negative slope = higher stress \rightarrow shorter life.
- Use `alt.data()` \rightarrow `alt.make()` \rightarrow `alt.parallel()` \rightarrow `alt.fit()` for the full workflow.

6 Repairable Systems Analysis

6.1 Introduction

Unlike life data analysis (Chapter 3), which focuses on the *time to first failure* of non-repairable components, **Repairable Systems Analysis** (Ascher and Feingold 1984; Meeker and Escobar 1998) models the entire sequence of recurring failures on a single system or fleet. This chapter covers the foundational theory and the `ReliaGrowR` tools for fitting models, computing fleet-level exposure, and estimating the Mean Cumulative Function.

6.2 Learning Objectives

By the end of this chapter, you will be able to:

- Define the distinction between repairable and non-repairable systems.
- Explain the Homogeneous Poisson Process (HPP) and Non-Homogeneous Poisson Process (NHPP).
- Describe the Power Law Process and its parameters.
- Compute fleet-level exposure using `exposure()`.
- Estimate the Mean Cumulative Function (MCF) using `mcf()`.
- Fit and interpret the Crow-AMSAA model using `nhpp()` from `ReliaGrowR`.
- Apply the Piecewise NHPP model to detect changes in failure behavior.
- Forecast future failure counts from a fitted Power Law model.

The `ReliaGrowR` package (Govan 2024) provides functions for repairable systems analysis in R.

```
library(ReliaGrowR)
```

6.3 Repairable vs Non-Repairable Systems

A **non-repairable system** is discarded after failure and replaced with a new, statistically identical unit. The unit of analysis is the *time to first failure*. Weibull analysis applies.

A **repairable system** is restored to service after each failure and continues accumulating operating time. The unit of analysis is the *sequence of inter-failure times* or cumulative failure counts over the system's life.

Two repair assumptions are common:

Repair type	Description	Model
Perfect repair	As-good-as-new after each repair	Renewal process
Minimal repair	As-bad-as-old, failure fixed, nothing else changed	NHPP

Most field maintenance data best fits the **minimal repair** assumption, making NHPP the standard framework.

The **recurrence rate** (intensity function) $\rho(t)$ is the instantaneous rate of failure occurrence at time t :

$$\rho(t) = \frac{d E[N(t)]}{dt}$$

6.4 Failure Processes: HPP and NHPP

6.4.1 Homogeneous Poisson Process (HPP)

The **HPP** assumes a *constant* failure rate λ throughout the system's life:

$$E[N(t)] = \lambda t \quad \rho(t) = \lambda \quad \text{MTBF} = 1/\lambda$$

The HPP is appropriate when failures occur at random with no trend.

6.4.2 Non-Homogeneous Poisson Process (NHPP)

The **NHPP** allows the intensity function to change with time:

$$\rho(t) = \frac{d\Lambda(t)}{dt}$$

When $\rho(t)$ increases, the system is deteriorating; when it decreases, the system is improving.

```

t <- seq(1, 100, by = 1)

# HPP (constant rate)
hpp_intensity <- rep(0.05, length(t))

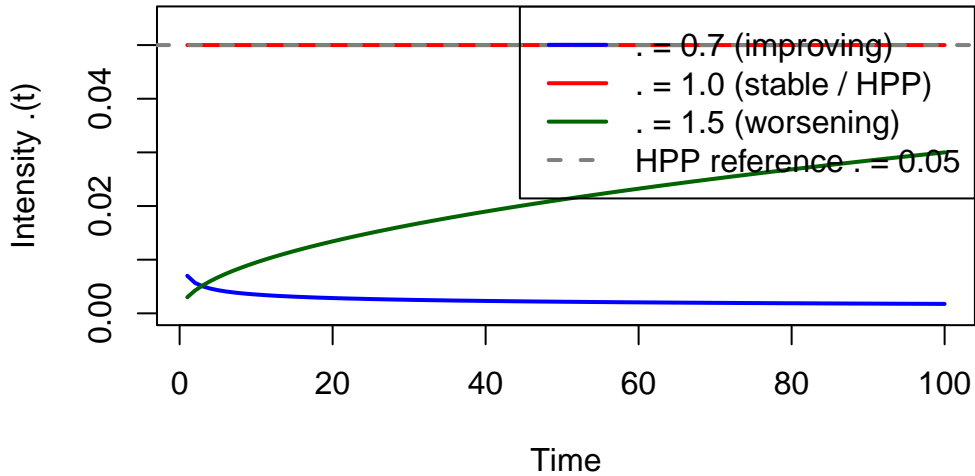
# NHPP Power Law scenarios
nhpp_improving <- 0.01 * 0.7 * t^(0.7 - 1) # beta = 0.7
nhpp_stable <- 0.05 * 1.0 * t^(1.0 - 1) # beta = 1.0
nhpp_worsening <- 0.002 * 1.5 * t^(1.5 - 1) # beta = 1.5

ylim_max <- max(c(hpp_intensity, nhpp_improving, nhpp_stable,
  ↪ nhpp_worsening)) * 1.1

plot(t, nhpp_improving, type = "l", col = "blue", lwd = 2,
      ylim = c(0, ylim_max),
      xlab = "Time", ylab = "Intensity (t)",
      main = "NHPP Power Law Intensity Functions")
lines(t, nhpp_stable, col = "red", lwd = 2)
lines(t, nhpp_worsening, col = "darkgreen", lwd = 2)
abline(h = 0.05, col = "gray50", lwd = 2, lty = 2)
legend("topright",
      legend = c(" = 0.7 (improving)", " = 1.0 (stable / HPP)",
        " = 1.5 (worsening)", "HPP reference = 0.05"),
      col = c("blue", "red", "darkgreen", "gray50"),
      lwd = 2, lty = c(1, 1, 1, 2))

```

NHPP Power Law Intensity Functions



6.5 The Power Law Process

The **Power Law Process** is the most widely used NHPP for repairable systems:

$$E[N(t)] = \lambda t^\beta \quad \rho(t) = \lambda \beta t^{\beta-1}$$

Parameter interpretation:

- $\lambda > 0$, scale parameter; overall magnitude of the failure rate.
- $\beta < 1$: decreasing intensity (system improving).
- $\beta = 1$: constant intensity = HPP.
- $\beta > 1$: increasing intensity (system deteriorating).

The **instantaneous MTBF** at time t :

$$\text{MTBF}(t) = \frac{1}{\rho(t)} = \frac{1}{\lambda \beta t^{\beta-1}}$$

Maximum likelihood estimates given n failures at times $t_1 < \dots < t_n$ over total time T :

$$\hat{\beta} = \frac{n}{\sum_{i=1}^n \ln(T/t_i)} \quad \hat{\lambda} = \frac{n}{T^{\hat{\beta}}}$$

💡 Review

A repairable system is observed to have an increasing failure rate over time. Which model is most appropriate?

Answer

NHPP Power Law with $\beta > 1$. An increasing intensity function is characteristic of a deteriorating system. HPP (constant rate) cannot capture this trend.

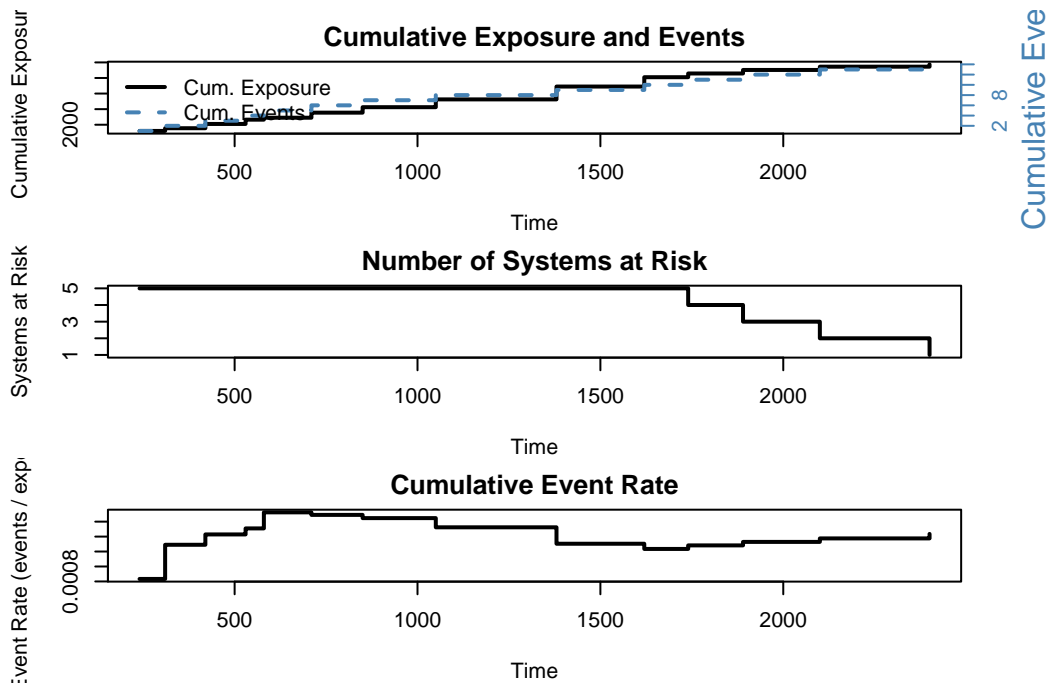
6.6 Exposure Analysis

When analyzing a *fleet*, individual units enter and leave observation at different times. **Exposure** is the total operating time accumulated across all units currently at risk.

The `exposure()` function computes: - **Cumulative exposure**: total system-time at risk up to each event time. - **Number at risk**: how many units were under observation at each time. - **Event rate**: cumulative events divided by cumulative exposure.

```
pump_data <- data.frame(
  id    = c(1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 4, 5, 5),
  time  = c(310, 850, 1620, 420, 1050, 2100, 580, 1890, 240, 710, 1380,
    ↪ 2400, 530, 1740),
  event = c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
)

exp_result <- exposure(id = pump_data$id, time = pump_data$time,
  event = pump_data$event)
plot(exp_result)
```



`exp_result` is a data frame containing three columns: `cum_exposure` (total system-time accumulated across all units at risk), `n_risk` (number of units under observation at each time), and `event_rate` (cumulative events divided by cumulative exposure). The exposure plot shows how total observed system-time accumulates and how the fleet-wide event rate behaves over time.

💡 Review

Why is it important to account for exposure when comparing failure rates across systems with different observation periods?

Answer

Without normalizing by exposure, a system observed for twice as long will appear to have twice as many failures even if its underlying rate is identical. Exposure provides the denominator, failures per unit time, so rates can be compared fairly.

6.7 Mean Cumulative Function

The **Mean Cumulative Function (MCF)** is a non-parametric estimate of the expected cumulative number of failures *per system* as a function of time. It makes no distributional assumptions and handles systems with different observation periods.

The **Nelson-Aalen estimator** is a non-parametric method that requires no distributional

assumption and naturally handles systems with different observation lengths (administrative censoring):

$$\hat{M}(t) = \sum_{t_j \leq t} \frac{d_j}{n_j}$$

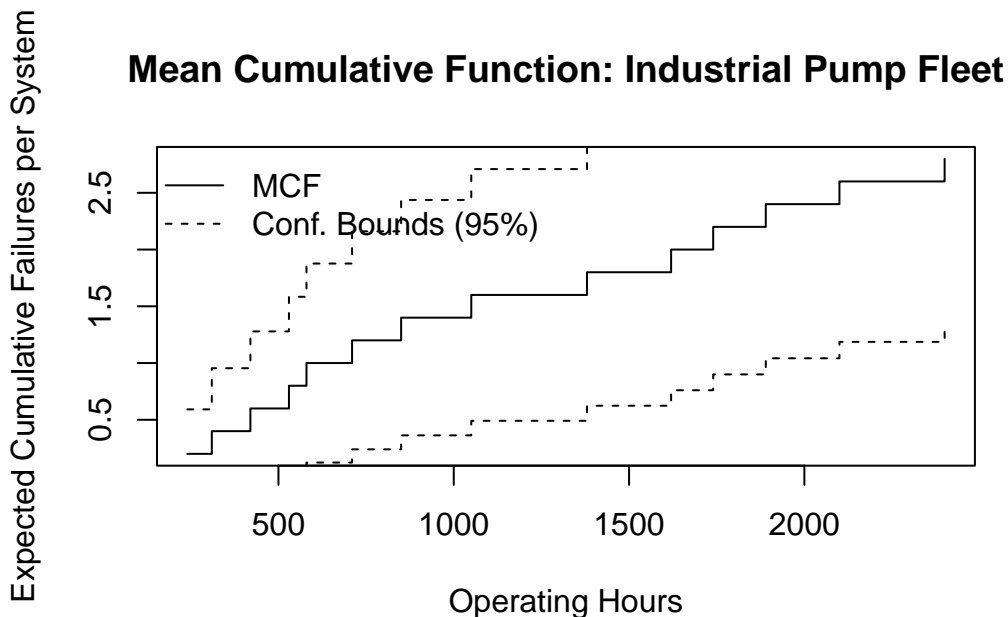
where d_j is the number of failures at time t_j and n_j is the number of systems at risk at t_j .

Pass `end_time` to `mcf()` when systems were observed beyond their last recorded failure; otherwise the risk set is deflated too early, inflating the rate estimate.

```
end_times <- c("1" = 3000, "2" = 3000, "3" = 3000, "4" = 3000, "5" = 3000)

mcf_result <- mcf(id = pump_data$id, time = pump_data$time,
                 event = pump_data$event,
                 end_time = end_times)

plot(mcf_result,
     main = "Mean Cumulative Function: Industrial Pump Fleet",
     xlab = "Operating Hours",
     ylab = "Expected Cumulative Failures per System")
```



The slope of the MCF reveals the recurrence trend: steepening = increasing failure rate; flattening = improving.

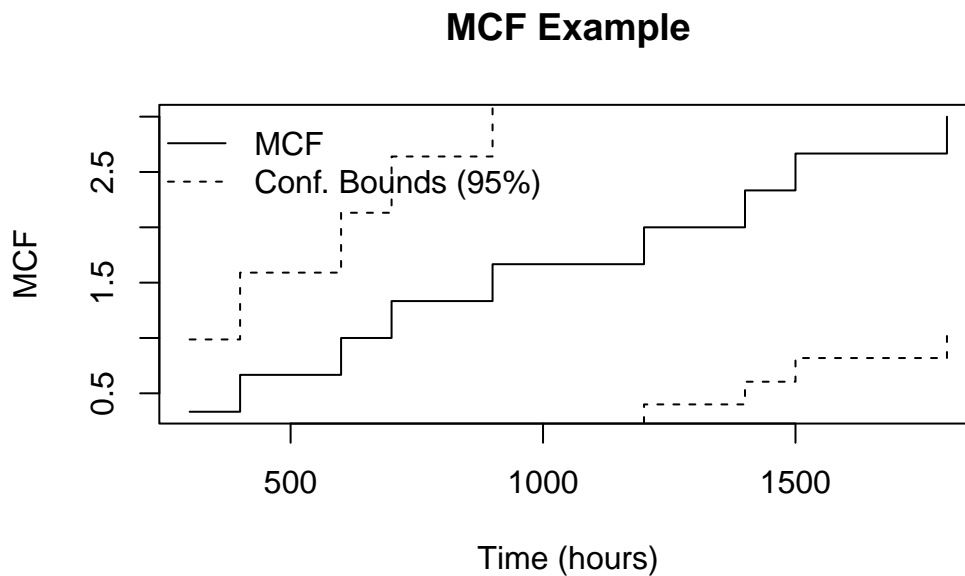
i Try It

Three systems were observed with the following failure times. Each was observed until hour 2000. Compute and plot the MCF.

```
id    <- c(rep("1", 3), rep("2", 4), rep("3", 2))
time  <- c(400, 900, 1500, 300, 700, 1200, 1800, 600, 1400)
event <- rep(1, 9)
end_times2 <- c("1" = 2000, "2" = 2000, "3" = 2000)
# fit <- mcf(id, time, event, end_time = end_times2)
# plot(fit)
```

Solution

```
id    <- c(rep("1", 3), rep("2", 4), rep("3", 2))
time  <- c(400, 900, 1500, 300, 700, 1200, 1800, 600, 1400)
event <- rep(1, 9)
end_times2 <- c("1" = 2000, "2" = 2000, "3" = 2000)
fit <- mcf(id = id, time = time, event = event, end_time = end_times2)
plot(fit, main = "MCF Example", xlab = "Time (hours)", ylab = "MCF")
```



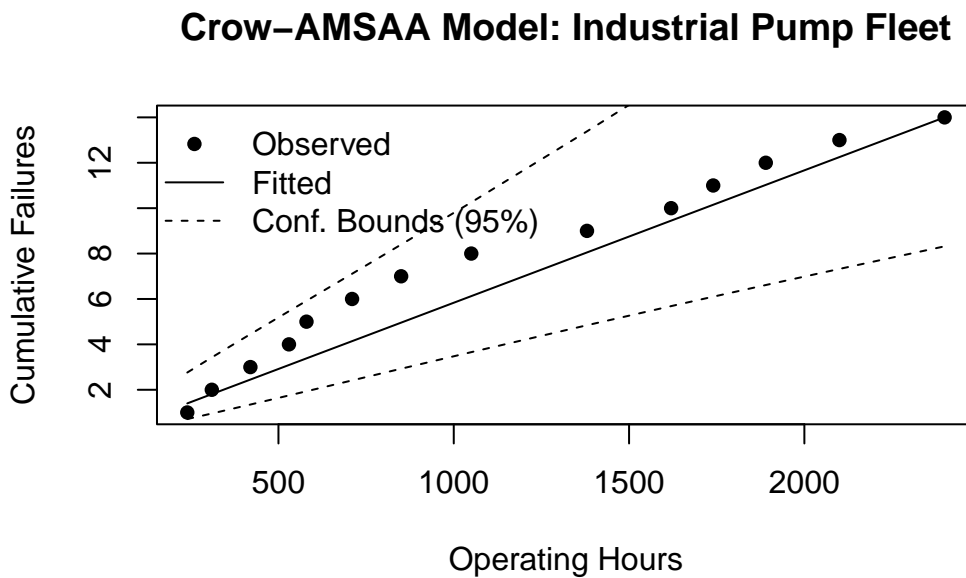
6.8 NHPP Analysis with nhpp()

The `nhpp()` function fits the Power Law Process (Crow-AMSAA model) (Crow 1975) to individual failure event data by maximum likelihood, returning estimates of λ and β .

A curve that bends downward (concave) indicates $\hat{\beta} < 1$, an improving system. Upward bending (convex) indicates $\hat{\beta} > 1$, a deteriorating system.

```
# pump_data defined above in the Exposure section
failure_times <- sort(pump_data$time[pump_data$event == 1])

fit <- nhpp(time = failure_times, event = rep(1, length(failure_times)))
plot(fit,
     main = "Crow-AMSAA Model: Industrial Pump Fleet",
     xlab = "Operating Hours",
     ylab = "Cumulative Failures")
```



The concave curve and $\hat{\beta} < 1$ indicate the pump fleet's failure rate is decreasing, a favorable operating regime.

i Try It

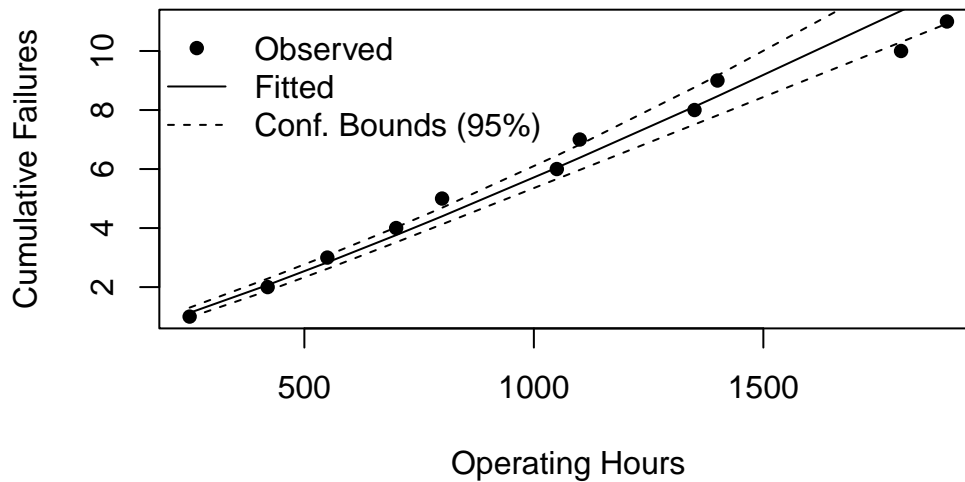
A compressor fleet recorded individual failure times across five systems. Fit a Power Law model using `nhpp()` with `method = "LS"`.

```
comp_times <- c(250, 800, 1400, 550, 1100, 1900, 700, 1350, 420, 1050,  
  ↪ 1800)  
# fit <- nhpp(time = comp_times, event = rep(1, length(comp_times)),  
  ↪ method = "LS")  
# plot(fit)
```

Solution

```
comp_times <- c(250, 800, 1400, 550, 1100, 1900, 700, 1350, 420, 1050,  
  ↪ 1800)  
fit <- nhpp(time = sort(comp_times),  
  event = rep(1, length(comp_times)), method = "LS")  
plot(fit, main = "Crow-AMSAA: Compressor Fleet",  
  xlab = "Operating Hours", ylab = "Cumulative Failures")
```

Crow-AMSAA: Compressor Fleet



6.9 Piecewise NHPP

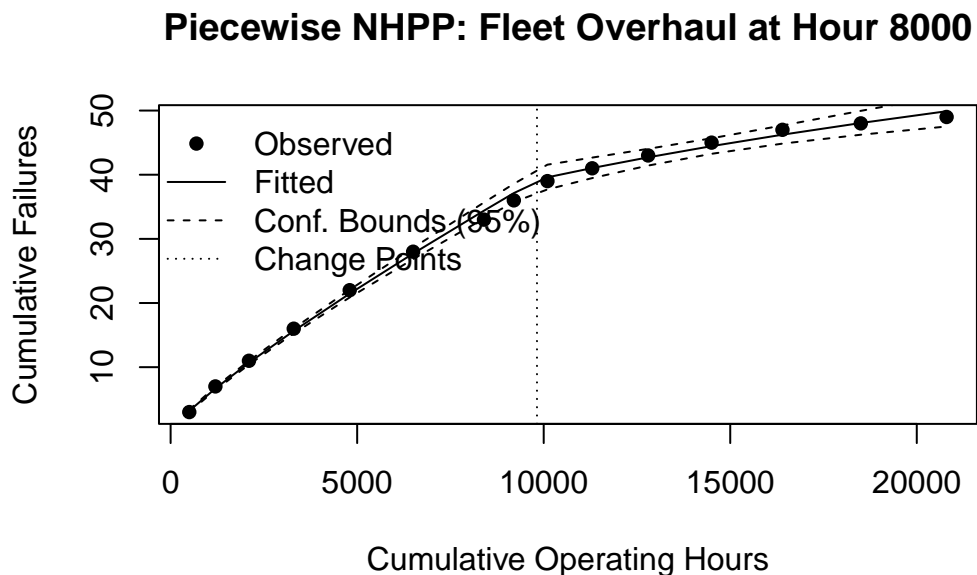
The **Piecewise NHPP** (Guo et al. 2010) fits a separate Power Law to each segment of the time axis, allowing for structural changes in failure behavior (overhauls, design modifications, operational changes):

$$E[N(t)] = \begin{cases} \lambda_1 t^{\beta_1} & 0 < t \leq \tau \\ E[N(\tau)] + \lambda_2 (t - \tau)^{\beta_2} & t > \tau \end{cases}$$

Pass a breakpoint estimate via the `breaks` argument. The `segmented` algorithm then optimizes the breakpoint location from the data.

```
times2 <- c(500, 1200, 2100, 3300, 4800, 6500, 8400, 9200, 10100, 11300,
           12800, 14500, 16400, 18500, 20800)
failures2 <- c(3, 4, 4, 5, 6, 6, 5, 3, 3, 2, 2, 2, 2, 1, 1)

fit_pw <- nhpp(time = times2, event = failures2, breaks = 8000, method =
  ↪ "LS")
plot(fit_pw,
     main = "Piecewise NHPP: Fleet Overhaul at Hour 8000",
     xlab = "Cumulative Operating Hours",
     ylab = "Cumulative Failures")
```



The first segment (pre-overhaul, steeper slope) gives way to a flatter or downward-bending second segment, confirming the overhaul reduced the failure rate. The fitted breakpoint may differ slightly from the input value; here it is estimated near hour 9,820.

 Review

What is the primary reason to use a Piecewise NHPP instead of a single Crow-AMSAA model?

Answer

To account for a **known or suspected structural change** in the failure process (overhaul, design change, operational change). A piecewise model fits each segment independently, capturing the change in failure behavior.

6.10 Interpreting Results and Predicting Future Failures

6.10.1 Reading $\hat{\beta}$

$\hat{\beta}$	Interpretation	Typical cause
< 1	Decreasing failure rate	Effective maintenance, infant mortality resolved
$= 1$	Constant failure rate	Random external shocks
> 1	Increasing failure rate	Wear-out, accumulating damage, aging

6.10.2 Cumulative vs Instantaneous MTBF

- **Cumulative MTBF** = $T/N(T)$, the historical average over all observed failures.
- **Instantaneous MTBF** at time $t = 1/\rho(t) = 1/(\hat{\lambda}\hat{\beta}t^{\hat{\beta}-1})$, the expected time to the *next* failure.

For $\hat{\beta} > 1$ (aging system), the instantaneous MTBF falls *below* the cumulative average at late times; the system's near-term reliability is worse than its history suggests.

6.10.3 Predicting Future Failures

Expected additional failures from current time T to a future time $T + \Delta t$:

$$E[\Delta N] = \hat{\lambda}(T + \Delta t)^{\hat{\beta}} - \hat{\lambda}T^{\hat{\beta}}$$

```

lambda    <- 0.0003
beta      <- 0.85
T_current <- 15000
T_future  <- 17000

expected_add <- lambda * T_future^beta - lambda * T_current^beta
cat("Expected additional failures:", round(expected_add, 2), "\n")

```

Expected additional failures: 0.12

Try It

A system has been operating for 5,000 hours. A Power Law fit yielded $\lambda = 0.0005$ and $\beta = 1.2$. Calculate the expected additional failures in the next 1,000 hours.

```

lambda    <- 0.0005
beta      <- 1.2
T_current <- 5000
T_future  <- 6000
# E[ΔN] = lambda * T_future^beta - lambda * T_current^beta

```

Solution

```

lambda    <- 0.0005
beta      <- 1.2
T_current <- 5000
T_future  <- 6000
expected_failures <- lambda * T_future^beta - lambda * T_current^beta
cat("Expected additional failures:", round(expected_failures, 2), "\n")

```

Expected additional failures: 3.36

Review

A system has a fitted Power Law model with $\beta = 1.35$. What is the operational implication?

Answer

The failure rate is **increasing**; wear-out or aging is occurring. The system may be approaching a major overhaul threshold. The instantaneous MTBF is below the cumulative historical average.

6.11 Summary

Key takeaways:

- Repairable systems analysis models the *sequence* of recurring failures, not just the first failure.
- The **Power Law Process** ($E[N(t)] = \lambda t^\beta$) is the standard NHPP for repairable systems.
- $\hat{\beta}$ determines whether the failure rate is improving (< 1), stable ($= 1$), or worsening (> 1).
- `exposure()` computes fleet-level operating time at risk.
- `mcf()` estimates cumulative failures per system non-parametrically (no distributional assumption).
- `nhpp()` fits the Crow-AMSAA model from individual failure times; add `breaks` for piecewise fitting.
- Future failure counts: $E[\Delta N] = \hat{\lambda}(T + \Delta t)^{\hat{\beta}} - \hat{\lambda}T^{\hat{\beta}}$.

7 Interactive Plots with ReliaPlotR

i HTML-Only Content

This chapter produces interactive **plotly** charts that require a web browser. In this PDF/EPUB version the code is shown without output. View the [online version](#) for the live interactive plots.

7.1 Introduction

The previous chapters produce static base-R plots. **ReliaPlotR** (Govan 2023a) is a companion package that turns those same fitted model objects into *interactive* plotly charts, with hover tooltips, zoom/pan controls, and built-in download buttons, without requiring you to change how you fit your models.

The API follows a single convention: pass the fitted object from **WeibullR**, **ReliaGrowR**, or **WeibullR.ALT** directly to the corresponding **plotly_*** function. The complete function map:

Function	Input object	Source package	Chapter
<code>plotly_wblr()</code>	<code>wblr</code>	WeibullR	Chapter 3
<code>plotly_contour()</code>	<code>wblr</code> (with <code>lrb</code> bounds)	WeibullR	Chapter 3
<code>plotly_duane()</code>	<code>duane</code>	ReliaGrowR	Chapter 4
<code>plotly_rga()</code>	<code>rga</code>	ReliaGrowR	Chapter 4
<code>plotly_alt()</code>	<code>alt.parallel</code>	WeibullR.ALT	Chapter 5
<code>plotly_rel()</code>	<code>alt.fit</code>	WeibullR.ALT	Chapter 5
<code>plotly_nhpp()</code>	<code>nhpp</code>	ReliaGrowR	Chapter 6
<code>plotly_mcf()</code>	<code>mcf</code>	ReliaGrowR	Chapter 6
<code>plotly_exposure()</code>	<code>exposure</code>	ReliaGrowR	Chapter 6

7.2 Installation

```
install.packages("ReliaPlotR")
```

```
library(ReliaPlotR)
library(WeibullR)
library(ReliaGrowR)
library(WeibullR.ALT)
```

ReliaPlotR is available on CRAN: <https://CRAN.R-project.org/package=ReliaPlotR>

7.3 Life Data Analysis Plots

These functions accept the `wblr` objects produced by `WeibullR`, exactly as shown in Chapter 3. No reformatting needed.

7.3.1 Weibull Probability Plot: `plotly_wblr()`

Pass any fitted `wblr` object (or a list of objects for overlays):

```
failures <- c(30, 49, 82, 90, 96)
obj <- wblr.conf(wblr.fit(wblr(failures)))
plotly_wblr(obj, fitCol = "blue", confCol = "blue",
            main = "Weibull Probability Plot")
```

Hover over any point to read off its exact time and cumulative probability. The results table (, , Anderson-Darling) appears in the lower right.

Including suspensions:

```
suspensions <- c(100, 45, 10)
obj_susp <- wblr.conf(wblr.fit(wblr(failures, suspensions)))
plotly_wblr(obj_susp, susp = suspensions,
            fitCol = "red", confCol = "red",
            main = "Weibull Plot with Suspensions")
```

To overlay two models, pass a list:

```
obj1 <- wblr.conf(wblr.fit(wblr(failures, col = "red")))
obj2 <- wblr.conf(wblr.fit(wblr(failures, suspensions, col = "purple")))
plotly_wblr(list(obj1, obj2),
            cols = c("red", "purple"),
            main = "Multi-Model Weibull Overlay")
```

7.3.2 Contour Plot: plotly_contour()

Pass a `wblr` object fitted with likelihood-ratio bounds (`method.conf = "lrb"`):

```
obj_lrb <- wblr.conf(wblr.fit(wblr(failures, col = "blue"), method.fit =
  ↪ "mle"),
                  method.conf = "lrb")
plotly_contour(obj_lrb,
              main = "Weibull Contour Plot (90% CL)",
              xlab = "Eta ( )", ylab = "Beta ( )")
```

Hover to read off any (,) combination inside the confidence region. Overlaying multiple contours:

```
set.seed(123)
data <- data.frame(
  time = c(rweibull(5, 0.5, 20), rweibull(10, 1, 10), rweibull(5, 2, 5),
           rweibull(100, 2, 10)),
  event = c(rep(1, 20), rep(0, 100)),
  failure_mode = c(rep("A", 5), rep("B", 10), rep("C", 5), rep("", 100))
)
dat1 <- dat2 <- dat3 <- data
dat1$event[dat1$failure_mode != "A"] <- 0
dat2$event[dat2$failure_mode != "B"] <- 0
dat3$event[dat3$failure_mode != "C"] <- 0

c1 <- wblr.conf(wblr.fit(wblr(dat1, col = "blue"), method.fit = "mle"),
  ↪ method.conf = "lrb")
c2 <- wblr.conf(wblr.fit(wblr(dat2, col = "red"), method.fit = "mle"),
  ↪ method.conf = "lrb")
c3 <- wblr.conf(wblr.fit(wblr(dat3, col = "orange"), method.fit = "mle"),
  ↪ method.conf = "lrb")

plotly_contour(list(c1, c2, c3),
```

```
cols = c("blue", "red", "orange"),
main = "Contour Comparison: Three Failure Modes")
```

7.4 Reliability Growth Plots

These functions accept fitted objects from `ReliaGrowR`, the same objects plotted in Chapter 4.

7.4.1 Duane Plot: `plotly_duane()`

```
times    <- c(100, 200, 300, 400, 500)
failures <- c(1, 2, 1, 3, 2)

fit_duane <- duane(times, failures)
plotly_duane(fit_duane,
             main = "Duane Reliability Growth",
             xlab = "Cumulative Time",
             ylab = "Cumulative MTBF")
```

7.4.2 Crow-AMSAA Plot: `plotly_rga()`

```
fit_rga <- rga(times, failures)
plotly_rga(fit_rga,
           main = "Crow-AMSAA Model",
           xlab = "Cumulative Time",
           ylab = "Cumulative Failures")
```

Hover over any data point to read the exact cumulative time and failure count. The fitted curve and confidence bounds can be toggled via the legend.

7.5 Accelerated Life Testing Plots

These functions accept `alt.make()` and `alt.fit()` objects from `WeibullR.ALT`; see Chapter 5.

7.5.1 ALT Probability Plot: `plotly_alt()`

```
t41 <- NelsonData("table4.1")
set170 <- alt.data(t41$C170f, s = t41$C170s, stress = 170)
set190 <- alt.data(t41$C190f, s = t41$C190s, stress = 190)
set220 <- alt.data(t41$C220f, s = t41$C220s, stress = 220)

arrobj <- alt.make(list(set170, set190, set220),
                  dist = "lognormal", alt.model = "arrhenius",
                  method.fit = "mle-rba")
prlobj <- alt.parallel(arrobj, view_parallel_fits = FALSE)

plotly_alt(prlobj,
           main = "Arrhenius-Lognormal ALT Probability Plot",
           xlab = "Time to Failure (hours)",
           ylab = "Probability of Failure")
```

Each color represents a different temperature level. Hover to inspect unreliability at any time. Note: `plotly_alt()` requires a parallelized object from `alt.parallel()`.

7.5.2 Life-Stress Relationship Plot: `plotly_rel()`

```
lnrobj <- alt.fit(prlobj)
plotly_rel(lnrobj,
           main = "Life-Stress Relationship: Arrhenius Model",
           xlab = "Temperature (°C)",
           ylab = "Time to Failure")
```

Each line represents a percentile of the failure distribution (P10, P63.2, P90, etc.). Hover over any point to read the expected life at that stress level and percentile.

7.5.3 Power Law–Weibull ALT Plots

The Power Law model applies to non-thermal stress (voltage, load, pressure). Here, electrical insulating fluids are tested at seven voltage levels (Nelson Table 3.1; see Chapter 5):

```

data_nelson <- NelsonData("table3.1")
data26 <- alt.data(data_nelson$kV26, stress = 26)
data28 <- alt.data(data_nelson$kV28, stress = 28)
data30 <- alt.data(data_nelson$kV30, stress = 30)
data32 <- alt.data(data_nelson$kV32, stress = 32)
data34 <- alt.data(data_nelson$kV34, stress = 34)
data36 <- alt.data(data_nelson$kV36, stress = 36)
data38 <- alt.data(data_nelson$kV38, stress = 38)

pwrojb <- alt.make(list(data26, data28, data30, data32, data34, data36,
  ↪ data38),
                  dist = "weibull", alt.model = "power")
expobj <- alt.parallel(pwrojb, view_parallel_fits = FALSE)

plotly_alt(expobj,
            main = "Power Law-Weibull ALT Probability Plot",
            xlab = "Time to Failure",
            ylab = "Probability of Failure")

```

```

pwrrojb <- alt.fit(expobj)
plotly_rel(pwrrojb,
           main = "Life-Stress Relationship: Power Law Model",
           xlab = "Voltage (kV)",
           ylab = "Time to Failure")

```

7.6 Repairable Systems Plots

These functions accept the fitted objects from ReliaGrowR, the same objects produced in Chapter 6. The section introduces the `pump_data` fleet dataset first via the non-parametric tools, then uses the same data for the parametric NHPP fit.

7.6.1 Exposure Plot: `plotly_exposure()`

```

pump_data <- data.frame(
  id      = c(1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 4, 5, 5),
  time    = c(310, 850, 1620, 420, 1050, 2100, 580, 1890,
             240, 710, 1380, 2400, 530, 1740),
  event   = rep(1, 14)
)

```

```

)

exp_result <- exposure(id = pump_data$id, time = pump_data$time,
                      event = pump_data$event)
plotly_exposure(exp_result,
                main = "Fleet Exposure Plot",
                xlab = "Operating Hours",
                ylab = "Event Rate (failures / hour)")

```

7.6.2 MCF Plot: `plotly_mcf()`

```

end_times <- c("1" = 3000, "2" = 3000, "3" = 3000, "4" = 3000, "5" = 3000)

mcf_result <- mcf(id = pump_data$id, time = pump_data$time,
                  event = pump_data$event, end_time = end_times)
plotly_mcf(mcf_result,
           main = "MCF: Industrial Pump Fleet",
           xlab = "Operating Hours",
           ylab = "Expected Cumulative Failures per System")

```

7.6.3 NHPP Plot: `plotly_nhpp()`

The parametric Crow-AMSAA fit uses the same `pump_data` fleet; individual failure times are extracted and sorted across all systems:

```

failure_times <- sort(pump_data$time[pump_data$event == 1])

fit_nhpp <- nhpp(time = failure_times, event = rep(1, length(failure_times)))
plotly_nhpp(fit_nhpp,
            main = "Crow-AMSAA: Industrial Pump Fleet",
            xlab = "Operating Hours",
            ylab = "Cumulative Failures")

```

7.7 Embedding in Reports and Dashboards

ReliaPlotR outputs are standard `plotly` objects and embed natively in:

Quarto / R Markdown: interactive widgets in HTML output, static fallback in PDF:

```

```{r}
library(ReliaPlotR)
library(WeibullR)
failures <- c(30, 49, 82, 90, 96)
obj <- wblr.conf(wblr.fit(wblr(failures)))
plotly_wblr(obj)
```

```

Shiny: use `plotlyOutput()` / `renderPlotly()`:

```

library(shiny)
library(plotly)
library(ReliaPlotR)
library(WeibullR)

ui <- fluidPage(plotlyOutput("wblr_plot"))

server <- function(input, output) {
  output$wblr_plot <- renderPlotly({
    obj <- wblr.conf(wblr.fit(wblr(c(30, 49, 82, 90, 96))))
    plotly_wblr(obj)
  })
}

shinyApp(ui, server)

```

7.8 When to Use ReliaPlotR

| Use ReliaPlotR when... | Use base WeibullR / ReliaGrowR plots when... |
|---|--|
| Delivering results to non-R stakeholders | Running quick exploratory analysis in a script |
| Embedding in an HTML report or dashboard | Generating PDF / print output directly |
| Stakeholders need to hover and inspect points | Batch-processing many analyses |
| Presenting in a web browser or Shiny app | Working in a non-HTML environment |

7.9 Getting Help

```
help(package = "ReliaPlotR")
```

- CRAN page: <https://CRAN.R-project.org/package=ReliaPlotR>

7.10 Summary

ReliaPlotR wraps every major fitted object in the ReliaLearnR ecosystem, including `wblr`, `duane`, `rga`, `nhpp`, `mcf`, `exposure`, `alt.make`, and `alt.fit`, in an interactive plotly chart. The pattern is consistent: fit your model with WeibullR, ReliaGrowR, or WeibullR.ALT as usual, then pass the result to the corresponding `plotly_*` function. All nine functions support the same customization arguments (titles, axis labels, colors, grid control) and return standard plotly objects compatible with Quarto, R Markdown, and Shiny.

8 Web-Based Analysis with ReliaShiny

8.1 Introduction

All of the analyses in this book require writing R code. **ReliaShiny** (Govan 2023b) removes that requirement; it is a point-and-click Shiny (Chang et al. 2022) web application that exposes the same reliability analysis workflows (life data analysis, reliability growth, accelerated life testing, repairable systems) through a graphical interface. No R programming knowledge is needed to run an analysis.

This chapter introduces ReliaShiny, tours its modules, and explains when to reach for it versus a scripted R workflow.

8.2 Installation and Launch

ReliaShiny is available on CRAN:

```
install.packages("ReliaShiny")
```

Once installed, launch the app from the R console:

```
library(ReliaShiny)  
ReliaShiny()
```

This opens the application in your default web browser. The app runs locally; no internet connection or account is required.

8.3 App Overview

When ReliaShiny opens, the left sidebar displays the available analysis modules, each corresponding to a chapter in this book:

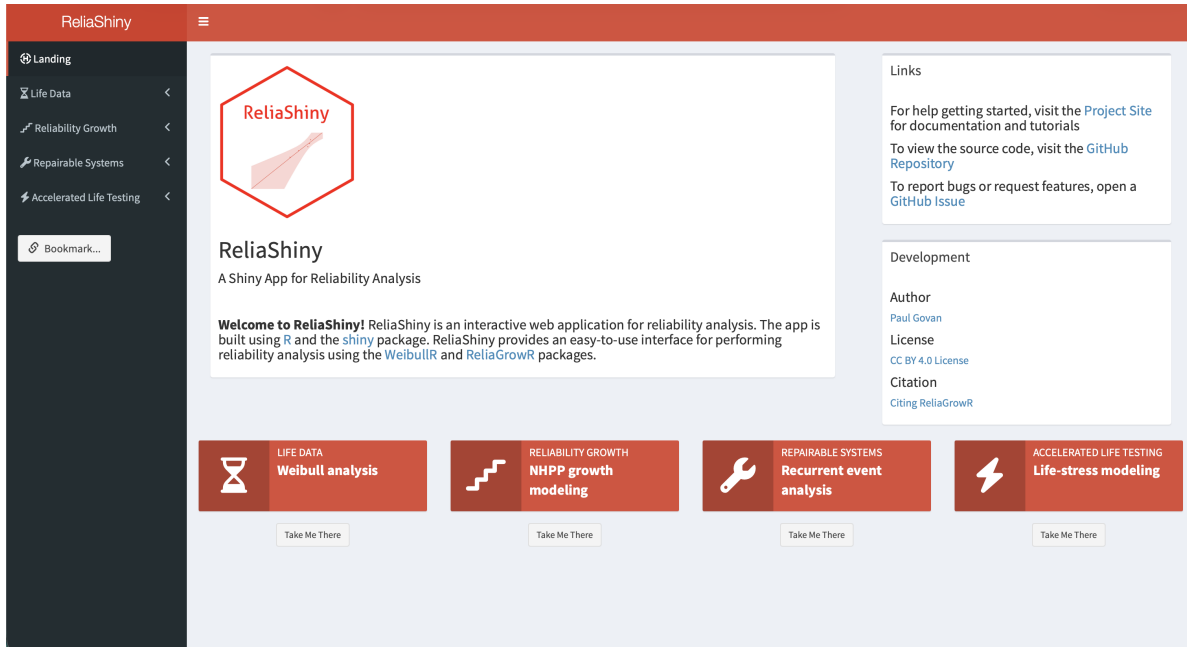


Figure 8.1: ReliaShiny landing page showing the sidebar module list.

| Module | Corresponding chapter | Key capabilities |
|---------------------------------|-----------------------|--|
| Life Data Analysis | Chapter 3 | 2-parameter, 3-parameter Weibull, Weibayes; MRR and MLE; right-censored data |
| Reliability Growth | Chapter 4 | Duane and Crow-AMSAA model fitting; piecewise NHPP |
| Accelerated Life Testing | Chapter 5 | Arrhenius and Power Law models; relationship plots |
| Repairable Systems | Chapter 6 | Power Law Process (NHPP); MCF; fleet exposure |

Each module follows the same three-step workflow:

1. **Input**: enter or upload your data using form fields, tables, or file uploads.
2. **Analyze**: click **Run** to fit the model.
3. **Output**: view plots, parameter estimates, and summary statistics; download results.

8.4 Module Walkthroughs

8.4.1 Life Data Analysis Module

The LDA module fits Weibull models and generates probability and contour plots. The app includes a preloaded **Time-to-Failure** dataset so you can explore the workflow immediately without uploading data.

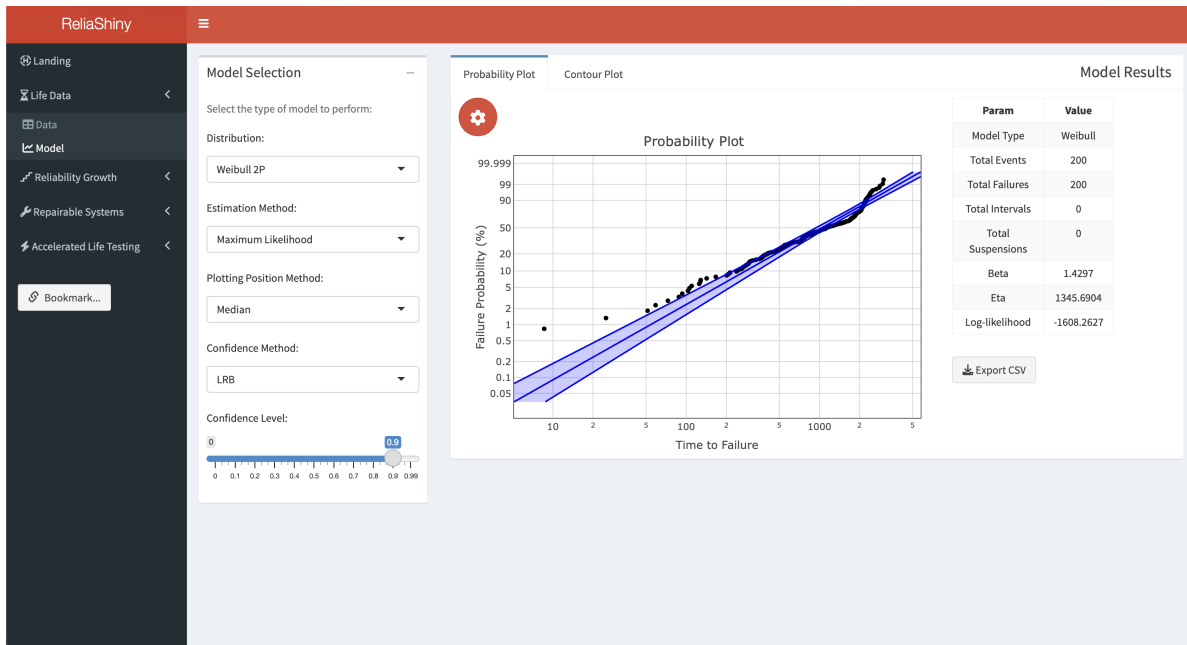


Figure 8.2: LDA module showing a fitted Weibull probability plot.

Workflow:

1. Select the *Life Data Analysis* tab in the sidebar.
2. Navigate to the **Data** sub-menu. The preloaded *Time-to-Failure* dataset is already loaded; explore additional options for data arrangement or upload your own CSV with columns **time** and **event** (1 = failure, 0 = suspension).
3. Navigate to the **Model** sub-menu. Select the distribution (2-parameter Weibull by default) and estimation method (MRR or MLE).
4. Click **Fit Model**. The **Probability Plot** appears in the main panel, with $\hat{\beta}$, $\hat{\eta}$, and the Anderson-Darling statistic.
5. Switch to the **Contour Plot** tab for an interactive likelihood ratio contour showing the joint confidence region for $\hat{\beta}$ and $\hat{\eta}$.
6. Click **Download Plot** to save the current chart as a PNG.

CSV format for upload:

```
time,event
30,1
49,1
82,1
90,1
96,1
100,0
45,0
10,0
```

A 0 in the `event` column marks a suspension (right-censored observation).

See the [Life Data Analysis article](#) for a full walkthrough.

8.4.2 Reliability Growth Module

The Reliability Growth module fits Crow-AMSAA and Duane reliability growth models. The app includes a preloaded **Simple Data** dataset for immediate exploration.

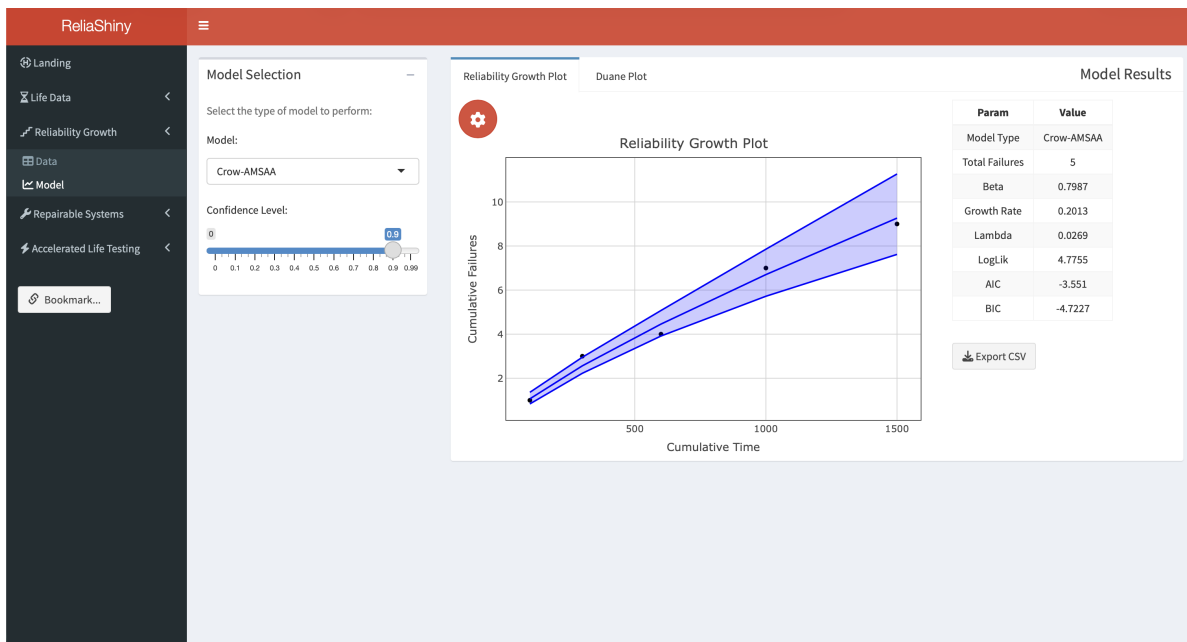


Figure 8.3: Reliability Growth module showing a fitted Crow-AMSAA cumulative failures plot.

Workflow:

1. Select *Reliability Growth* in the sidebar.
2. Navigate to the **Data** sub-menu. The preloaded *Simple Data* dataset is loaded; designate the **failures** column as the failure indicator, or upload your own data.
3. Navigate to the **Model** sub-menu.
4. Click **Fit**. The **Reliability Growth Plot** (Crow-AMSAA cumulative failures) appears with the fitted curve and $\hat{\beta}$.
5. Switch to the **Duane Plot** tab to view the log-log cumulative MTBF plot for the same data.

See the [Reliability Growth Analysis article](#) for a full walkthrough.

8.4.3 Accelerated Life Testing Module

The ALT module fits Arrhenius and Power Law acceleration models using the `WeibullR.ALT` package (see Chapter 5). The app includes the preloaded **Nelson Data** dataset.

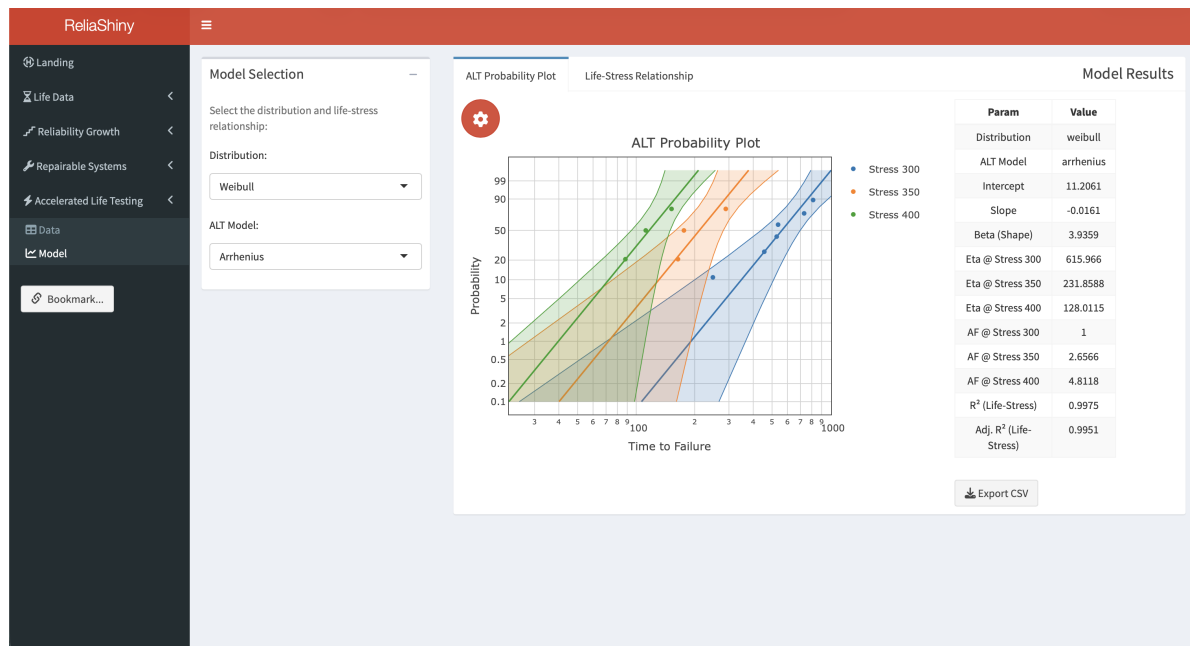


Figure 8.4: ALT module showing a probability plot and life-stress relationship.

Workflow:

1. Select *Accelerated Life Testing* in the sidebar.
2. Navigate to the **Data** sub-menu. The preloaded *Nelson Data* is already loaded. Map the columns: select the **Stress Level**, **Time to Failure**, and **Event Indicator** columns from the dropdowns.

3. Navigate to the **Model** sub-menu. Choose the acceleration model (**Arrhenius** for temperature, **Power Law** for voltage/mechanical stress) and distribution (Weibull or Lognormal).
4. Click **Fit ALT Model**. The **ALT Probability Plot** appears, with one curve per stress level.
5. Switch to the **Life-Stress Relationship** tab to view how predicted life varies across the stress range, with percentile lines (P10, P63.2, P90).

See the [Accelerated Life Testing](#) article for a full walkthrough.

8.4.4 Repairable Systems Module

The Repairable Systems module fits the Power Law Process (Crow-AMSAA) and computes the MCF and exposure for fleet data. The app includes a preloaded **Simple Data Set**.

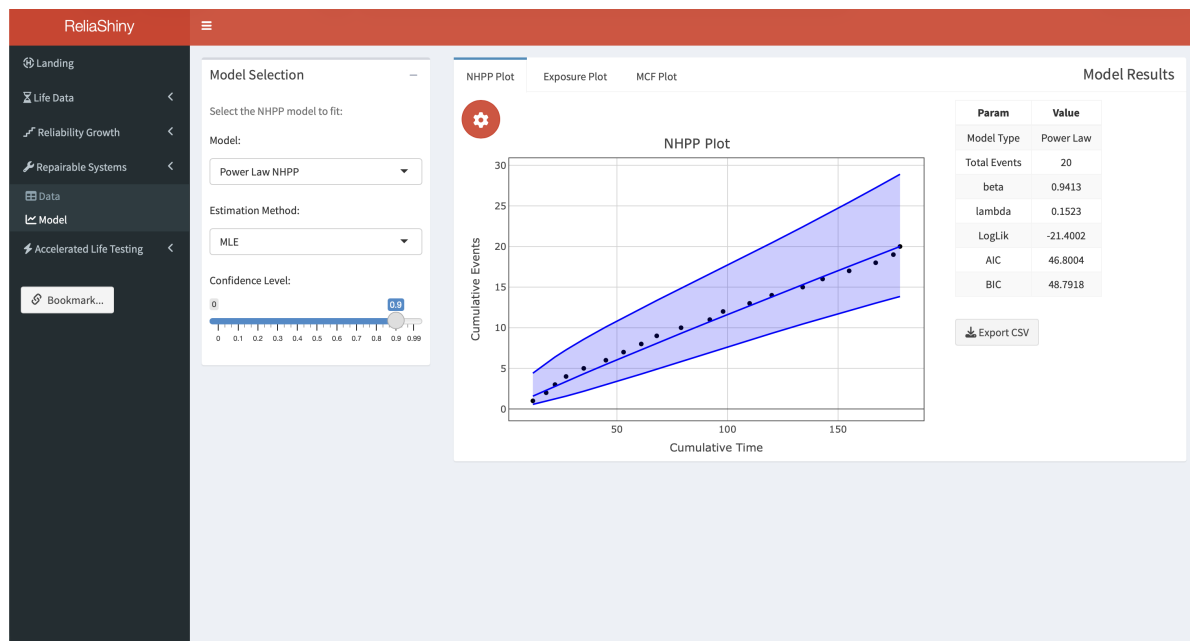


Figure 8.5: Repairable Systems module showing a Mean Cumulative Function plot.

Workflow:

1. Select *Repairable Systems* in the sidebar.
2. Navigate to the **Data** sub-menu. The preloaded *Simple Data Set* is loaded; map the **System ID**, **Event Time**, and **Event Indicator** columns, or upload your own fleet data with columns `id`, `time`, and `event`.
3. Navigate to the **Model** sub-menu.

4. Click **Analyze**. Three plots are produced:

- **NHPP Plot:** Crow-AMSAA Power Law fit with cumulative failures curve and $\hat{\beta}$, $\hat{\lambda}$.
- **Exposure Plot:** fleet-level event rate as a function of operating time.
- **MCF Plot:** non-parametric Mean Cumulative Function with confidence bounds.

Example CSV format:

```
id,time,event
1,310,1
1,850,1
1,1620,1
2,420,1
2,1050,1
2,3000,0
```

The last row for system 2 (`event = 0`) indicates that system 2 was observed until hour 3,000 without a third failure.

See the [Repairable Systems Analysis article](#) for a full walkthrough.

8.5 Data Upload Tips

- **CSV files:** core column names (`time`, `event`, `id`, `qty`) are case-sensitive and must be **lowercase**. Stress column names (e.g., `TempC`, `Voltage`) may use any valid R name; they are selected in the app's dropdown.
- **Missing values:** rows with `NA` in the `time` column are automatically excluded.
- **Large datasets:** the app handles datasets up to ~10,000 rows without performance issues.
- **Units:** ReliaShiny is unit-agnostic; use hours, years, cycles, or any consistent unit. Label your outputs accordingly.

8.6 Downloading Results

Every module provides download buttons for:

- **Plot PNG:** the current chart at screen resolution.
- **Results CSV:** parameter estimates and summary statistics in tabular form.
- **Report HTML:** a self-contained report combining the inputs, plot, and results (available in LDA and RT modules).

8.7 When to Use ReliaShiny vs Scripted R

| Situation | ReliaShiny | Scripted R |
|---|------------|------------|
| Quick one-off analysis | | |
| Teaching / demonstration | | |
| Sharing with non-R users | | |
| Automating repeated analyses | | |
| Custom plots and formatting | | |
| Reproducible research / version control | | |
| Embedding results in a Quarto book | | |
| Large batch processing | | |

ReliaShiny excels at rapid exploration, prototyping, and sharing results with stakeholders who do not use R. For production workflows, automated pipelines, and fully reproducible analyses, scripted R with the underlying packages (WeibullR, ReliaGrowR, WeibullR.ALT) gives more control.

8.8 Getting Help

Within the app, click the ? icon on any panel for a tooltip explaining the inputs and outputs for that module.

From R:

```
help(package = "ReliaShiny")
```

Additional resources:

- CRAN page: <https://CRAN.R-project.org/package=ReliaShiny>
- Articles and walkthroughs: <https://paulgovan.github.io/ReliaShiny/>

8.9 Summary

ReliaShiny provides a point-and-click interface to the ReliaLearnR analysis ecosystem, covering life data analysis, reliability growth, accelerated life testing, and repairable systems, with no R coding required. Install it with `install.packages("ReliaShiny")` and launch it with `ReliaShiny()`. Use it for rapid exploration, stakeholder demonstrations, and sharing results; switch to scripted R for reproducibility, automation, and custom outputs.

9 Reproducible Reliability Analysis with Quarto

Scripted R solves the *computation* problem, producing analyses that are exact, repeatable, and version-controlled. But a folder of R scripts and PNG plots is still hard to share, audit, or publish. Quarto (Allaire et al. 2024) solves the *documentation* problem by weaving code, equations, narrative, and figures into a single re-executable source file. The result is a document that is both the analysis *and* the report, one that any colleague can re-run months later and reproduce every number exactly. See Section 9.9 for guidance on when to choose Quarto over ReliaShiny or AI/MCP.

9.1 Learning Objectives

By the end of this chapter, you will be able to:

- Explain what reproducibility means in a reliability analysis context and why it matters for audits, re-analysis, and team collaboration.
- Describe the role of Quarto as a literate-programming tool for combining R code, equations, and narrative.
- Create a `.qmd` document with a YAML front matter block, code chunks, and inline LaTeX equations.
- Render a `.qmd` to HTML, PDF, and Word from the R console or terminal.
- Use parameterized Quarto reports to run the same analysis on different datasets without editing code.
- Embed WeibullR, ReliaGrowR, and ReliaLearnR analyses inside a `.qmd` and save publication-quality plots.
- Distinguish when to use Quarto, ReliaShiny, and AI/MCP for a given reliability task.

9.2 Why Reproducibility Matters

9.2.1 Audits and Regulatory Compliance

Defence, aerospace, nuclear, and medical-device industries require that every reliability analysis be traceable, including the data, the method, and the software version. A Quarto document combined with an `renv.lock` package snapshot provides an exact, time-stamped audit trail. Regulators can re-run the analysis years later and confirm the results.

9.2.2 Re-Analysis and Collaboration

A colleague inheriting a folder of PNG plots and a spreadsheet cannot reproduce the analysis. A `.qmd` file is self-contained: every assumption, model choice, and filtering step is visible in the code. When field data is updated, re-running the document updates every table, plot, and inline number automatically.

9.2.3 Version Control

Quarto source files are plain text. They can be committed to Git alongside data, line-diffed, and reviewed in pull requests. Changes to an analysis are visible as diffs, which is not possible with binary spreadsheets or saved Shiny sessions.

Review

Name two situations in a reliability engineering context where a fully reproducible analysis would be required.

Answer

Examples include: regulatory or contractual audits requiring traceability of parameter estimates; re-analysis after a field event using updated failure data; peer review of a reliability growth claim; handing off an analysis to a new team member.

9.3 Introduction to Quarto

Quarto is an open-source scientific and technical publishing system built on Pandoc and maintained by Posit. A `.qmd` file is a plain-text document that interleaves YAML front matter, Markdown prose, and executable code chunks. When rendered, Quarto executes the code, captures the output, and produces HTML, PDF, Word, or presentations from the same source.

| Term | Meaning |
|--------------------------|---|
| <code>.qmd</code> | Quarto Markdown source file |
| YAML front matter | Document metadata block delimited by <code>---</code> |
| Code chunk | Fenced block (<code>```\${code}```</code>) containing executable R code |
| Chunk option | <code>#!</code> -prefixed line controlling chunk behaviour (<code>eval</code> , <code>echo</code> , <code>fig-cap</code>) |
| Render | Convert the <code>.qmd</code> to its target format |
| <code>_quarto.yml</code> | Project-level configuration for multi-file books and websites |

9.3.1 Installing Quarto

The Quarto CLI is a standalone download from <https://quarto.org/docs/get-started/>. The `quarto` R package (Allaire 2024) provides R-level wrappers:

```
install.packages("quarto")
quarto::quarto_version() # confirm the CLI is found
```

9.3.2 Anatomy of a .qmd File

A minimal `.qmd` has three zones: a YAML front matter block, Markdown prose, and fenced R chunks.

```
---
title: "My Reliability Report"
author: "Your Name"
date: today
format: html
---

## Introduction

This report analyses pump failure data using the Weibull distribution.

```{r}
#| echo: true
library(WeibullR)
failures <- c(30, 49, 82, 90, 96)
fit <- wblr.fit(wblr(failures), method.fit = "mle")
plot(fit)
```
```

9.3.3 Rendering

From the R console:

```
library(quarto)
quarto_render("my-report.qmd") # default format
quarto_render("my-report.qmd", output_format = "pdf")
quarto_render("my-report.qmd", output_format = "docx")
```

From the terminal:

```
quarto render my-report.qmd
quarto render my-report.qmd --to pdf
```

9.4 A Basic Reliability Report in Quarto

9.4.1 YAML Front Matter for Reliability Reports

A professional deliverable typically targets both HTML and PDF. The `code-fold: true` option collapses code blocks in HTML output, keeping reports tidy for stakeholders while keeping the code accessible.

```
---
title: "Pump Fleet Weibull Analysis"
author: "Reliability Engineering"
date: today
format:
  html:
    toc: true
    number-sections: true
    code-fold: true
    code-tools: true
  pdf:
    toc: true
    number-sections: true
execute:
  echo: true
  warning: false
  message: false
---
```

9.4.2 Embedding Equations

Quarto uses LaTeX for mathematical notation. Display equations are delimited by `$$...$$`:

The Weibull reliability function is:

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

where β is the shape parameter and η is the characteristic life.

9.4.3 Code Chunk Options

| Option | Effect |
|---------------------------------|--|
| <code>#\ eval: false</code> | Show code but do not run it |
| <code>#\ echo: false</code> | Run code but hide the source block |
| <code>#\ include: false</code> | Run code, hide both code and output |
| <code>#\ fig-cap: "..."</code> | Add a figure caption |
| <code>#\ fig-width: 6</code> | Set output figure width in inches |
| <code>#\ label: fig-xxx</code> | Enable cross-referencing with <code>@fig-xxx</code> |
| <code>#\ cache: true</code> | Cache the result; re-run only when the chunk changes |

9.4.4 A Complete Weibull Report Block

```
library(WeibullR)

failures    <- c(30, 49, 82, 90, 96)
suspensions <- c(100, 45, 10)

obj <- wblr(failures, suspensions)
obj <- wblr.fit(obj, method.fit = "mle")
obj <- wblr.conf(obj, method.conf = "lrb")
plot(obj)
```

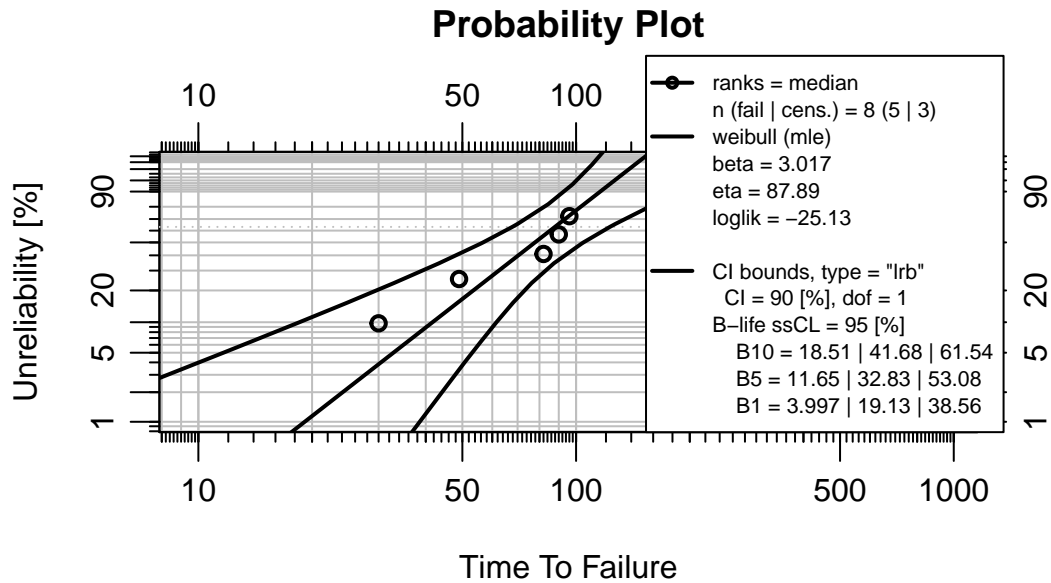


Figure 9.1: Weibull probability plot for pump fleet failures with MLE fit and likelihood ratio confidence bounds.

Inline R expressions embed computed values directly in prose without hard-coding numbers, a key reproducibility benefit. For example:

```
The fitted shape parameter is  $\hat{\beta}$  = `r round(obj$fit[[1]]$beta,
↪ 2)`.
```

When the analysis is re-run with updated data, this sentence updates automatically.

i Try It

Create a file `pump-report.qmd` with the YAML front matter shown above. Add a code chunk that fits a Weibull model to failure times `c(120, 205, 310, 445, 590, 780)` using MLE and plots the result. Render it to HTML.

Solution

```
library(WeibullR)
failures <- c(120, 205, 310, 445, 590, 780)
obj <- wblr.fit(wblr(failures), method.fit = "mle")
plot(obj)
```

```
Add quarto_render("pump-report.qmd") in a separate R session to render.
```

9.5 Parameterized Reports

9.5.1 What Are Parameters?

Quarto parameters let a single `.qmd` template run against different datasets or settings without touching any code. Parameters are declared in YAML and accessed in R via `params$name`.

9.5.2 Declaring Parameters in YAML

```
---
title: "Reliability Report"
params:
  dataset: "pump_failures.csv"
  component: "Pump A"
  conf_level: 0.90
---
```

9.5.3 Using Parameters in Code

```
data      <- read.csv(params$dataset)
failures  <- data$time[data$event == 1]

cat("Component:", params$component, "\n")
cat("Confidence level:", params$conf_level, "\n")
```

9.5.4 Rendering Multiple Reports

A loop over `quarto_render()` produces one report per component from a single template:

```
library(quarto)

components <- c("Pump A", "Pump B", "Compressor C")
datasets   <- c("pump_a.csv", "pump_b.csv", "comp_c.csv")
```

```

for (i in seq_along(components)) {
  quarto_render(
    "reliability-report.qmd",
    execute_params = list(
      dataset = datasets[i],
      component = components[i]
    ),
    output_file = paste0(gsub(" ", "_", components[i]), "_report.html")
  )
}

```

Review

A reliability engineer must produce monthly Weibull reports for six pump models. Without parameterization, how many `.qmd` files are needed? With parameterization?

Answer

Without parameters: six separate files, each requiring manual edits when the analysis changes. With parameters: one template file rendered six times, with changes to the analysis logic made in one place and propagated to all reports automatically.

9.6 Integration with Reliability R Packages

All reliability packages used in this book work inside `.qmd` files without modification, and Quarto captures their output automatically.

9.6.1 WeibullR: Life Data Analysis

See Chapter 3 for the statistical background. Inside a `.qmd`, a complete Weibull analysis is:

```

library(WeibullR)

failures <- c(500, 1200, 900, 1500, 750)
suspensions <- c(2000, 2000)

obj <- wblr(x = failures, s = suspensions)
obj <- wblr.fit(obj, method.fit = "mle")
obj <- wblr.conf(obj, method.conf = "lrb")
plot(obj)

```

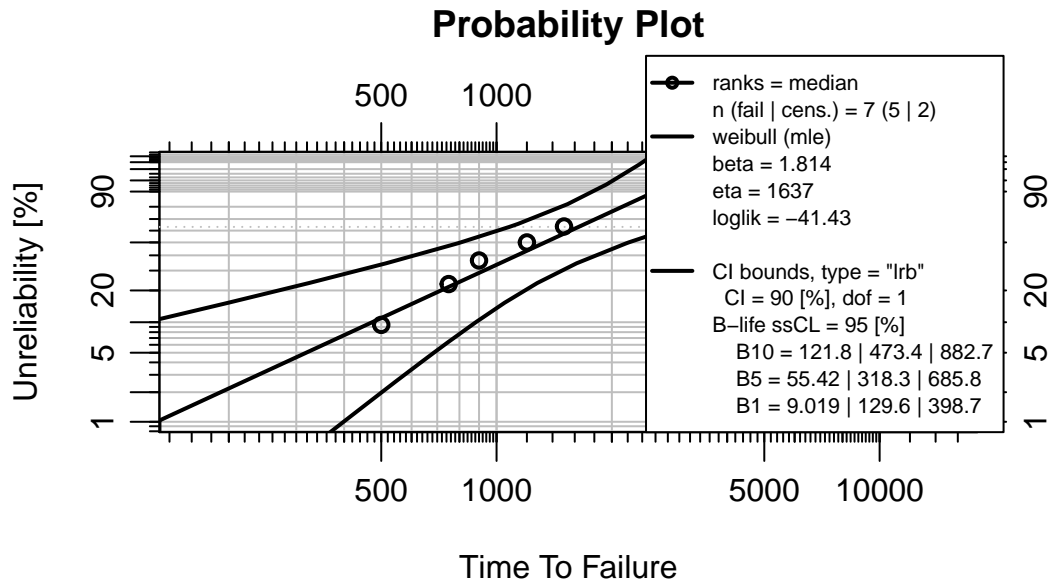


Figure 9.2: Weibull probability plot with MLE fit and confidence bounds.

9.6.2 ReliaGrowR: Reliability Growth

See Chapter 4 for Crow-AMSAA model details:

```

library(ReliaGrowR)

times    <- c(9.2, 25, 61.5, 260, 300, 710, 916, 1010, 1220)
failures <- rep(1, length(times))
rga_fit  <- rga(times, failures)
plot(rga_fit)

```

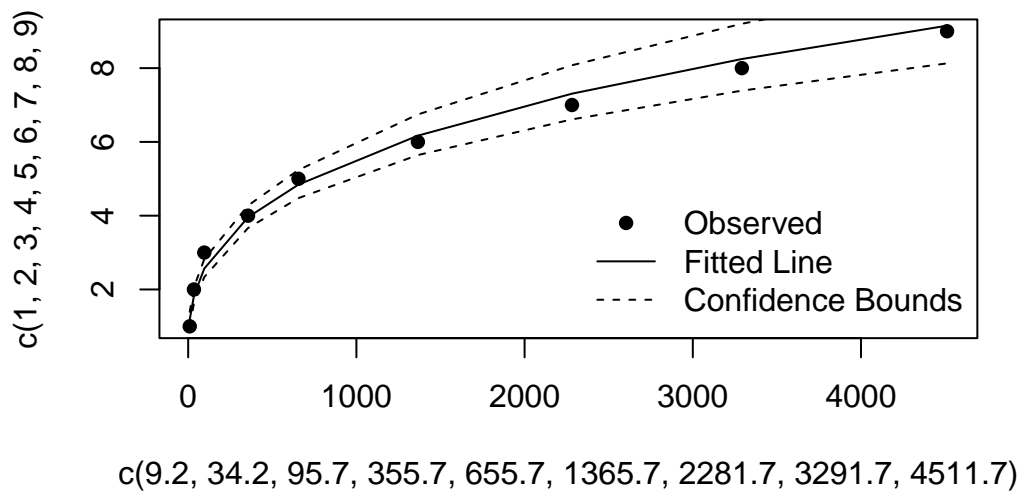


Figure 9.3: Crow-AMSAA reliability growth fit.

9.6.3 ReliaLearnR: RAM Metrics

See Chapter 1 for the RAM helper functions:

```
library(ReliaLearnR)

reliability <- rel(outageTime = 10, totalTime = 5 * 365)
availability <- avail(unavailTime = 25, totalTime = 5 * 365)
failure_rate <- fr(failures = 12, totalTime = 5 * 365)

cat(sprintf("Reliability:   %.1f%%\n", reliability * 100))
```

Reliability: 99.5%

```
cat(sprintf("Availability:  %.1f%%\n", availability * 100))
```

Availability: 98.6%

```
cat(sprintf("Failure rate:  %.4f failures/day\n", failure_rate))
```

Failure rate: 0.0066 failures/day

9.6.4 Saving Plots

To save a static plot from inside a .qmd for use in presentations or other documents:

```
png("weibull_plot.png", width = 800, height = 600, res = 150)
plot(obj)
dev.off()
```

For interactive ReliaPlotR plots (see Chapter 7):

```
library(ReliaPlotR)
library(htmlwidgets)

p <- plotly_wblr(obj)
saveWidget(p, "weibull_interactive.html", selfcontained = TRUE)
```

i Try It

In a .qmd file, fit a Crow-AMSAA model to failure times `c(50, 100, 180, 310, 450, 620)` with `T = 800`, plot the result, and add a prose sentence using inline R that reports the fitted $\hat{\beta}$.

Solution

```
library(ReliaGrowR)
times    <- c(50, 100, 180, 310, 450, 620)
failures <- rep(1, length(times))
fit      <- rga(times, failures)
plot(fit)
```

```
Inline sentence: "The fitted shape parameter is  $\hat{\beta} =$ `r
↪ round(fit$betas, 3)`."
```

9.7 This Book Is Built with Quarto

i This Book Is a Quarto Project

Reliability Analysis with R: A Companion Guide is itself built with Quarto. Every chapter is a `.qmd` file. Every code block you have read in this book was executed by Quarto at render time and its output captured automatically. The project configuration lives in `_quarto.yml` at the root of the book directory, which specifies the chapter order, HTML theme (`cosmo/darkly`), PDF document class (`scrreprt`), TOC depth, and global chunk options (`echo: true, warning: false, message: false`).

To build the book yourself, clone the repository and run:

```
library(quarto)
quarto_render() # renders all chapters in _quarto.yml order
```

or from the terminal:

```
quarto render
```

The rendered book is published to GitHub Pages on each push, so the online version always reflects the exact source in the repository.

9.8 Quarto Books and Multi-File Projects

A Quarto book is a `type: book` project where `_quarto.yml` controls everything: chapter order, sidebar search, bibliography, HTML theme, PDF layout, and global execution options. Adding a new chapter requires only creating a `.qmd` file and adding one line to the chapters list, with no other configuration changes needed.

The global `execute: block` in `_quarto.yml` sets default chunk behaviour across all chapters:

```
execute:
  echo: true
  warning: false
  message: false
  cache: false
```

Individual chunks can override these defaults with their own `#|` options. `cache: false` is appropriate during active writing to ensure output is always fresh; switch to `#| cache: true` on individual chunks for computationally expensive analyses.

9.9 When to Use Quarto vs. ReliaShiny vs. AI/MCP

Each tool in the ReliaLearnR ecosystem excels in a different context. Quarto is the right choice when the deliverable must be reproducible, auditable, or version-controlled. ReliaShiny (see Chapter 8) suits rapid exploration or stakeholder demonstrations with no R requirement. AI + MCP (see Chapter 10) enables conversational, interpretive, and iterative analysis in plain language.

| Situation | Quarto | ReliaShiny | AI + MCP |
|--|--------|------------|----------|
| Reproducible analysis with version control | | | |
| Automated batch reports across multiple datasets | | | |
| Embedding results in a book, paper, or report | | | |
| Custom plot formatting and multi-panel layouts | | | |
| Long-running or computationally intensive analyses | | | |
| Quick one-off exploration without writing code | | | |
| Sharing analysis with non-R users via a browser | | | |
| Teaching or live stakeholder demonstration | | | |
| Conversational model selection and interpretation | | | |
| Iterating on an analysis in plain language | | | |
| Explaining results to a non-technical audience | | | |

9.10 Getting Help

```
help(package = "quarto")
```

- Quarto documentation: <https://quarto.org/docs/guide/>
- Quarto for R: <https://quarto.org/docs/computations/r.html>
- Quarto gallery: <https://quarto.org/docs/gallery/>

9.11 Summary

Quarto is the standard tool for reproducible, publishable reliability analysis in R. By combining code, equations, prose, and figures in a single `.qmd` source file, it eliminates the gap between analysis and report and makes every result traceable back to its inputs.

- **Reproducibility:** a `.qmd` file is the complete specification of an analysis; anyone with the file, data, and packages can regenerate every number and plot.

- **Parameterization:** `params:` in YAML enables one template to produce reports for many components, datasets, or time periods without changing any code.
- **Integration:** WeibullR, ReliaGrowR, WeibullR.ALT, ReliaPlotR, and ReliaLearnR all work inside `.qmd` files without modification.
- **Formats:** `quarto_render()` produces HTML, PDF, Word, and more from the same source file.
- **Books:** this guide is a `type: book` Quarto project; `_quarto.yml` controls chapter order, theme, sidebar, and global chunk options.
- **Complement, not replacement:** Quarto does not replace ReliaShiny (use ReliaShiny for point-and-click exploration) or AI + MCP (use MCP for conversational analysis), as each tool excels in a different context (see Section 9.9).

9.12 Review Questions

Review Questions

1. What is the difference between `#! eval: false` and `#! echo: false` in a Quarto code chunk?

Answer

`eval: false` shows the code but does not run it, so no output is produced. `echo: false` runs the code and captures the output but hides the code block from the reader.

2. You need to produce monthly Weibull reports for eight pump families. How would you structure a parameterized Quarto workflow?

Answer

Declare `params: dataset:` and `params: component:` in the YAML front matter. Write the analysis once using `params$dataset` and `params$component`. Call `quarto_render()` in a loop with different `execute_params` for each family. Eight reports are produced from one `.qmd` source file.

3. Name two chunk options that control figure output in a Quarto document.

Answer

Any two of: `fig-cap` (caption text), `fig-width` / `fig-height` (dimensions in inches), `label` (enables `@fig-xxx` cross-referencing), `fig-alt` (accessibility alt text).

4. This book is itself a Quarto project. What file controls the chapter order, HTML theme, and global chunk options?

Answer

`_quarto.yml`, the project configuration file at the root of the book directory.

5. When would you choose ReliaShiny over a Quarto document for a reliability analysis?

Answer

When the audience does not use R (ReliaShiny requires no coding), when the goal is rapid interactive exploration rather than a deliverable document, or when a live stakeholder

demonstration is needed.

10 AI-Assisted Reliability Analysis with MCP

10.1 Introduction

All of the analyses in this book require writing R code or navigating a Shiny app. **Model Context Protocol (MCP)** opens a third path: letting an AI assistant call your R functions directly during a conversation, so you can describe what you want in plain language and the AI runs the analysis, interprets the results, and generates the plots.

MCP is an open standard that defines how an AI assistant discovers and calls external tools in the same way a web browser calls REST APIs. Reliability workflows are a natural fit: fitting a growth model, predicting future failures, computing a fleet MCF, and saving an interactive plot are all discrete, well-defined operations that map cleanly to MCP tools.

This chapter shows how to expose `ReliaGrowR` (Govan 2024) and `ReliaPlotR` (Govan 2023a) as an MCP server so that Claude Desktop, Claude Code, or any other MCP-compatible client can run reliability analyses on demand.

10.2 Prerequisites

Install the MCP infrastructure packages. `ellmer` (Wickham and Cheng 2025) provides the `tool()` constructor; `mcptools` (Posit PBC 2025) provides the server transport:

```
install.packages(c("ellmer", "mcptools"))
```

Ensure the reliability packages are also installed:

```
install.packages(c("ReliaGrowR", "ReliaPlotR", "htmlwidgets"))
```

10.3 MCP Tools for ReliaGrowR

The `reliagrow_mcp_tools()` function below returns a named list of `ellmer::tool()` objects. Each tool is a thin wrapper: it accepts plain-text inputs (times and failures as comma-separated strings), calls the underlying `ReliaGrowR` function, extracts the key scalar results, and returns a plain list that the AI receives as JSON.

```
library(ellmer)
library(ReliaGrowR)

reliagrow_mcp_tools <- function() {

  parse_vec <- function(x) as.numeric(strsplit(trimws(x), "\\s*,\\s*")[[1]])

  list(

    fit_duane = tool(
      .name      = "fit_duane",
      .description = "Fit a Duane reliability growth model to cumulative
        ↪ failure data.
        ↪ Returns the growth slope (beta), scale parameter (K),
        ↪ and
        ↪ cumulative MTBF at the last observation.",
      times      = type_string("Comma-separated cumulative times (e.g.
        ↪ '100,200,300')"),
      failures    = type_string("Comma-separated cumulative failure counts (e.g.
        ↪ '1,2,1')"),
      function(times, failures) {
        t <- parse_vec(times)
        f <- parse_vec(failures)
        fit <- duane(t, f)
        list(
          beta = round(fit$beta, 4),
          K    = round(fit$K, 4),
          cmtbf = round(fit$K * max(t)^(fit$beta - 1), 2)
        )
      }
    ),

    fit_crow_amsaa = tool(
      .name      = "fit_crow_amsaa",
      .description = "Fit a Crow-AMSAA (NHPP Power Law) reliability growth
        ↪ model.
```

```

        Returns beta (shape) and lambda (scale). Beta < 1 means
        reliability is improving; beta > 1 means worsening.",
times      = type_string("Comma-separated cumulative times"),
failures   = type_string("Comma-separated cumulative failure counts"),
function(times, failures) {
  t <- parse_vec(times)
  f <- parse_vec(failures)
  fit <- rga(t, f, model_type = "Crow-AMSAA")
  list(
    beta    = round(fit$beta, 4),
    lambda  = round(fit$lambda, 6)
  )
}
),

fit_piecewise_nhpp = tool(
  .name          = "fit_piecewise_nhpp",
  .description   = "Fit a piecewise Crow-AMSAA model with an optional known
  ↪ breakpoint.
        If no breakpoint is supplied, the change point is
  ↪ detected automatically.
        Returns per-segment beta and lambda values.",
times          = type_string("Comma-separated cumulative times"),
failures       = type_string("Comma-separated cumulative failure counts"),
breakpoint     = type_string("Optional breakpoint time, or empty string for
  ↪ auto-detection"),
function(times, failures, breakpoint = "") {
  t <- parse_vec(times)
  f <- parse_vec(failures)
  bp <- if (nchar(trimws(breakpoint)) == 0) NULL else
  ↪ as.numeric(breakpoint)
  fit <- if (is.null(bp)) {
    rga(t, f, model_type = "Piecewise NHPP")
  } else {
    rga(t, f, model_type = "Piecewise NHPP", breaks = bp)
  }
  list(
    segments    = length(fit$beta),
    beta        = round(fit$beta, 4),
    lambda      = round(fit$lambda, 6),
    breakpoints = fit$breaks
  )
}
)

```

```

    }
  ),

  predict_growth = tool(
    .name          = "predict_growth",
    .description   = "Predict cumulative failures at future times from a
    ↪ fitted Crow-AMSAA model.",
    times         = type_string("Historical cumulative times used to fit the
    ↪ model"),
    failures      = type_string("Historical cumulative failure counts"),
    pred_times    = type_string("Future times to predict at
    ↪ (comma-separated)"),
    function(times, failures, pred_times) {
      t <- parse_vec(times)
      f <- parse_vec(failures)
      pt <- parse_vec(pred_times)
      fit <- rga(t, f)
      pred <- predict_rga(fit, times = pt)
      data.frame(
        time          = pt,
        predicted_failures = round(pred$predicted, 2),
        lower_95      = round(pred$lower, 2),
        upper_95      = round(pred$upper, 2)
      )
    }
  ),

  fit_nhpp = tool(
    .name          = "fit_nhpp",
    .description   = "Fit a Crow-AMSAA Non-Homogeneous Poisson Process (NHPP)
    ↪ to
    ↪ repairable systems fleet failure times. Returns beta
    ↪ and lambda.",
    failure_times = type_string("Comma-separated failure times across all
    ↪ systems (pooled)"),
    function(failure_times) {
      ft <- parse_vec(failure_times)
      fit <- nhpp(time = ft, event = rep(1L, length(ft)))
      list(
        beta    = round(fit$beta, 4),
        lambda  = round(fit$lambda, 6)
      )
    }
  )
)

```

```

    }
  ),

  compute_mcf = tool(
    .name      = "compute_mcf",
    .description = "Compute the non-parametric Mean Cumulative Function
    ↪ (MCF) for a fleet.
    Provide system IDs, event times, event indicators
    ↪ (1=failure, 0=censored),
    and end times. Returns the MCF table.",
    ids      = type_string("Comma-separated system IDs (e.g.
    ↪ '1,1,1,2,2,2')"),
    times    = type_string("Comma-separated event times"),
    events   = type_string("Comma-separated event indicators (1=failure,
    ↪ 0=censored)"),
    end_times = type_string("Comma-separated end-of-observation times, one
    ↪ per unique system"),
    function(ids, times, events, end_times) {
      id <- parse_vec(ids)
      t  <- parse_vec(times)
      ev <- as.integer(parse_vec(events))
      et <- parse_vec(end_times)
      uid <- sort(unique(id))
      end <- setNames(et, as.character(uid))
      result <- mcf(id = id, time = t, event = ev, end_time = end)
      df <- as.data.frame(result)
      df[, sapply(df, is.numeric)] <- round(df[, sapply(df, is.numeric)],
    ↪ 4)
      df
    }
  ),

  plan_rdt = tool(
    .name      = "plan_rdt",
    .description = "Plan a Reliability Demonstration Test (RDT). Calculates
    ↪ the required
    test time to demonstrate a reliability target at a
    ↪ given confidence level.",
    target      = type_number("Target reliability (e.g. 0.90 for 90%)"),
    mission_time = type_number("Mission time in hours (or any consistent
    ↪ unit)"),
    conf_level  = type_number("Confidence level (e.g. 0.80 for 80%)"),

```

```

beta      = type_number("Assumed Weibull shape parameter"),
n         = type_integer("Number of test units"),
function(target, mission_time, conf_level, beta, n) {
  plan <- rdt(
    target      = target,
    mission_time = mission_time,
    conf_level  = conf_level,
    beta        = beta,
    n           = n
  )
  list(
    required_test_time = round(plan$test_time, 1),
    target_reliability = target,
    mission_time       = mission_time,
    confidence_level   = conf_level,
    sample_size        = n
  )
}
)
}

```

i What happens inside each tool

Each `tool()` call takes a `.name`, a `.description` that the AI reads to decide when to call the tool, named argument descriptors (`type_string()`, `type_number()`, etc.), and a handler function. The handler does the real work and returns a plain list or data frame — no raw R objects. The AI sees the returned value as structured JSON.

10.4 MCP Tools for ReliaPlotR

The `reliaplotr_mcp_tools()` function wraps the four main `ReliaGrowR`-backed plot functions. Because `plotly` widgets cannot be sent through a text protocol, each tool saves a self-contained HTML file and returns its path. The AI then reports the path to the user.

```

library(ellmer)
library(ReliaGrowR)
library(ReliaPlotR)
library(htmlwidgets)

```

```

reliaplotr_mcp_tools <- function() {

  parse_vec <- function(x) as.numeric(strsplit(trimws(x), "\\s*,\\s*")[[1]])

  save_plot <- function(widget, prefix) {
    outpath <- tempfile(pattern = prefix, fileext = ".html")
    htmlwidgets::saveWidget(widget, file = outpath, selfcontained = TRUE)
    list(path = outpath)
  }

  list(

    plot_duane = tool(
      .name          = "plot_duane",
      .description = "Fit a Duane model and save an interactive Duane plot as
↪ a
                    self-contained HTML file. Returns the file path.",
      times      = type_string("Comma-separated cumulative times"),
      failures   = type_string("Comma-separated cumulative failure counts"),
      title      = type_string("Optional plot title"),
      function(times, failures, title = "Duane Reliability Growth") {
        t <- parse_vec(times)
        f <- parse_vec(failures)
        fit <- duane(t, f)
        p <- plotly_duane(fit, main = title,
                        xlab = "Cumulative Time",
                        ylab = "Cumulative MTBF")
        save_plot(p, "duane_")
      }
    ),

    plot_crow_amsaa = tool(
      .name          = "plot_crow_amsaa",
      .description = "Fit a Crow-AMSAA model and save an interactive
↪ cumulative
                    failures plot as a self-contained HTML file. Returns
↪ the file path.",
      times      = type_string("Comma-separated cumulative times"),
      failures   = type_string("Comma-separated cumulative failure counts"),
      title      = type_string("Optional plot title"),
      function(times, failures, title = "Crow-AMSAA Reliability Growth") {
        t <- parse_vec(times)

```

```

    f <- parse_vec(failures)
    fit <- rga(t, f)
    p <- plotly_rga(fit, main = title,
                   xlab = "Cumulative Time",
                   ylab = "Cumulative Failures")
    save_plot(p, "rga_")
  }
),

plot_nhpp = tool(
  .name          = "plot_nhpp",
  .description   = "Fit a Crow-AMSAA NHPP to repairable system failure
↪ times and
↪ save an interactive plot as a self-contained HTML
↪ file.",
  failure_times = type_string("Comma-separated pooled failure times"),
  title         = type_string("Optional plot title"),
  function(failure_times, title = "Crow-AMSAA NHPP") {
    ft <- parse_vec(failure_times)
    fit <- nhpp(time = ft, event = rep(1L, length(ft)))
    p <- plotly_nhpp(fit, main = title,
                    xlab = "Operating Hours",
                    ylab = "Cumulative Failures")
    save_plot(p, "nhpp_")
  }
),

plot_mcf = tool(
  .name          = "plot_mcf",
  .description   = "Compute the non-parametric MCF for a fleet and save an
↪ interactive
↪ plot as a self-contained HTML file.",
  ids            = type_string("Comma-separated system IDs"),
  times         = type_string("Comma-separated event times"),
  events        = type_string("Comma-separated event indicators (1=failure,
↪ 0=censored)"),
  end_times     = type_string("Comma-separated end-of-observation times, one
↪ per system"),
  title         = type_string("Optional plot title"),
  function(ids, times, events, end_times, title = "Mean Cumulative
↪ Function") {
    id <- parse_vec(ids)

```

```

t <- parse_vec(times)
ev <- as.integer(parse_vec(events))
et <- parse_vec(end_times)
uid <- sort(unique(id))
end <- setNames(et, as.character(uid))
result <- mcf(id = id, time = t, event = ev, end_time = end)
p <- plotly_mcf(result, main = title,
               xlab = "Operating Hours",
               ylab = "Expected Cumulative Failures per System")
save_plot(p, "mcf_")
}
)
)
}

```

10.5 Launching the MCP Server

Save the two tool-list functions and the launch call in a single script, for example `mcp-server.R`:

```

# mcp-server.R
library(ReliaGrowR)
library(ReliaPlotR)
library(ellmer)
library(mcptools)
library(htmlwidgets)

# --- paste reliagrow_mcp_tools() and reliaplotr_mcp_tools() here, or
#   ↪ source() them ---

mcptools::mcp_server(
  tools = c(reliagrow_mcp_tools(), reliaplotr_mcp_tools())
)

```

Running this script starts an MCP server on the standard-IO transport, ready to accept tool calls from any connected client.

10.6 Connecting to Claude Desktop

Claude Desktop reads its MCP server list from a JSON config file. On macOS, open or create `~/Library/Application Support/Claude/claude_desktop_config.json` and add an entry for the reliability server:

```
{
  "mcpServers": {
    "reliagrow": {
      "command": "Rscript",
      "args": ["/absolute/path/to/mcp-server.R"]
    }
  }
}
```

Restart Claude Desktop. The ReliaGrowR and ReliaPlotR tools will appear in the tool palette and Claude will call them automatically when relevant.

10.7 Connecting to Claude Code

From the terminal, register the server with Claude Code using the `claude mcp add` command:

```
claude mcp add -s user reliagrow -- Rscript /absolute/path/to/mcp-server.R
```

Verify the server is registered:

```
claude mcp list
```

From that point, any Claude Code session will have access to the reliability tools.

10.8 Example AI Conversations

Once connected, you can have conversations like the following.

Example 1 — Crow-AMSAA fit and interpretation:

You: I ran a reliability growth test. The cumulative times were 100, 200, 300, 400, 500 hours with 1, 2, 1, 3, 2 failures. Fit a Crow-AMSAA model and tell me whether reliability is improving.

Claude: (*calls fit_crow_amsaa*) The fitted model gives $\hat{\lambda} = 0.87$ and $\hat{\lambda}_0 = 0.023$. Because $\hat{\lambda} < 1$, cumulative failures are growing more slowly than linearly — reliability is **improving**. The system is responding well to corrective actions during test.

Example 2 — Forecasting and test-continuation decision:

You: Using the same data, predict cumulative failures at 600, 800, and 1000 hours, and tell me whether I should continue testing.

Claude: (*calls predict_growth*) At 600 hours: **10.2 failures** (95% CI: 8.1–12.7). At 800 hours: **12.9** (CI: 10.1–16.5). At 1000 hours: **15.4** (CI: 11.8–20.1). The growth trend is still active ($\hat{\lambda} < 1$). If your target is fewer than 15 cumulative failures at 1000 hours, you are on track — but the upper confidence bound exceeds 15, so additional test time would tighten that uncertainty.

Example 3 — Interactive MCF plot for stakeholders:

You: I have a fleet of 3 pumps. IDs: 1,1,1,2,2,3,3. Times: 310,850,1620,420,1050,580,1890. All events are failures. End times: 3000,3000,3000. Generate an MCF plot I can share with the maintenance team.

Claude: (*calls plot_mcf*) Done. The interactive MCF plot has been saved to `/var/folders/.../mcf_abc123.html`. Open that file in any web browser to explore it — hover over any point to see the exact expected cumulative failures per system. The MCF is concave upward across the fleet, indicating that failure rate is increasing with age; a preventive maintenance interval may be warranted.

10.9 When to Use AI vs Scripted R

| Situation | AI + MCP | Scripted R |
|--|----------|------------|
| Quick one-off fit during a conversation | | |
| Explaining results to non-R stakeholders | | |
| Iterating on model choice interactively | | |
| Automating repeated batch analyses | | |
| Reproducible research / version control | | |
| Full custom plot formatting | | |
| Embedding results in a Quarto report | | |
| Large datasets or performance-sensitive work | | |

The AI + MCP path excels at conversational exploration, where you want to ask a question, get an answer in plain language, and then ask a follow-up. Scripted R remains the right choice for production pipelines and fully reproducible analyses.

10.10 Getting Help

```
help(package = "ReliaGrowR")  
help(package = "ReliaPlotR")
```

- mcptools GitHub: <https://github.com/posit-dev/mcptools>
- ellmer GitHub: <https://github.com/tidyverse/ellmer>
- MCP specification: <https://modelcontextprotocol.io>

10.11 Summary

MCP lets an AI assistant call ReliaGrowR and ReliaPlotR functions directly during a conversation. The `reliagrow_mcp_tools()` function defines seven tools, Duane fit, Crow-AMSAA fit, piecewise NHPP, growth forecast, NHPP for repairable systems, MCF, and RDT planning, that each return plain JSON results. The `reliaplotr_mcp_tools()` function adds four plot tools that save self-contained interactive HTML files. Register the server with Claude Desktop via `claude_desktop_config.json` or with Claude Code via `claude mcp add`, then describe your data in plain language and let the AI run the analysis.

Appendix: R Quick Reference

This appendix lists every R package and function used in this book, organized by package, with the chapter(s) where each appears. Use it as a lookup reference when you want to find the right function quickly without having to search through the chapters.

Install all packages at once with:

```
install.packages(c(
  "ReliaLearnR",
  "WeibullR",
  "WeibullR.ALT",
  "ReliaGrowR",
  "DiagrammeR",
  "ReliaPlotR",
  "ReliaShiny"
))
```

ReliaLearnR

RAM (Reliability, Availability, and Maintainability) calculation helpers. All five functions accept scalar or vector inputs and return a single numeric scalar.

```
install.packages("ReliaLearnR")
library(ReliaLearnR)
```

| Function | Description | Chapter |
|---|---|---------|
| <code>rel(outageTime, totalTime)</code> | Reliability: $1 - \text{outage}/\text{total}$ | Ch. 1 |

| Function | Description | Chapter |
|--|--|---------|
| <code>avail(unavailTime, totalTime)</code> | Availability:
1 – unavailable/total
(failures + maintenance) | Ch. 1 |
| <code>mttf(failures, totalTime)</code> | Mean Time to Failure for
non-repairable items | Ch. 1 |
| <code>mtbf(failures, totalTime)</code> | Mean Time Between Failures
for repairable systems | Ch. 1 |
| <code>fr(failures, totalTime)</code> | Failure rate
$\lambda = \text{failures}/\text{total time}$ | Ch. 1 |

Example usage:

```
library(ReliaLearnR)
```

```
rel(outageTime = 10, totalTime = 5 * 365) # reliability over 5 years
```

```
[1] 0.9945205
```

```
avail(unavailTime = 25, totalTime = 5 * 365) # availability with maintenance
```

```
[1] 0.9863014
```

```
mttf(failures = 5, totalTime = 100 * 5) # MTTF for a fleet of 100 units
```

```
[1] 100
```

```
mtbf(failures = 5, totalTime = 45000) # MTBF for a repairable machine
```

```
[1] 9000
```

```
fr(failures = 75, totalTime = 100 * 5000) # failure rate (failures per  
↪ hour)
```

```
[1] 0.00015
```

WeibullR

Life data analysis: Weibull probability plotting, distribution fitting, and confidence bounds.

```
install.packages("WeibullR")
library(WeibullR)
```

| Function | Description | Chapter |
|----------------------------------|--|----------|
| <code>wblr(...)</code> | Create a Weibull object from time/event data | Ch. 3, 7 |
| <code>wblr.fit(obj, ...)</code> | Fit a distribution (MLE or MRR) to a <code>wblr</code> object | Ch. 3, 7 |
| <code>wblr.conf(obj, ...)</code> | Compute confidence bounds on a fitted <code>wblr</code> object | Ch. 3, 7 |
| <code>MLEw2p(...)</code> | Fit a 2-parameter Weibull distribution via MLE directly | Ch. 3 |
| <code>plot_contour(obj)</code> | Static likelihood ratio contour plot | Ch. 3 |

Example usage:

```
library(WeibullR)

failures <- c(500, 1200, 900, 1500, 750)
da <- wblr(x = failures, s = c(2000, 2000)) # 2 suspensions
da <- wblr.fit(da, method.fit = "mle")
da <- wblr.conf(da, method.conf = "lrb")
plot(da)
```

WeibullR.ALT

Accelerated Life Testing (ALT): fit life-stress relationships such as Arrhenius, Power Law, and Eyring to multi-stress datasets.

```
install.packages("WeibullR.ALT")
library(WeibullR.ALT)
```

| Function | Description | Chapter |
|--------------------------------|--|----------|
| <code>NelsonData(name)</code> | Load built-in Nelson ALT example datasets | Ch. 5, 7 |
| <code>alt.data(...)</code> | Prepare an ALT data object from raw time/event/stress data | Ch. 5, 7 |
| <code>alt.make(...)</code> | Create an ALT model object (specify distribution and relationship) | Ch. 5, 7 |
| <code>alt.parallel(obj)</code> | Constrain Weibull slopes to be equal across stress levels | Ch. 5, 7 |
| <code>alt.fit(obj, ...)</code> | Fit the life-stress relationship to the ALT model | Ch. 5, 7 |

Example usage:

```
library(WeibullR.ALT)

nd <- NelsonData("E2")
da <- alt.data(nd)
mo <- alt.make(da, distribution = "Weibull", relationship = "Arrhenius")
mo <- alt.parallel(mo)
mo <- alt.fit(mo)
plot(mo)
```

ReliaGrowR

Reliability growth testing and repairable systems analysis: Crow-AMSAA, Duane, Power Law Process, and Mean Cumulative Function.

```
install.packages("ReliaGrowR")
library(ReliaGrowR)
```

| Function | Description | Chapter |
|------------------------------------|---|----------|
| <code>rdt(...)</code> | Plan a Reliability Demonstration Test | Ch. 4 |
| <code>rga(times, ...)</code> | Fit a Crow-AMSAA (NHPP Power Law) reliability growth model | Ch. 4, 7 |
| <code>predict_rga(obj, ...)</code> | Forecast cumulative failures from a <code>rga</code> object | Ch. 4 |
| <code>duane(times, ...)</code> | Fit a Duane reliability growth model | Ch. 4, 7 |
| <code>nhpp(...)</code> | Fit a Power Law Process to repairable systems data | Ch. 6, 7 |
| <code>mcf(...)</code> | Estimate the Mean Cumulative Function (Nelson-Aalen) | Ch. 6, 7 |
| <code>exposure(...)</code> | Compute fleet exposure from multi-system event data | Ch. 6, 7 |

Example usage:

```
library(ReliaGrowR)

# Reliability growth
times <- c(9.2, 25, 61.5, 260, 300, 710, 916, 1010, 1220)
rga_fit <- rga(times, T = 1500)
plot(rga_fit)

# Repairable systems
events <- data.frame(id = c(1,1,1,2,2), time = c(100,300,500,200,450))
mcf_fit <- mcf(events)
plot(mcf_fit)
```

DiagrammeR

Graph and diagram visualisation: render Graphviz DOT diagrams for reliability block diagrams and fault trees.

```
install.packages("DiagrammeR")
library(DiagrammeR)
```

| Function | Description | Chapter |
|-----------------------------|--|---------|
| <code>grViz(diagram)</code> | Render a Graphviz DOT string as an SVG diagram | Ch. 2 |

Example usage:

```
library(DiagrammeR)

grViz("
  digraph series {
    rankdir = LR
    node [shape = rectangle]
    A -> B -> C
  }
")
```

ReliaPlotR

Interactive, plotly-based versions of every static plot produced by WeibullR, WeibullR.ALT, and ReliaGrowR. Drop-in replacements that return `plotly` objects embeddable in Quarto documents and Shiny apps.

```
install.packages("ReliaPlotR")
library(ReliaPlotR)
```

| Function | Description | Chapter |
|----------------------------------|---|---------|
| <code>plotly_wblr(obj)</code> | Interactive Weibull probability plot | Ch. 7 |
| <code>plotly_contour(obj)</code> | Interactive likelihood ratio contour plot | Ch. 7 |

| Function | Description | Chapter |
|-----------------------------------|---|---------|
| <code>plotly_duane(obj)</code> | Interactive Duane reliability growth plot | Ch. 7 |
| <code>plotly_rga(obj)</code> | Interactive Crow-AMSAA plot | Ch. 7 |
| <code>plotly_alt(obj)</code> | Interactive ALT probability plot | Ch. 7 |
| <code>plotly_rel(obj)</code> | Interactive life-stress relationship plot | Ch. 7 |
| <code>plotly_nhpp(obj)</code> | Interactive NHPP intensity (recurrence rate) plot | Ch. 7 |
| <code>plotly_mcf(obj)</code> | Interactive Mean Cumulative Function plot | Ch. 7 |
| <code>plotly_exposure(obj)</code> | Interactive fleet exposure plot | Ch. 7 |

Example usage:

```
library(ReliaPlotR)
library>WeibullR)

da <- wblr(x = c(500, 1200, 900, 1500, 750))
da <- wblr.fit(da, method.fit = "mle")
da <- wblr.conf(da, method.conf = "lrb")
plotly_wblr(da) # interactive version of plot(da)
```

ReliaShiny

A point-and-click Shiny web application that wraps all of the above analyses in a browser-based UI; no coding required. Modules cover RAM, Life Data Analysis, Reliability Growth, ALT, Repairable Systems, and RBD.

```
install.packages("ReliaShiny")
library(ReliaShiny)
```

| Function | Description | Chapter |
|--------------|--|---------|
| ReliaShiny() | Launch the ReliaShiny web application in the default browser | Ch. 8 |

Example usage:

```
library(ReliaShiny)
ReliaShiny()
```

Glossary and Formula Reference

A quick-reference guide to the key symbols and formulas used throughout this book. Each entry links back to the chapter where it is first introduced. Symbols are listed in order of first introduction.

Key Symbols

| Symbol | Name | Definition | First used |
|-----------|---------------------------------------|---|---------------------------|
| β | Shape parameter | Controls whether the failure rate increases, decreases, or stays constant over time | Chapter 1 |
| η | Scale parameter (characteristic life) | Time at which $F(t) = 63.2\%$ for any β | Chapter 3 |
| λ | Failure rate | Failures per unit time (constant rate, exponential model) | Chapter 1 |
| $\rho(t)$ | Intensity / recurrence rate | Time-varying failure rate for repairable systems | Chapter 6 |
| $R(t)$ | Reliability function | Probability of survival to time t | Chapter 1 |
| $F(t)$ | Cumulative distribution function | Probability of failure by time t ; $F(t) = 1 - R(t)$ | Chapter 3 |

| Symbol | Name | Definition | First used |
|--------|----------------------------|--|------------|
| MTTF | Mean Time To Failure | Expected life for non-repairable items | Chapter 1 |
| MTBF | Mean Time Between Failures | Expected time between failures for repairable systems | Chapter 1 |
| MTTR | Mean Time To Repair | Expected time to restore a failed system | Chapter 1 |
| AF | Acceleration Factor | Ratio of life at use conditions to life at elevated stress | Chapter 5 |
| E_a | Activation energy | Energy barrier for thermally activated failure (eV) | Chapter 5 |
| k | Boltzmann constant | 8.617×10^{-5} eV/K | Chapter 5 |
| n | Stress exponent | Sensitivity of life to non-thermal stress (Power Law) | Chapter 5 |
| t_0 | Failure-free period | Minimum time before any failure can occur (3-parameter Weibull) | Chapter 3 |
| AD | Anderson-Darling statistic | Goodness-of-fit measure; lower values indicate a better fit | Chapter 3 |
| MCF | Mean Cumulative Function | Expected cumulative failures per system over time (non-parametric) | Chapter 6 |

| Symbol | Name | Definition | First used |
|----------------------|--------------------|--|------------|
| β_{CCF} | CCF
beta-factor | Fraction of total failure rate attributable to common cause failures | Chapter 2 |

Key Formulas

RAM (Chapter 1)

$$R = 1 - \frac{\text{failed time}}{\text{total time}}$$

$$A = 1 - \frac{\text{unavailable time}}{\text{total time}} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

$$\lambda = \frac{\text{failures}}{\text{total time}}, \quad \text{MTTF} = \frac{1}{\lambda}, \quad \text{MTBF} = \frac{1}{\lambda}$$

$$R(t) = e^{-\lambda t} \quad (\text{exponential model})$$

B_n life : time t such that $F(t) = n/100$

Weibull Distribution (Chapter 3)

$$R(t) = \exp\left[-\left(\frac{t}{\eta}\right)^\beta\right], \quad F(t) = 1 - R(t)$$

$$F(\eta) = 1 - e^{-1} \approx 63.2\% \quad \text{for any } \beta$$

3-parameter Weibull with failure-free period t_0 :

$$R(t) = \exp\left[-\left(\frac{t - t_0}{\eta}\right)^\beta\right]$$

Reliability Block Diagrams (Chapter 2)

$$R_{\text{series}} = \prod_{i=1}^n R_i$$

$$R_{\text{parallel}} = 1 - \prod_{i=1}^n (1 - R_i)$$

$$R_{\text{k-out-of-n}} = \sum_{i=k}^n \binom{n}{i} R^i (1 - R)^{n-i}$$

Beta-factor CCF model:

$$\lambda_{\text{CCF}} = \beta_{\text{CCF}} \cdot \lambda_{\text{total}}, \quad \lambda_{\text{ind}} = (1 - \beta_{\text{CCF}}) \cdot \lambda_{\text{total}}$$

Accelerated Life Testing (Chapter 5)

Arrhenius (temperature):

$$AF = \exp\left[\frac{E_a}{k} \left(\frac{1}{T_{\text{use}}} - \frac{1}{T_{\text{stress}}}\right)\right]$$

Power Law (non-thermal stress):

$$AF = \left(\frac{S_{\text{stress}}}{S_{\text{use}}}\right)^n$$

$$\text{Life}_{\text{use}} = AF \times \text{Life}_{\text{stress}}$$

Reliability Growth Analysis (Chapter 4)

Duane cumulative MTBF (K = scale parameter estimated from data):

$$\text{CMTBF}(t) = K \cdot t^{\beta-1}$$

Crow-AMSAA cumulative failures (λ_0 = scale parameter; β = shape parameter):

$$N(t) = \lambda_0 \cdot t^\beta$$

| β (Crow-AMSAA) | Meaning |
|----------------------|--|
| < 1 | Failures decreasing, reliability improving |
| $= 1$ | Constant rate, stable |
| > 1 | Failures increasing, reliability worsening |

Warning

Note: Crow-AMSAA β and Duane β have **opposite** interpretations: Duane $\beta > 1$ means *improving*, Crow-AMSAA $\beta > 1$ means *worsening*. See Chapter 4 for details.

Repairable Systems (Chapter 6)

Power Law intensity (recurrence rate):

$$\rho(t) = \lambda\beta t^{\beta-1}$$

Instantaneous MTBF at time t :

$$\text{MTBF}(t) = \frac{1}{\rho(t)}$$

Abernethy, Robert B. 2004. *The New Weibull Handbook*. 5th ed. R.B. Abernethy.

Allaire, JJ. 2024. *Quarto: R Interface to Quarto Markdown Publishing System*. <https://CRAN.R-project.org/package=quarto>.

- Allaire, JJ, Charles Teague, Carlos Scheidegger, Yihui Xie, and Christophe Dervieux. 2024. *Quarto*. <https://doi.org/10.5281/zenodo.5960048>.
- Ascher, Harold, and Harry Feingold. 1984. *Repairable Systems Reliability*. Marcel Dekker.
- Billinton, Roy, and Ronald N. Allan. 1992. *Reliability Evaluation of Engineering Systems*. 2nd ed. Plenum Press.
- Chang, Winston, Joe Cheng, JJ Allaire, et al. 2022. *Shiny: Web Application Framework for r*. <https://doi.org/10.32614/CRAN.package.shiny>.
- Crow, Larry H. 1975. *Reliability Analysis for Complex, Repairable Systems*. Army Material Systems Analysis Activity. <https://apps.dtic.mil/sti/citations/ADA020296>.
- Duane, J. T. 1964. “Learning Curve Approach to Reliability Monitoring.” *IEEE Transactions on Aerospace* 2: 563–66. <https://doi.org/10.1109/TA.1964.4319640>.
- Govan, Paul. 2023a. *ReliaPlotR: Interactive Reliability Analysis Plots*. <https://doi.org/10.32614/CRAN.package.ReliasPlotR>.
- Govan, Paul. 2023b. *ReliaShiny: A 'Shiny' App for Reliability Analysis*. <https://doi.org/10.32614/CRAN.package.ReliasShiny>.
- Govan, Paul. 2024. *ReliaGrowR: Reliability Growth Analysis*. <https://doi.org/10.32614/CRAN.package.ReliasGrowR>.
- Guo, H., A. Mettas, G. Sarakakis, and P. Niu. 2010. “Piecewise NHPP Models with Maximum Likelihood Estimation for Repairable Systems.” *2010 Proceedings – Annual Reliability and Maintainability Symposium (RAMS)*, 1–7. <https://doi.org/10.1109/RAMS.2010.5448029>.
- Meeker, William Q., and Luis A. Escobar. 1998. *Statistical Methods for Reliability Data*. Wiley-Interscience.
- Nelson, Wayne. 1990. *Accelerated Testing*. Wiley-Interscience.
- Posit PBC. 2025. *Mcptools: Model Context Protocol Tools for r*. <https://github.com/posit-dev/mcptools>.
- Silkworth, David. 2020. “WeibullR: An r Package for Weibull Analysis for Reliability Engineers.” *10th International Symposium on Data Science and Its Applications (ISDSA2019)*, 43–53. <https://doi.org/10.35566/isdsa2019c3>.
- Silkworth, David. 2022. *WeibullR.ALT: Accelerated Life Testing Using 'WeibullR'*. <https://doi.org/10.32614/CRAN.package.WeibullR.ALT>.

[//doi.org/10.32614/CRAN.package.WeibullR.ALT](https://doi.org/10.32614/CRAN.package.WeibullR.ALT).

Silkworth, David, and Jurgen Symynck. 2022. *WeibullR: Weibull Analysis for Reliability Engineering*. <https://doi.org/10.32614/CRAN.package.WeibullR>.

Vesely, W., F. Goldberg, N. Roberts, and D. Haasl. 1981. *Fault Tree Handbook*. NUREG-0492. US Nuclear Regulatory Commission.

Wickham, Hadley, and Joe Cheng. 2025. *Ellmer: Chat with Large Language Models*. <https://github.com/tidyverse/ellmer>.