9 Time-Series Analysis

R. Heller · Date: 2026-05-11 · Last changed: 2026-10-30

Stationarity, ARIMA, exponential smoothing, state-space models, spectral analysis, and forecasting evaluation. Examples use biomedical signals — heart rate, glucose, EEG — rather than macroeconomic data.

This chapter contains **30 method pages** and **1 lab**. If you are not sure which method to read, return to Chapter 0 and follow the decision tree to the right node.

9.1 Method pages

Method	Source slug
ARIMA Models	`arima-models`
Augmented Dickey-Fuller Test	`adf-test`
Bayesian Changepoint Detection	`bcp-bayesian-changepoint`
Changepoint Detection	`changepoint-detection`
Diebold-Mariano Test	`diebold-mariano`
Differencing a Time Series	`differencing`
ETS Models	`ets-models`
Exponential Smoothing	`exponential-smoothing`
Forecast Accuracy Metrics	`forecast-accuracy-metrics`
GARCH Models	`garch-volatility`
Granger Causality	`granger-causality`
Holt-Winters Method	`holt-winters`
Interpreting ACF and PACF	`acf-pacf-interpretation`
KPSS Test	`kpss-test`
Moving Averages	`moving-averages`
Phillips-Perron Test	`phillips-perron`
Rolling-Origin Cross-Validation	`rolling-origin-cv`
Seasonal ARIMA (SARIMA)	`seasonal-arima`
Seasonal Decomposition (STL)	`seasonal-decomposition-stl`
Spectral Analysis	`spectral-analysis`
State-Space Models	`state-space-models`
Stationarity Tests	`stationarity-tests`
The Kalman Filter	`kalman-filter`
The Ljung-Box Test	`ljung-box`
Time Series: Introduction	`time-series-introduction`

Method	Source slug
VECM and Cointegration	`vecm-cointegration`
Vector Autoregression (VAR)	`multivariate-var`
Wavelet Analysis	`wavelet-analysis`
White Noise Tests	`white-noise-tests`
X-13ARIMA-SEATS Decomposition	`x13-decomposition`

9.2 Labs

Lab
Time series basics

9.3 Interpreting ACF and PACF

9.3.1 Introduction

The autocorrelation function (ACF) and partial autocorrelation function (PACF) are the foundational diagnostic tools for ARIMA model identification. The Box-Jenkins approach uses their characteristic patterns to guess plausible orders for AR and MA terms before fitting. Even with modern automated tools (`auto.arima`), reading the ACF and PACF remains an essential skill for sanity-checking automated choices, identifying seasonal structure, and diagnosing residuals.

9.3.2 Prerequisites

A working understanding of autoregressive (AR) and moving-average (MA) processes, the difference between correlation and partial correlation, and stationarity as a precondition for meaningful ACF interpretation.

9.3.3 Theory

The ACF ρ_k measures correlation between y_t and y_{t-k}. The PACF ϕ_{kk} measures correlation between y_t and y_{t-k} after removing the effects of shorter lags $1, 2, \ldots, k-1$. Box-Jenkins identification rules:

Process	ACF	PACF
AR(p)	Tails off (decays slowly)	Cuts off at lag p
MA(q)	Cuts off at lag q	Tails off
ARMA(p, q)	Tails off	Tails off

The dashed lines in R's ACF plot show approximate 95 % confidence bands ($\pm 1.96/\sqrt{n}$) under a white-noise null; values outside the bands are statistically significant at the 5 % level.

For seasonal data, expect spikes at multiples of the seasonal period; multiplicative seasonal patterns produce decaying spikes at $s, 2s, 3s, \ldots$ in the ACF.

9.3.4 Assumptions

The series is stationary; if not, difference first until it becomes stationary, then read the ACF / PACF on the differenced series.

9.3.5 R Implementation

```r
library(forecast)

set.seed(2026)
ar_series ← arima.sim(list(ar = 0.7), n = 200)
ma_series ← arima.sim(list(ma = -0.5), n = 200)

par(mfrow = c(2, 2))
acf(ar_series, main = "ACF of AR(1)")
pacf(ar_series, main = "PACF of AR(1)")
acf(ma_series, main = "ACF of MA(1)")
pacf(ma_series, main = "PACF of MA(1)")
par(mfrow = c(1, 1))

ggtsdisplay(ar_series)
```

9.3.6 Output & Results

The four-panel display shows characteristic Box-Jenkins patterns: AR(1) ACF tails off geometrically while its PACF cuts off after lag 1; MA(1) ACF cuts off after lag 1 while its PACF tails off. `ggtsdisplay()` integrates the time-series plot with ACF and PACF in a single ggplot-style view.

9.3.7 Interpretation

A reporting sentence: "The ACF of the simulated AR(1) tailed off geometrically (lag-1 correlation 0.69, decaying smoothly); the PACF cut off sharply after lag 1 (0.69 at lag 1, well within bands at lags 2 onward), consistent with the AR(1) data-generating process. The MA(1) showed the reverse pattern, confirming Box-Jenkins identification rules." Identify the model class from the patterns before fitting any model.

9.3.8 Practical Tips

- Always difference the series until stationary before reading ACF / PACF on the levels; non-stationary series produce decaying ACFs that mimic AR.
- Use `forecast::ggtsdisplay()` for a combined time-series + ACF + PACF view; faster than three separate plots.

- Significance bands assume a white-noise null; after model fitting, the residual ACF is the key diagnostic and should be approximately within bands.
- Seasonal patterns appear at lags equal to the seasonal period; spikes at $s, 2s, 3s$ on the ACF indicate seasonal AR/MA structure.
- Complex ACF / PACF patterns (mixed ARMA, seasonal-plus-non-seasonal) are difficult to identify by eye; reach for `auto.arima` and verify the chosen orders against the ACF.
- For very long series, even tiny correlations are statistically significant; focus on the magnitude and pattern, not on which lags fall just outside the bands.

9.3.9 R Packages Used

Base R `acf()` and `pacf()`; `forecast::ggtsdisplay()` for ggplot-style integrated diagnostics; `feasts::ACF()`, `PACF()`, `gg_tsdisplay()` for the modern tidyverts equivalent; `astsa::acf2()` for compact two-panel display.

9.3.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.3.11 See also — labs in this chapter

- Time series basics

9.4 Augmented Dickey-Fuller Test

9.4.1 Introduction

The augmented Dickey-Fuller (ADF) test is the canonical test for unit roots in time series. A unit root corresponds to random-walk-like non-stationarity in which shocks have permanent effects on the level of the series; standard regression and ARIMA(p, 0, q) models give nonsense results when applied to such data. The ADF test asks whether a unit root is present and, if rejected, supports modelling the series as stationary (or trend-stationary). Pairing ADF with KPSS gives complementary information: the two tests have flipped null hypotheses, and agreement strengthens the verdict.

9.4.2 Prerequisites

A working understanding of stationarity, autoregressive models, the difference between mean-reverting and random-walk processes, and the role of differencing in inducing stationarity.

9.4.3 Theory

The ADF test fits the regression

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \sum_{i=1}^{p} \delta_i \Delta y_{t-i} + \varepsilon_t,$$

and tests $\gamma = 0$ (unit root) against $\gamma < 0$ (stationary). The "augmented" p lagged differences absorb autocorrelation in the residuals and validate the test under more general AR processes than the original Dickey-Fuller. Under the unit-root null, the test statistic does not follow a standard normal distribution; tabulated Dickey-Fuller critical values are used.

Three deterministic specifications are common: no intercept (rare), intercept only ("drift"), and intercept plus linear trend. The choice should reflect the visible behaviour of the series.

9.4.4 Assumptions

A linear autoregressive structure; no structural breaks (which invalidate the test); the lag order p is large enough to whiten the residuals.

9.4.5 R Implementation

```
library(tseries); library(urca)

x ← cumsum(rnorm(100))     # random walk = unit root

adf.test(x)                # default lag order
adf.test(diff(x))          # stationary after differencing

# urca version with formal output
summary(ur.df(x, type = "drift", selectlags = "AIC"))
```

9.4.6 Output & Results

`adf.test()` returns the ADF statistic and a p-value computed from the Dickey-Fuller distribution. `urca::ur.df()` provides richer output with critical values at multiple significance levels and automatic lag selection by AIC or BIC.

9.4.7 Interpretation

A reporting sentence: "The ADF test on the random-walk series gave a statistic of -1.50 ($p = 0.52$, fail to reject unit root); after first-differencing, ADF gave -9.83 ($p < 0.01$, reject unit root in favour of stationarity), confirming the series is integrated of order 1." Always state the deterministic specification (drift, trend, none) and the lag order chosen.

9.4.8 Practical Tips

- Include an intercept (`type = "drift"`) for series that fluctuate around a non-zero mean; add a trend (`type = "trend"`) for series with a visible linear trend.
- Use `selectlags = "AIC"` or `"BIC"` in `urca::ur.df()` for automatic lag selection; the default in `tseries::adf.test()` is `trunc(...)` which is sometimes too small.
- Structural breaks invalidate ADF; if a break is plausible, use Zivot-Andrews (`urca::ur.za()`) which extends the test to allow one endogenous break.
- Pair with KPSS for the complementary perspective: agreement (ADF rejects, KPSS does not) confirms stationarity; the reverse confirms a unit root.
- Results are sensitive to specification; try several deterministic options and lag orders, and report a robust summary if conclusions hold across them.
- For multi-series unit-root testing on panel data, use Im-Pesaran-Shin (`plm::purtest`) or Levin-Lin-Chu.

9.4.9 R Packages Used

`tseries::adf.test()` for the canonical ADF; `urca::ur.df()` for richer output and lag-selection tools; `urca::ur.za()` for the Zivot-Andrews test with structural breaks; `forecast::ndiffs()` for an integrated workflow that determines the required differencing order from KPSS / ADF / Phillips-Perron.

9.4.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.4.11 See also — labs in this chapter

- Time series basics

9.5 ARIMA Models

9.5.1 Introduction

ARIMA models are the classical workhorse of time-series forecasting. They combine three ideas: autoregression (the current value depends on lagged values), differencing (the current value depends on changes rather than levels), and moving averages of the errors. An ARIMA model is specified by three integers (p, d, q): the order of autoregression, the order of differencing, and the order of moving averages. With a seasonal extension (SARIMA), the same model captures weekly, monthly, or annual cycles.

9.5.2 Prerequisites

The reader should understand the concepts of a stationary time series, autocorrelation, and white noise, and should have seen an ACF/PACF plot before.

9.5.3 Theory

An ARMA(p, q) model for a stationary series Y_t is

$$Y_t = c + \phi_1 Y_{t-1} + \ldots + \phi_p Y_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \ldots + \theta_q \varepsilon_{t-q},$$

where ε_t is white noise. When the series is non-stationary, differencing is applied: $\Delta Y_t = Y_t - Y_{t-1}$, or more generally $\Delta^d Y_t$. An ARIMA(p, d, q) model is an ARMA(p, q) model fit to the d-th-order differenced series.

For seasonal data with period s, SARIMA$(p, d, q)(P, D, Q)_s$ adds seasonal autoregressive, differencing, and moving-average terms at lag s. A monthly retail sales series might be well described by a SARIMA$(1, 1, 1)(0, 1, 1)_{12}$.

Model identification traditionally follows the Box-Jenkins methodology:

1. Check stationarity; difference until stationary.
2. Inspect the ACF and PACF of the differenced series; the decay patterns suggest p and q.
3. Fit candidate models and choose by AIC/BIC.
4. Diagnose residuals: they should be white noise.

In practice, `auto.arima()` automates steps 1-3 and reliably finds a good model.

9.5.4 Assumptions

ARIMA models assume:

1. The differenced series is stationary (constant mean, constant variance, autocorrelation depending only on lag).
2. Errors are uncorrelated and have constant variance (homoscedastic).
3. Errors are approximately normally distributed (for prediction-interval validity).

Violations of normality affect interval coverage but not point forecasts; violations of stationarity invalidate the whole model.

9.5.5 R Implementation

The `forecast` package provides the classical interface; the newer `fable` package integrates with the tidyverse.

```r
library(forecast)

data(AirPassengers)
plot(AirPassengers)

fit <- auto.arima(AirPassengers)
summary(fit)

checkresiduals(fit)

fc <- forecast(fit, h = 24)
autoplot(fc) +
  labs(x = "Year", y = "Air passengers (thousands)",
       title = "24-month ahead forecast with 80/95% PIs")
```

`auto.arima()` searches over a range of $(p, d, q)(P, D, Q)_{12}$ specifications and returns the best by corrected AIC. `checkresiduals()` plots the residual time series, ACF, histogram, and runs a Ljung-Box test for remaining autocorrelation.

9.5.6 Output & Results

For the `AirPassengers` series, `auto.arima()` selects an ARIMA$(0,1,1)(0,1,1)_{12}$ model on the log scale (with automatic Box-Cox transformation). The Ljung-Box test on residuals gives $p \approx 0.5$, indicating no remaining autocorrelation. The forecast plot shows point predictions with 80% and 95% prediction intervals that fan out over the 24-month horizon.

9.5.7 Interpretation

The model says that after seasonal and regular differencing on the log scale, the series is well described by a simple MA(1) structure in both the regular and seasonal dimensions. The growing width of the prediction intervals reflects accumulating uncertainty – a feature, not a bug, of any honest forecast.

9.5.8 Practical Tips

- Always inspect the series visually before and after differencing; automated tools can miss obvious issues like level shifts or outlier clusters.
- Use information criteria (AICc preferred for small samples) for model comparison, not p-values.
- Validate forecasts on a held-out tail of the series (train/test split in time), not by in-sample fit.
- If the series shows clear seasonality, try both SARIMA and ETS (`ets()`) and let cross-validated accuracy choose.
- For very long series with complex seasonality, consider `prophet`, `fable::fable`, or structural time-series models instead of ARIMA.

9.5.9 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.5.10 See also — labs in this chapter

- Time series basics

9.6 Bayesian Changepoint Detection

9.6.1 Introduction

Bayesian changepoint detection returns a posterior probability of a changepoint at each time, rather than a hard list of detected change times. The richer output quantifies uncertainty about both the existence and the location of changes, making it preferable to classical methods (PELT, binary segmentation) when the analyst wants to communicate "the signal probably shifted near time 200, with some uncertainty about the exact time" rather than committing to an arbitrary threshold. Bayesian approaches also support online detection as data stream in.

9.6.2 Prerequisites

A working understanding of Bayesian inference, classical changepoint detection, and the difference between offline (batch) and online (streaming) analysis.

9.6.3 Theory

Offline Bayesian changepoint detection (Barry-Hartigan 1993, Fearnhead 2006) places a prior on the number and locations of changepoints plus segment parameters and samples from the joint posterior via Gibbs sampling, particle filters, or recursive forward-backward algorithms. The output is a marginal posterior probability of changepoint at each time and posterior summaries of segment parameters.

Online Bayesian changepoint detection (BOCD) (Adams & MacKay 2007) computes the run-length distribution — the posterior probability that the time since the last changepoint equals r — recursively as each new data point arrives. The run-length distribution updates in $O(t)$ per step, making the method viable for streaming applications.

9.6.4 Assumptions

Segment-parameter priors are chosen appropriately for the data; conditional independence between segments given the changepoint structure; the chosen segment model (Normal mean, regression coefficients) matches the underlying signal.

9.6.5 R Implementation

```r
library(bcp)

set.seed(2026)
x <- c(rnorm(100, 0, 1), rnorm(100, 3, 1), rnorm(100, -1, 1))

fit <- bcp(x)
plot(fit)
fit$posterior.prob   # per-time changepoint probability
```

9.6.6 Output & Results

`bcp()` returns the posterior changepoint probability at each time and the posterior mean of the segmented signal. The plot displays both — the smooth signal estimate above and the posterior probability below — making it easy to identify clearly supported changepoints (probability spikes near 1) versus uncertain candidates.

9.6.7 Interpretation

A reporting sentence: "Bayesian changepoint detection returned posterior changepoint probabilities exceeding 0.95 at times 100 and 200, recovering the simulated three-segment structure; the second changepoint had a slightly tighter posterior distribution (probability mass concentrated within ± 1 time step) than the first, reflecting the larger between-segment mean difference at that point." Communicate posterior probabilities, not hard threshold-based detections.

9.6.8 Practical Tips

- Tune the hyperparameter `w0` (prior on between-changepoint variance) for stability; larger values favour fewer, sharper changepoints.
- Online variants (`ocp` package) implement BOCD for real-time detection; useful in monitoring applications where decisions must be made as data arrive.
- Visualise the posterior probability time series rather than extracting hard changepoints by thresholding; thresholding throws away the uncertainty information that motivates the Bayesian approach.
- Small samples give diffuse posteriors; in tight cases, the posterior probability at the true changepoint may peak at only 0.5 to 0.7. More data sharpens the posterior.
- For multivariate changepoint detection, `MultiBNPRiver` and `mbcp` extend the framework to vector-valued series, at substantially higher computational cost.
- Compare against classical methods (PELT) on the same data; Bayesian methods are richer in output but classical methods are typically much faster.

9.6.9 R Packages Used

`bcp` for the canonical offline Barry-Hartigan implementation; `ocp` for online Bayesian changepoint detection; `mcp` for Bayesian changepoint regression with explicit segment models; `bnpa` for Bayesian non-parametric changepoint detection; `Rbeast` for Bayesian time-series decomposition with changepoint awareness.

9.6.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.6.11 See also — labs in this chapter

- Time series basics

9.7 Changepoint Detection

9.7.1 Introduction

Changepoint detection identifies the times at which the statistical properties of a time series — mean, variance, regression coefficients, or full distribution — change abruptly. Applications range from monitoring physiological signals (heart-rate or EEG state changes), to detecting regime shifts in financial and economic series, to segmenting long sensor recordings into homogeneous epochs. The output is a list of changepoint times together with the within-segment summaries between them; the methodological choices concern the change type, the detection algorithm, and the penalty controlling sensitivity.

9.7.2 Prerequisites

A working understanding of hypothesis testing, segmentation, and the trade-off between detecting true changepoints and false positives.

9.7.3 Theory

Three algorithm classes dominate:

- **Binary segmentation**: recursively detect a single changepoint, partition the series, and apply the same procedure to each segment. Fast but greedy; can miss configurations of multiple nearby changepoints.
- **PELT** (Pruned Exact Linear Time): finds the optimal segmentation under a penalty in $O(n)$ when the penalty satisfies a regularity condition. The standard modern method.
- **Segment Neighbourhood**: dynamic programming for the optimal partition under a fixed maximum number of changepoints.

The penalty (BIC, MBIC, Hannan-Quinn, or AIC) controls how many changepoints are reported; a larger penalty yields fewer changepoints. Different change types have different test statistics: mean changes use the likelihood-ratio test under a Normal model; variance changes use the same with σ as the parameter; meanvar combines both; regression changepoints generalise to coefficient breaks.

9.7.4 Assumptions

Independent observations within segments (or correctly specified within-segment dependence structure); the change type matches the underlying signal (mean shifts vs variance shifts vs regression breaks).

9.7.5 R Implementation

```r
library(changepoint)

set.seed(2026)
x ← c(rnorm(100, 0, 1), rnorm(100, 3, 1), rnorm(100, -1, 1))

# Mean changes via PELT
cp ← cpt.mean(x, method = "PELT", penalty = "BIC")
cpts(cp)
plot(cp)

# Variance changes
cp_v ← cpt.var(x, method = "PELT")
cpts(cp_v)
```

9.7.6 Output & Results

`cpt.mean()` returns a `cpt` object with the changepoint times accessible via `cpts()`, the within-segment means via `param.est()`, and a plot showing the detected segments. The simulated three-segment structure is recovered with changepoints at times 100 and 200.

9.7.7 Interpretation

A reporting sentence: “PELT with BIC penalty detected two changepoints at times 100 and 200, recovering the simulated three-segment mean structure (segment means 0.04, 2.95, -0.97); a sensitivity analysis with MBIC and Hannan-Quinn penalties yielded the same changepoints.” Always report the algorithm, the penalty, and a sensitivity check across penalties.

9.7.8 Practical Tips

- Choose the right change type: `cpt.mean` for mean shifts, `cpt.var` for variance shifts, `cpt.meanvar` for both, regression-based methods for coefficient breaks.
- The penalty controls false positives; try several (BIC, MBIC, Hannan-Quinn) and check that conclusions are robust.
- For non-stationary noise within segments, model the noise (ARIMA) first and apply changepoint detection to the residuals.
- Bayesian alternatives (`bcp::bcp`) give per-time changepoint posterior probabilities, which are more informative than a hard list of detected changepoints.
- PELT is offline — it requires the full series. For online (real-time) detection, use CUSUM, Page-Hinkley, or Bayesian online methods.
- For multivariate changepoint detection, `ecp` and `InspectChangepoint` extend PELT-style methods to vector time series.

9.7.9 R Packages Used

`changepoint` for `cpt.mean()`, `cpt.var()`, `cpt.meanvar()`, and PELT/Binary Segmentation algorithms; `bcp` for Bayesian changepoint detection with posterior probabilities; `ecp` for non-parametric multivariate changepoints; `cpm` for online changepoint methods.

9.7.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.7.11 See also — labs in this chapter

- Time series basics

9.8 Diebold-Mariano Test

9.8.1 Introduction

The Diebold-Mariano test compares the forecasting accuracy of two competing models on a held-out test window. Where naive accuracy comparisons (one model has lower MSE than another) ignore the statistical noise in the difference, the DM test treats the loss-difference series as a stochastic process and asks whether its mean is significantly non-zero. The test works with any loss function and accommodates the autocorrelation that multi-step-ahead forecast errors typically exhibit, making it the standard inferential tool for forecast comparisons.

9.8.2 Prerequisites

A working understanding of forecast accuracy metrics, the difference between in-sample and out-of-sample evaluation, and basic familiarity with autocorrelation in residual series.

9.8.3 Theory

For two forecasts of the same series with errors $e_{1,t}$ and $e_{2,t}$ on a test window of length T, define $d_t = L(e_{1,t}) - L(e_{2,t})$ where L is a chosen loss (squared error, absolute error, log-squared error). The null is $\mathbb{E}[d_t] = 0$ — equal expected forecast accuracy. The test statistic is the standardised mean of d_t with a long-run-variance estimator that accounts for the natural autocorrelation in the loss-difference series.

The Harvey-Leybourne-Newbold (HLN) small-sample correction adjusts the asymptotic Normal reference for finite T and forecast horizon h, reducing the over-rejection of the original DM test in small samples.

9.8.4 Assumptions

The loss-difference series is covariance-stationary; forecast errors are observable on overlapping or sequential test sets; the chosen loss is appropriate for the forecasting question.

9.8.5 R Implementation

```r
library(forecast)

train ← window(AirPassengers, end = c(1957, 12))
test  ← window(AirPassengers, start = c(1958, 1))

e1 ← test - forecast(auto.arima(train), h = length(test))$mean
e2 ← test - forecast(ets(train), h = length(test))$mean

dm.test(e1, e2, alternative = "two.sided", h = 1, power = 2)
```

9.8.6 Output & Results

`dm.test()` returns the DM statistic, p-value, and the chosen lag for autocovariance estimation. The HLN correction is applied by default in `forecast::dm.test()`.

9.8.7 Interpretation

A reporting sentence: "On the held-out test window of three years (1958–1960), the Diebold-Mariano test comparing ARIMA and ETS forecasts at one-step-ahead horizon gave $\mathrm{DM} = -1.42$ with the HLN small-sample correction ($p = 0.16$); no significant difference in forecast accuracy under squared-error loss." Always state the test window, forecast horizon, and loss function.

9.8.8 Practical Tips

- Use `power = 2` for squared-error loss (default), `power = 1` for absolute error; choose based on the substantive forecasting cost function.
- For multi-step-ahead horizons, set `h` to the forecast horizon; the long-run variance uses h to determine the truncation lag for autocovariance estimation.
- Report the test window length, forecast horizon, and chosen loss; results depend on all three.
- For model selection across many candidate models, the DM test is pairwise; the model-confidence-set approach (`MCS::mcsTest`) extends to many models with multiple-comparison correction.
- The DM test compares forecast methods, not models per se; methods that produce the same point forecast (different prediction-interval widths) may be DM-equivalent despite different underlying assumptions.
- For nested models, the conventional DM test under-covers; use Clark-West or West tests instead.

9.8.9 R Packages Used

`forecast::dm.test()` for the canonical implementation with HLN correction; `MCS` for the model-confidence-set extension to multi-model comparisons; `multDM` for multivariate DM tests; `multMCS` for MCS extensions to multivariate forecasts.

9.8.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.8.11 See also — labs in this chapter

- Time series basics

9.9 Differencing a Time Series

9.9.1 Introduction

Differencing is the simplest tool for making a non-stationary series stationary: replace each observation by its difference from the previous one. First differences remove linear trends; seasonal differences remove fixed-period seasonality; combined regular and seasonal differencing handles both. The order of differencing d in ARIMA(p, d, q) is exactly the count of first differences applied; the seasonal differencing order D in SARIMA $(p, d, q)(P, D, Q)_s$ counts seasonal differences.

9.9.2 Prerequisites

A working understanding of stationarity, AR models, and the relationship between trend, seasonality, and the spectrum of a series.

9.9.3 Theory

Standard differencing operations:

- **First differences**: $\Delta y_t = y_t - y_{t-1}$. Removes linear (level-shift) non-stationarity.
- **Second differences**: $\Delta^2 y_t = \Delta y_t - \Delta y_{t-1}$. Removes quadratic trend. Rarely needed in practice; second differences typically over-difference and inflate residual variance.
- **Seasonal differences** at period s: $\Delta_s y_t = y_t - y_{t-s}$. Removes constant seasonal patterns at frequency $1/s$.
- **Combined regular and seasonal**: $\Delta \Delta_s y_t = (y_t - y_{t-1}) - (y_{t-s} - y_{t-s-1})$.

Over-differencing introduces unnecessary moving-average structure (the differenced series has a strong negative lag-1 autocorrelation); always use the minimum differencing order needed for stationarity. KPSS- or ADF-based automated selection (`forecast::ndiffs` and `nsdiffs`) chooses the required orders.

9.9.4 Assumptions

The non-stationarity is of the kind that differencing addresses (stochastic trend or fixed-period seasonality); deterministic polynomial trends are better handled by detrending regression.

9.9.5 R Implementation

```
library(forecast)

x ← AirPassengers
plot(x)

# First difference
plot(diff(x))

# Seasonal difference (lag = 12 for monthly)
plot(diff(x, lag = 12))

# Both
plot(diff(diff(x, lag = 12)))

# Automatic: number of differences needed
ndiffs(log(x))         # regular
nsdiffs(log(x))        # seasonal
```

9.9.6 Output & Results

`diff(x, lag = s, differences = d)` produces $\Delta_s^d y$. `ndiffs()` returns the required regular differencing order based on KPSS by default; `nsdiffs()` returns the seasonal differencing order. For the log-AirPassengers series, both regular and seasonal differencing once each produces a stationary residual.

9.9.7 Interpretation

A reporting sentence: “Monthly air-passenger counts on the log scale required one regular difference and one seasonal difference at lag 12 to achieve stationarity (KPSS $p > 0.10$ on the doubly-differenced series); the resulting series motivates an ARIMA $(0,1,1)(0,1,1)_{12}$ model on the log scale, equivalent to a multiplicative seasonal ARIMA on the original scale.” Always confirm stationarity after differencing.

9.9.8 Practical Tips

- Log-transform first for series with variance that grows with the level; the log scale is then additive and the differencing is well-behaved.

- ARIMA(p, d, q) encodes the differencing order d; the corresponding `Arima()` call automatically applies the difference and works with the original series for prediction.
- `forecast::ndiffs()` and `nsdiffs()` automate the choice of regular and seasonal differencing orders; useful as a starting point for `auto.arima`.
- Over-differencing inflates residual variance and introduces a strong negative MA(1) coefficient; if you see θ_1 near -0.95 in a differenced model, consider whether the differencing was excessive.
- Always confirm stationarity after differencing with ADF and KPSS; if either test still indicates non-stationarity, examine the data for structural breaks or non-linear trends.
- For deterministic polynomial trends, detrending regression is more efficient than differencing; ARIMA's stochastic-trend interpretation differs.

9.9.9 R Packages Used

Base R `diff()` for differencing operations; `forecast::ndiffs()`, `nsdiffs()`, and `auto.arima()` for automated differencing decisions; `feasts::unitroot_ndiffs()` and `unitroot_nsdiffs()` for the modern tidyverts equivalents; `tseries::adf.test()` and `kpss.test()` for confirming stationarity after differencing.

9.9.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.9.11 See also — labs in this chapter

- Time series basics

9.10 ETS Models

9.10.1 Introduction

ETS models — Error, Trend, Seasonal — express the exponential-smoothing family in a unified state-space form. Each component (error, trend, seasonal) is independently classified as additive, multiplicative, or absent, giving a small library of well-understood models. The state-space framework allows proper likelihood-based estimation, AIC-based model selection, and principled prediction intervals derived from the underlying stochastic structure rather than ad-hoc heuristics.

9.10.2 Prerequisites

A working understanding of exponential smoothing (simple, Holt's, Holt-Winters), state-space models, and the additive vs multiplicative distinction for time-series components.

9.10.3 Theory

The notation ETS(E, T, S) classifies each component:

- **Error**: A (additive) or M (multiplicative).
- **Trend**: N (none), A (additive), Ad (damped additive), M (multiplicative), Md (damped multiplicative).
- **Seasonal**: N (none), A (additive), or M (multiplicative).

Examples: ETS(A, N, N) is simple exponential smoothing; ETS(A, A, N) is Holt's linear trend; ETS(A, A, A) is additive Holt-Winters; ETS(M, A, M) gives multiplicative errors plus additive trend plus multiplicative seasonality. `forecast::ets()` enumerates valid candidates and selects by AIC.

The state-space form provides exact likelihood, AIC for model selection, and analytic (or simulation-based) prediction intervals. Multiplicative components require strictly positive data; the AIC-based selection avoids choosing an invalid model.

9.10.4 Assumptions

The chosen ETS specification matches the data's components; for prediction intervals, the innovations are approximately Normal (or simulated from the fitted residual distribution).

9.10.5 R Implementation

```r
library(forecast)

x ← AirPassengers

fit ← ets(x)
summary(fit)
autoplot(fit)

fc ← forecast(fit, h = 24)
autoplot(fc)

# Residual diagnostics
checkresiduals(fit)
```

9.10.6 Output & Results

`summary(fit)` returns the chosen model code (e.g., "M, A, M"), smoothing parameters α, β, γ, AIC, and BIC. `forecast()` produces point forecasts and 80 % / 95 % prediction intervals from the fitted state-space form. `checkresiduals()` provides Ljung-Box and graphical diagnostics.

9.10.7 Interpretation

A reporting sentence: "`ets()` selected ETS(M, A, M) for the airline series ($\alpha = 0.34$, $\beta = 0.01$, $\gamma = 1.0$, AIC = 1{,}393); the model captures the upward trend and the multiplicative seasonality whose amplitude grows with the level. Standardised residuals passed Ljung-Box at lag 24 ($p = 0.61$)." Always state the chosen ETS specification and the smoothing parameters.

9.10.8 Practical Tips

- `ets()` automatically selects the optimal ETS variant by AIC; override with `model = "AAA"` etc. when a specific form is required for replication.
- Multiplicative components require strictly positive data; AIC-based selection avoids invalid choices automatically.
- For complex multi-seasonality (e.g. weekly + yearly), TBATS (`forecast::tbats()`) extends ETS with multiple seasonal layers and Box-Cox transformation.
- ETS prediction intervals can be very wide at long horizons because the multiplicative variance compounds; for a more nuanced view, simulate many sample paths via `forecast(fit, simulate = TRUE, npaths = 1000)`.
- Compare ETS against ARIMA via rolling-origin cross-validation; both are strong baseline forecasters and the better choice depends on the series.
- For damped trend (which extrapolates more conservatively at long horizons), use `model = "AAdA"` or let `ets()` discover it through AIC.

9.10.9 R Packages Used

`forecast::ets()` for the canonical ETS implementation with AIC-based selection; `fable::ETS()` for the modern tidyverts equivalent; `forecast::tbats()` for multiple seasonalities and Box-Cox transformation; `smooth::es()` for advanced ETS variants including ETS plus regressors.

9.10.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.10.11 See also — labs in this chapter

- Time series basics

9.11 Exponential Smoothing

9.11.1 Introduction

Exponential smoothing is a family of forecasting methods that produces predictions as weighted averages of past observations with geometrically declining weights. The simplest form (simple exponential smoothing, SES) handles series with no trend and no seasonality; Holt's method adds a linear trend; Holt-Winters extends to seasonal series with additive or multiplicative seasonality. The methods are competitive with ARIMA at short horizons, conceptually simpler, and the natural choice when the series structure matches one of the supported variants.

9.11.2 Prerequisites

A working understanding of moving averages, forecasting basics, and the trade-off between adaptiveness and smoothness.

9.11.3 Theory

Simple exponential smoothing recursively updates a level estimate:

$$\hat{y}_{t+1|t} = \alpha y_t + (1-\alpha)\hat{y}_{t|t-1},$$

with smoothing parameter $\alpha \in (0, 1)$. Larger α adapts more quickly to recent values; smaller α smooths more aggressively.

Holt's method adds a trend component with its own smoothing parameter β. Holt-Winters adds a seasonal component with parameter γ in either additive (constant seasonal amplitude) or multiplicative (seasonal amplitude grows with level) form. The equivalent state-space representations form the ETS family — Error/Trend/Seasonal — coded as ETS(A, A, A), ETS(M, Ad, M), and so on, with A = additive, M = multiplicative, N = none, Ad = damped additive trend.

9.11.4 Assumptions

The chosen ETS specification matches the data's components; smoothing parameters are constant over time; for prediction intervals, innovations are approximately Normal.

9.11.5 R Implementation

```r
library(forecast)

x ← AirPassengers

# Holt-Winters multiplicative (for multiplicative seasonality)
hw ← HoltWinters(x, seasonal = "multiplicative")
plot(hw)
fc ← forecast(hw, h = 24)
plot(fc)

# Equivalent via ets()
fit_ets ← ets(x)
summary(fit_ets)
plot(forecast(fit_ets, h = 24))
```

9.11.6 Output & Results

`HoltWinters()` returns smoothing parameters and the fitted level, trend, and seasonal components. `ets()` automatically selects the ETS variant by AIC and produces equivalent output with prediction intervals from the state-space form.

9.11.7 Interpretation

A reporting sentence: "Holt-Winters multiplicative smoothing captured the growing seasonal amplitude in airline passengers (smoothing parameters $\alpha = 0.34$, $\beta = 0.01$, $\gamma = 1.0$); 24-month forecasts extend the trend with realistic seasonal variation, and 95 % prediction intervals widen from $\pm 4\%$ at 1 month to $\pm 18\%$ at 24 months." Always state the ETS variant chosen and the smoothing parameter values.

9.11.8 Practical Tips

- `ets()` automatically selects the ETS family by AIC; specify `model` argument explicitly to force a specific structure.
- For multiplicative seasonality, either fit ETS(...,M) directly or log-transform the series and fit ETS(...,A); both approaches give equivalent forecasts.
- Exponential smoothing is competitive with ARIMA for short horizons (1–4 periods); ARIMA tends to win at longer horizons.
- Prediction intervals from `forecast()` assume Normal innovations; check residuals for skewness and heavy tails.
- For short series ($T < 30$), simple SES or Holt often beats Holt-Winters because parameter estimation is unstable for the more complex models.
- The damped-trend ETS(...,Ad) variant is a sensible default when the trend is plausibly slowing in the forecast horizon; pure additive trends extrapolate forever.

9.11.9 R Packages Used

Base R `HoltWinters()` for the classical implementation; `forecast::ets()` for the modern state-space ETS family with AIC-based selection; `fable::ETS()` for the tidyverts equivalent; `smooth::es()` for advanced exponential-smoothing variants including ETS

plus regressors.

9.11.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.11.11 See also — labs in this chapter

- Time series basics

9.12 Forecast Accuracy Metrics

9.12.1 Introduction

Forecast accuracy is evaluated through error metrics that compare predictions against held-out actuals. Each metric encodes a different loss function and has different units, so the choice of metric depends on the substantive cost structure of forecasting errors. The four most widely reported are MAE, RMSE, MAPE, and MASE; understanding their differences and trade-offs is essential for fair comparison across forecasting methods and across series.

9.12.2 Prerequisites

A working understanding of forecasting, the difference between in-sample and out-of-sample evaluation, and basic loss-function concepts.

9.12.3 Theory

Let $e_t = y_t - \hat{y}_t$ be the forecast error on the test window of length n.

- **Mean Absolute Error**: $\mathrm{MAE} = \frac{1}{n}\sum |e_t|$. Scale-dependent, same units as the data; treats all errors equally.
- **Root Mean Squared Error**: $\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum e_t^2}$. Scale-dependent, penalises large errors more heavily; the natural metric when squared loss is the cost function.
- **Mean Absolute Percentage Error**: $\mathrm{MAPE} = \frac{100}{n}\sum |e_t / y_t|$. Scale-independent percentage; undefined for $y_t = 0$ and asymmetric (penalises over-forecasts more than under-forecasts at the same absolute level).
- **Mean Absolute Scaled Error**: $\mathrm{MASE} = \mathrm{MAE}/\bar{e}_{\mathrm{naive}}$, scaled by the in-sample MAE of a naive forecast. Scale-independent, symmetric, defined for all data (including zeros). Values below 1 mean "better than naive seasonal forecast in sample".

For probabilistic forecasts, the continuous ranked probability score (CRPS) generalises MAE to forecast distributions and is the standard scoring rule for prediction intervals.

9.12.4 Assumptions

A comparable held-out test window of sufficient length to give stable error estimates; the chosen metric matches the substantive loss function of forecasting errors.

9.12.5 R Implementation

```r
library(forecast)

x ← AirPassengers
train ← window(x, end = c(1957, 12))
test  ← window(x, start = c(1958, 1))

fit  ← auto.arima(train)
fc   ← forecast(fit, h = length(test))
accuracy(fc, test)
```

9.12.6 Output & Results

`accuracy()` returns a table with ME (mean error, indicates bias), RMSE, MAE, MPE, MAPE, MASE, and ACF1 (lag-1 autocorrelation of residuals) on both the training and test sets. The training-set numbers measure in-sample fit; the test-set numbers are the genuine out-of-sample predictive performance.

9.12.7 Interpretation

A reporting sentence: “On the held-out three-year window, the ARIMA forecast achieved test-set RMSE of 22.8 passengers, MAE of 18.5, and MASE of 0.42; the MASE below 1 confirms substantially better predictive performance than a naive seasonal forecast.” Always report multiple metrics — RMSE for squared-error context, MAE for absolute-error context, MASE for scale-free comparison.

9.12.8 Practical Tips

- Report multiple metrics; they reward different error properties and a model can dominate on one while losing on another.
- MAPE is popular in business reporting but misleading for series with values near zero or with negative values; use sMAPE or MASE instead.
- MASE is the recommended scale-free metric in academic forecasting evaluation; it is symmetric, defined for zeros, and benchmarked against a meaningful baseline.
- Evaluate on a held-out tail (chronological train/test split), never in-sample; in-sample errors are systematically optimistic.
- For comparing many forecasts across many series, aggregate metrics via rolling-origin cross-validation; a single train/test split is sensitive to the choice of cutoff.
- For probabilistic forecasts, report CRPS or interval scores; point-forecast metrics ignore the prediction-interval calibration.

9.12.9 R Packages Used

`forecast::accuracy()` for the standard suite of point-forecast metrics; `Metrics::rmse()`, `mae()`, `mape()` for individual metrics; `scoringRules` for proper scoring rules including CRPS for probabilistic forecasts; `fabletools::accuracy()` for the modern tidyverts equivalent.

9.12.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.12.11 See also — labs in this chapter

- Time series basics

9.13 GARCH Models

9.13.1 Introduction

GARCH (Generalised Autoregressive Conditional Heteroscedasticity) models capture time-varying conditional variance — the volatility clustering pattern in which quiet periods alternate with volatile ones. Financial returns are the prototypical application: stock prices show stretches of low daily variability punctuated by high-volatility crisis episodes, a structure that conditional-Normal ARMA models cannot accommodate. GARCH extends ARMA by adding an autoregressive equation for the conditional variance itself, making volatility a forecastable quantity.

9.13.2 Prerequisites

A working understanding of ARMA models, conditional distributions, and the distinction between unconditional and conditional variance.

9.13.3 Theory

The GARCH(1, 1) model decomposes returns as

$$r_t = \mu + \varepsilon_t, \qquad \varepsilon_t = \sigma_t z_t, \quad z_t \sim \mathcal{N}(0,1),$$

with the conditional variance evolving as

$$\sigma_t^2 = \omega + \alpha \varepsilon_{t-1}^2 + \beta \sigma_{t-1}^2.$$

The mean structure can be combined with an ARMA component; GARCH specifically models the variance of the innovations. Stationary unconditional variance requires $\alpha + \beta < 1$; values close to 1 indicate highly persistent volatility (typical for daily financial returns).

Extensions handle asymmetric responses to positive vs negative shocks (EGARCH, TGARCH, APARCH), variance feedback into the mean (GARCH-M), and multivariate volatility (DCC, BEKK). For non-Normal returns, t- or skewed-t-distributed innovations are standard.

9.13.4 Assumptions

Conditional Normality (or t, skewed-t) of innovations z_t; stationarity of the conditional variance ($\alpha + \beta < 1$); the chosen GARCH order is appropriate for the data.

9.13.5 R Implementation

```r
library(rugarch)

# Simulate GARCH(1,1) returns
spec ← ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
                  mean.model = list(armaOrder = c(0, 0), include.mean = TRUE),
                  distribution.model = "norm")

set.seed(2026)
sim ← ugarchpath(spec = ugarchspec(
  variance.model = list(garchOrder = c(1, 1)),
  mean.model = list(armaOrder = c(0, 0))),
  n.sim = 500)
r ← as.numeric(fitted(sim))

# Fit
fit ← ugarchfit(spec, data = r)
show(fit)
plot(fit, which = 3)   # conditional sd
```

9.13.6 Output & Results

`ugarchfit()` returns parameter estimates (ω, α, β, mean parameters), log-likelihood, AIC/BIC, and Ljung-Box-style tests on standardised and squared standardised residuals. The conditional-standard-deviation plot shows the time-varying volatility extracted by the model.

9.13.7 Interpretation

A reporting sentence: "A GARCH(1, 1) fit to daily returns estimated $\hat{\omega} = 0.0001$, $\hat{\alpha} = 0.08$, $\hat{\beta} = 0.91$ ($\hat{\alpha} + \hat{\beta} = 0.99$, very persistent volatility); standardised residuals showed no remaining ARCH effects (Ljung-Box on squared residuals $p = 0.42$). Conditional-volatility forecasts widen modestly over the 30-day horizon." Always report the persistence sum and standardised-residual diagnostics.

9.13.8 Practical Tips

- Check standardised residuals (`residuals(fit, standardize = TRUE)`); they should be approximately iid and often heavier-tailed than Normal, motivating t or skewed-t innovations.
- For asymmetric responses to positive vs negative shocks (the leverage effect in equity returns), use EGARCH or TGARCH; standard GARCH is symmetric and misses this.
- `rugarch` is the industry standard for univariate GARCH; for multivariate volatility (covariance matrices), `rmgarch` extends to DCC, BEKK, and copula-GARCH.
- Value-at-Risk forecasts come from the conditional-variance forecast plus a quantile of the innovation distribution; `ugarchforecast()` and `ugarchroll()` support rolling VaR estimation.
- GARCH on returns is conventional, but the framework applies to any series with volatility clustering — physiological signals, atmospheric data, traffic counts.
- For very long series, the parameter estimates can be stable but the conditional-volatility series may need recursive re-estimation; use `ugarchroll()` for genuine out-of-sample evaluation.

9.13.9 R Packages Used

`rugarch` for univariate GARCH and its many extensions; `rmgarch` for multivariate GARCH (DCC, BEKK); `fGarch` as an alternative implementation; `bayesGARCH` for Bayesian GARCH; `tsgarch` for the modern tidyverse-flavoured interface.

9.13.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.13.11 See also — labs in this chapter

- Time series basics

9.14 Granger Causality

9.14.1 Introduction

Granger causality tests whether past values of one time series improve the prediction of another, beyond what is already explained by the latter's own history. It is a *statistical* concept, not a *causal* one in the philosophical sense: a series Granger-causes another if

including its lags reduces the residual variance of the predicted series, which is a useful but limited notion. Two series can Granger-cause each other without any direct causal link if both are driven by a common unobserved factor.

9.14.2 Prerequisites

A working understanding of vector autoregressive (VAR) models, F-tests on joint coefficient hypotheses, and the difference between predictive and causal claims.

9.14.3 Theory

Series X Granger-causes series Y at lag p if the lagged values $X_{t-1}, \ldots, X_{t-p}$ significantly improve the regression of Y_t on its own lags $Y_{t-1}, \ldots, Y_{t-p}$. The test is an F-test on the joint hypothesis that all coefficients on lagged X are zero. Within a VAR system with K series, Granger causality from one series to another (or to all others) is computed similarly with the residuals jointly estimated.

A non-trivial extension is "instantaneous" causality: X and Y contemporaneously correlated after conditioning on lagged values of both. This is included by some implementations (e.g., `vars::causality`).

9.14.4 Assumptions

Stationary series — the test is invalid for unit-root processes. No unmeasured common cause that drives both series; otherwise the Granger-causal direction is uninterpretable.

9.14.5 R Implementation

```r
library(lmtest); library(vars)

data(ChickEgg)   # eggs / chickens

# Does egg production Granger-cause chicken population?
grangertest(chicken ~ egg, order = 3, data = ChickEgg)
grangertest(egg ~ chicken, order = 3, data = ChickEgg)

# Within a VAR
fit ← VAR(ChickEgg, p = 3)
causality(fit, cause = "egg")$Granger
causality(fit, cause = "chicken")$Granger
```

9.14.6 Output & Results

`grangertest()` returns the F-statistic, degrees of freedom, and p-value for the joint hypothesis of zero coefficients on the cause's lags. `vars::causality()` extends this within a VAR framework, handling the multivariate residual structure correctly and adding the instantaneous-causality test.

9.14.7 Interpretation

A reporting sentence: “Egg production Granger-caused chicken population at lag 3 ($F(3, 48) = 4.9, p = 0.005$); the reverse test was not significant ($p = 0.39$). This suggests egg counts contain predictive information about future chicken populations beyond past chicken counts, but does not establish a direct causal mechanism.” Always frame Granger results as predictive, not causal.

9.14.8 Practical Tips

- Granger causality requires stationarity; difference or detrend first using KPSS / ADF guidance.
- Results are sensitive to lag order; try several plausible lags and report the chosen value with justification.
- Granger causality does not imply causation; a common confounder produces apparent bidirectional Granger causality between two effects.
- For more than two series, use the VAR-based `vars::causality()` rather than pairwise `grangertest()`; the joint estimation handles cross-series correlations.
- Non-linear Granger causality exists (`bootGran`, `NlinTS`); the standard tests are linear and miss non-linear predictive relationships.
- For frequency-domain Granger causality, the `vars::fevd()` and Geweke decomposition tools partition predictive contribution by frequency band.

9.14.9 R Packages Used

`lmtest::grangertest()` for the bivariate test; `vars::VAR()` and `causality()` for VAR-based multivariate Granger causality; `tsDyn` for non-linear extensions; `NlinTS` for non-linear and information-theoretic causality measures.

9.14.10 For Reviewers

Use the shared reviewer checklist, together with this method’s assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript’s application of the method.

9.14.11 See also — labs in this chapter

- Time series basics

9.15 Holt-Winters Method

9.15.1 Introduction

The Holt-Winters method extends exponential smoothing to handle both trend and seasonality via three coupled smoothing equations — level, trend, and seasonal — each with its own smoothing parameter. It is the seasonal generalisation of Holt's linear method and the conceptual ancestor of the modern ETS family. Its strength is interpretability and competitive forecasting performance with relatively few parameters; its weakness is the assumption of a fixed seasonal period and stable parameters across the entire series.

9.15.2 Prerequisites

A working understanding of simple exponential smoothing, Holt's linear-trend method, and the additive-vs-multiplicative seasonal distinction.

9.15.3 Theory

The additive Holt-Winters equations are

$$
\begin{aligned}
L_t &= \alpha(y_t - S_{t-s}) + (1-\alpha)(L_{t-1} + T_{t-1}),\\
T_t &= \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1},\\
S_t &= \gamma(y_t - L_t) + (1-\gamma)S_{t-s}.
\end{aligned}
$$

The forecast is $\hat{y}_{t+h} = L_t + hT_t + S_{t-s+1+(h-1) \bmod s}$. The multiplicative variant replaces the seasonal additions with multiplications and is appropriate when seasonal amplitude grows with the level.

The three smoothing parameters α, β, γ each lie in $[0, 1]$: small values give slow adaptation (smoother estimates), large values give fast adaptation (more responsive but noisier).

9.15.4 Assumptions

The seasonal period s is fixed and known; the smoothing parameters are stable over the series; for prediction intervals, residuals are approximately Normal.

9.15.5 R Implementation

```r
library(forecast)

x <- AirPassengers

# Multiplicative
hw_m <- HoltWinters(x, seasonal = "multiplicative")
hw_m$alpha; hw_m$beta; hw_m$gamma

fc <- forecast(hw_m, h = 24)
plot(fc)
```

9.15.6 Output & Results

`HoltWinters()` returns the three smoothing parameters, the fitted level / trend / seasonal components, and the fitted values on the original series. `forecast()` extends to multi-step-ahead predictions with 80 % / 95 % intervals.

9.15.7 Interpretation

A reporting sentence: "Holt-Winters multiplicative smoothing of monthly air-passenger counts estimated $\hat{\alpha} = 0.20$, $\hat{\beta} = 0.03$, $\hat{\gamma} = 0.80$; the resulting 24-month forecast continues the upward trend with proportionally growing seasonal amplitude. The slow β indicates a stable trend slope, while the fast γ allows the seasonal shape to adapt over time." Always state the smoothing parameters and the additive-vs-multiplicative choice.

9.15.8 Practical Tips

- Use additive Holt-Winters for stable seasonal amplitude; multiplicative when seasonal amplitude scales with the level.
- Log-transforming the series often allows additive Holt-Winters to substitute for multiplicative; both produce essentially equivalent forecasts.
- Parameters α, β, γ near 0 give slow updates (highly smoothed); near 1 give fast adaptation (responsive but noisy).
- For complex seasonality (multiple periods, irregular spacing), the older Holt-Winters fails; use BATS or TBATS (`forecast::bats()` , `tbats()`) which handle these cases.
- `forecast::ets()` automates the choice between additive, multiplicative, and damped variants by AIC; reach for it before `HoltWinters()` for new analyses.
- Diagnostic check: the residuals should look like white noise; `checkresiduals(fc)` runs Ljung-Box and visual diagnostics.

9.15.9 R Packages Used

Base R `HoltWinters()` for the classical implementation; `forecast::ets()` for the modern state-space ETS family that includes Holt-Winters as ETS(...,A,A) or ETS(...,M,M); `forecast::tbats()` for multi-seasonal extensions; `fable::ETS()` for the tidyverts equivalent.

9.15.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.15.11 See also — labs in this chapter

- Time series basics

9.16 The Kalman Filter

9.16.1 Introduction

The Kalman filter is the workhorse algorithm for state-space models with linear-Gaussian observation and transition equations. It recursively combines the previous state estimate with each new observation to produce an updated state estimate together with its uncertainty, in a closed-form Gaussian update that is provably optimal under the linear-Gaussian assumptions. Originally developed for aerospace tracking, it now underpins virtually every classical time-series filtering application — economic indicators, signal processing, target tracking, financial volatility estimation.

9.16.2 Prerequisites

A working understanding of state-space models, linear algebra, and the multivariate Normal distribution.

9.16.3 Theory

For a linear-Gaussian state-space model with observation $y_t = Fx_t + v_t$ and transition $x_t = Gx_{t-1} + w_t$, the Kalman filter alternates two steps at each time:

Predict:

$$x_{t|t-1} = Gx_{t-1|t-1}, \qquad P_{t|t-1} = GP_{t-1|t-1}G^\top + W.$$

Update:

$$x_{t|t} = x_{t|t-1} + K_t(y_t - Fx_{t|t-1}), \qquad P_{t|t} = (I - K_tF)P_{t|t-1},$$

with Kalman gain $K_t = P_{t|t-1}F^\top(FP_{t|t-1}F^\top + V)^{-1}$.

The smoother (Rauch-Tung-Striebel) is a backward recursion that combines forward filter estimates with future observations to produce smoothed estimates $x_{t|T}$. Non-linear extensions include the extended Kalman filter (linearisation around the current estimate) and the unscented Kalman filter (sigma-point sampling); particle filters handle fully non-Gaussian and non-linear models via Monte Carlo.

9.16.4 Assumptions

Linear Gaussian observation and transition equations with known parameters (F, G, V, W). Parameters can be estimated by MLE on the innovations.

9.16.5 R Implementation

```r
library(dlm)

# Local-level model on Nile
mod ← dlmModPoly(order = 1, dV = 15100, dW = 1470)
filt ← dlmFilter(Nile, mod)

plot(Nile)
lines(dropFirst(filt$m), col = "red", lwd = 2)

# Smoother
smo ← dlmSmooth(filt)
lines(dropFirst(smo$s), col = "blue", lwd = 2)
```

9.16.6 Output & Results

`dlmFilter()` returns the filtered state estimates ($x_{t|t}$) and their covariances. `dlmSmooth()` produces the retrospective smoothed estimates ($x_{t|T}$). Plotting both on the original series shows the filter's adaptation in real time and the smoother's retrospective best estimate.

9.16.7 Interpretation

A reporting sentence: "The Kalman filter applied to the Nile river-flow series produced one-step-ahead forecasts (filtered states) that adapt to each new observation; the smoothed estimate, using the full series in retrospect, identified a sharp level drop near 1899 that the forward filter takes several years to track." Distinguish filtered (real-time) from smoothed (retrospective) estimates in any report.

9.16.8 Practical Tips

- For non-linear models, use the extended Kalman filter (`KFAS::EKF` or hand-coded) for mild non-linearity, the unscented Kalman filter for stronger non-linearity, or particle filters (`SMCSamplers`, `SMC2`) for arbitrary non-linear / non-Gaussian models.
- Uncertainty propagates naturally through both predict and update steps; report state estimates with their estimated standard errors.
- The log-likelihood of the model can be computed from the innovations (`dlmLL` in `dlm`) and used for MLE-based parameter estimation; the Kalman filter is the engine, MLE is the parameter-fitting wrapper.
- The choice of initial state $x_{0|0}$ and initial variance $P_{0|0}$ matters for the first few time steps; diffuse initialisation ($P_{0|0}$ very large) is the standard choice when no prior information is available.
- For extremely high-dimensional state spaces (large meteorological models), the ensemble Kalman filter approximates the covariance with a low-rank ensemble; standard Kalman fails in that setting.
- Pair filtering with diagnostic checks on the innovations (one-step-ahead residuals); they should be approximately white noise under correct model specification.

9.16.9 R Packages Used

`dlm` for `dlmFilter()`, `dlmSmooth()`, and `dlmLL()`; `KFAS` for fast filtering with non-Gaussian observations; `FKF` for the fast Kalman filter implementation; `bsts` for Bayesian structural time-series alternatives; `nimble` for hand-built non-linear filters via probabilistic programming.

9.16.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.16.11 See also — labs in this chapter

- Time series basics

9.17 KPSS Test

9.17.1 Introduction

The Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test is the natural complement to the augmented Dickey-Fuller (ADF) test for assessing stationarity. The two tests have flipped null hypotheses: ADF tests a unit-root null against a stationary alternative; KPSS tests a stationary null against a unit-root alternative. Running both and comparing their conclusions gives a more robust verdict than either alone, because a single test that fails to reject is uninformative — failure to reject a unit-root null is not evidence of stationarity, and vice versa.

9.17.2 Prerequisites

A working understanding of stationarity, unit-root processes (random walks), and the augmented Dickey-Fuller test as the complementary stationarity diagnostic.

9.17.3 Theory

The KPSS framework decomposes $y_t = r_t + \beta t + \varepsilon_t$, where $r_t = r_{t-1} + u_t$ is a random walk. Under the null of stationarity, the variance of the random-walk innovations is zero ($\sigma_u^2 = 0$); under the alternative, r_t contributes a stochastic trend. The test statistic is based on partial sums of residuals from regressing y_t on (optionally) a constant and a trend; large partial-sum behaviour indicates a unit root and the test rejects stationarity.

The "Level" version tests stationarity around a constant mean; the "Trend" version tests stationarity around a deterministic trend. Choosing between them depends on whether the series has a visible time trend.

9.17.4 Assumptions

Correct model specification (level vs trend stationarity); the lag truncation parameter is adequate for the underlying autocorrelation structure (default works for most applications).

9.17.5 R Implementation

```r
library(tseries)

# Stationary series
y1 ← rnorm(100)
kpss.test(y1, null = "Level")

# Random walk (unit root)
y2 ← cumsum(rnorm(100))
kpss.test(y2, null = "Level")

# Trend-stationary series
y3 ← 0.5 * (1:100) + rnorm(100)
kpss.test(y3, null = "Trend")
```

9.17.6 Output & Results

`kpss.test()` returns the KPSS statistic and the p-value (with a note about the truncation lag and the type of stationarity tested). Small statistics correspond to large p-values and indicate insufficient evidence to reject stationarity; large statistics reject and support a unit root.

9.17.7 Interpretation

A reporting sentence: “The stationary white-noise series gave a KPSS statistic of 0.08 ($p > 0.10$, fail to reject stationarity); the random-walk series gave a statistic of 2.34 ($p < 0.01$, reject stationarity).” Always state which null was tested (Level vs Trend) and pair with an ADF test for the complementary perspective.

9.17.8 Practical Tips

- `null = "Level"` for plain stationarity around a constant mean; `"Trend"` for stationarity around a deterministic trend. Choose based on whether a time-trend is visible in the series.
- Bandwidth (lag-truncation) choice affects test size; the default is adequate for most series. For long-memory or strongly autocorrelated series, increase the truncation manually.
- Combine with the ADF test: if both agree (ADF rejects unit root, KPSS fails to reject stationarity), stationarity is well-supported; if both disagree (ADF fails, KPSS rejects), a unit root is well-supported.
- If both tests give ambiguous answers (ADF fails, KPSS fails), the series is borderline and additional diagnostics (autocorrelation structure, Phillips-Perron, fractional integration) help.

- For multivariate or panel data, use Im-Pesaran-Shin or Levin-Lin-Chu tests; KPSS is univariate.
- Differencing the series and re-testing is the standard workflow if a unit root is detected; the differenced series should pass both KPSS and ADF.

9.17.9 R Packages Used

`tseries::kpss.test()` for the canonical implementation; `urca::ur.kpss()` for an alternative with finer control over deterministic components and bandwidth selection; `tseries::adf.test()` for the complementary ADF test; `forecast::ndiffs()` for an integrated workflow that automatically applies KPSS to determine the required differencing order.

9.17.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.17.11 See also — labs in this chapter

- Time series basics

9.18 The Ljung-Box Test

9.18.1 Introduction

The Ljung-Box test is the standard portmanteau test for residual autocorrelation in time-series models. After fitting an ARIMA, ETS, or other time-series model, the question is whether the residuals are white noise; the Ljung-Box test pools sample autocorrelations over the first several lags into a single chi-squared statistic. A non-significant result supports model adequacy; a significant result indicates missed dynamics that should motivate refining the model.

9.18.2 Prerequisites

A working understanding of autocorrelation functions, ARIMA models, and the white-noise assumption on time-series residuals.

9.18.3 Theory

$$Q = n(n+2)\sum_{k=1}^{h}\frac{\hat{\rho}_k^2}{n-k} \sim \chi^2_{h-p-q} \text{ under white noise,}$$

where $\hat{\rho}_k$ is the sample ACF at lag k, h is the maximum lag, and $p + q$ is the number of ARMA parameters fit. The `fitdf` argument in `Box.test()` adjusts the degrees of freedom for parameters estimated from the data; for residuals from raw data (no model fitted), set `fitdf = 0`.

The Ljung-Box improves on the older Box-Pierce statistic by introducing the small-sample correction $n(n+2)$ that prevents under-rejection at small n. For seasonal models, the maximum lag should span at least one seasonal period.

9.18.4 Assumptions

Residuals are iid under the null; the maximum lag h is small relative to the series length n.

9.18.5 R Implementation

```
set.seed(2026)
x ← arima.sim(list(ar = 0.5), n = 200)
fit ← arima(x, order = c(1, 0, 0))
e ← residuals(fit)

# Ljung-Box at lag 20, 1 parameter fit
Box.test(e, lag = 20, type = "Ljung-Box", fitdf = 1)

# On an unfit series (no fitdf adjustment)
Box.test(x, lag = 20, type = "Ljung-Box")
```

9.18.6 Output & Results

`Box.test()` returns the chi-squared statistic and the p-value at the chosen lag. Pair with the ACF and PACF plots of residuals for a visual confirmation; isolated significant autocorrelations may not move the omnibus test but are still substantively meaningful.

9.18.7 Interpretation

A reporting sentence: “Residuals from the AR(1) fit gave a Ljung-Box statistic at lag 20 of $Q = 18.9$ on 19 degrees of freedom ($p = 0.45$), consistent with white noise. The ACF and PACF of residuals showed no isolated significant lags, supporting model adequacy.” Always pair the test result with the lag chosen and the parameters subtracted.

9.18.8 Practical Tips

- Choose the maximum lag h around $\min(10, n/5)$ or based on domain knowledge; for monthly data with annual seasonality, $h \geq 24$ is sensible.
- Use `fitdf = p + q` for ARMA(p, q) fitted models; the test ignores the parameter cost otherwise and is anti-conservative.
- For seasonal models, set the maximum lag to at least one full seasonal period to detect missed seasonal autocorrelation.
- Rejection at any lag indicates missed dynamics; re-examine the model order, add seasonal terms, or consider a different family (state-space, GARCH).

- `forecast::checkresiduals()` shows the Ljung-Box result alongside ACF and time-series plots, integrating numerical and visual diagnostics.
- The Ljung-Box is sensitive to autocorrelation but not to non-stationarity or distributional assumptions; complement with KPSS / ADF and a residual normality plot.

9.18.9 R Packages Used

Base R `Box.test()` for the canonical implementation; `forecast::checkresiduals()` for an integrated diagnostic with time-series and ACF plots; `feasts::ljung_box()` for the modern tidyverts interface; `astsa::sarima()` provides automatic Ljung-Box reporting in its diagnostic output.

9.18.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.18.11 See also — labs in this chapter

- Time series basics

9.19 Moving Averages

9.19.1 Introduction

A moving average smooths a time series by replacing each observation with an average of nearby values. It is the oldest and simplest tool for highlighting trend by suppressing high-frequency noise; it underpins more sophisticated techniques (X-11 decomposition, Henderson trend filters) and is fundamental to financial chart analysis. The choice between trailing, centred, and weighted moving averages governs the trade-off between smoothness and lag, and choosing the right window size for the application is the principal craft of moving-average smoothing.

9.19.2 Prerequisites

A working understanding of time series, the trade-off between smoothness and lag, and the role of window size in noise suppression.

9.19.3 Theory

Three main variants:

- **Simple moving average**: arithmetic mean of k consecutive observations. Trailing alignment uses the last k values up to time t (one-sided, introduces lag); centred alignment averages $(k-1)/2$ before and after (two-sided, no lag but undefined at the boundaries).
- **Weighted moving average**: assigns weights to each observation in the window — triangular, Hamming, Spencer 15-term (used in classical seasonal adjustment), or any user-defined kernel.
- **Exponential moving average**: geometrically declining weights, $\hat{y}_t = \alpha y_t + (1-\alpha)\hat{y}_{t-1}$. Closely related to simple exponential smoothing.

A centred MA with window equal to the seasonal period eliminates the seasonal component, leaving trend plus residual; this is the foundation of classical seasonal decomposition.

9.19.4 Assumptions

Regular sampling intervals (irregular spacing requires alternative methods); stationarity is not required, but the moving-average filter assumes a slowly-varying trend.

9.19.5 R Implementation

```r
library(zoo); library(TTR)

x ← AirPassengers

# Trailing mean
sm_3 ← rollmean(x, 3, align = "right", fill = NA)

# Centred 12-month MA (removes annual seasonality)
sm_12 ← rollmean(x, 12, align = "center", fill = NA)

# Weighted MA
sm_w ← TTR::WMA(x, n = 12, wts = 1:12)

plot(x); lines(sm_12, col = "red", lwd = 2)
```

9.19.6 Output & Results

`rollmean()` returns the smoothed series with NA at boundaries (trailing alignment leaves NAs at the start; centred alignment leaves NAs at both ends). `TTR::WMA()` produces weighted alternatives. Plotting the centred 12-month MA on top of the raw series reveals the trend that the seasonal cycle hides.

9.19.7 Interpretation

A reporting sentence: “A centred 12-month moving average revealed the secular trend in monthly air-passenger counts, with the smoothed series showing approximately 5 % annual growth and the seasonal cycle largely eliminated by averaging over a full period.” Quote the window size and alignment in any report.

9.19.8 Practical Tips

- A centred moving average of window equal to the seasonal period eliminates the seasonal cycle exactly; this is the X-11 first step.
- Trailing moving averages introduce lag proportional to half the window size; use only when real-time estimation is needed and the lag is acceptable.
- For smoother curves with minimal lag, exponential smoothing, LOESS, or spline smoothers are often preferable to wide moving averages.
- Centred MA is undefined at the start and end of the series; for the boundary, use one-sided MA, asymmetric weights, or an end-effects correction.
- Weighted MAs with carefully chosen kernels (Spencer's 15-term, Henderson's filters) are used in official-statistics seasonal adjustment because they preserve cubic polynomial trends while suppressing irregular components.
- Pair MA smoothing with the original series on the same plot; the gap between them is the seasonal-plus-irregular component.

9.19.9 R Packages Used

`zoo::rollmean()` and `rollmeanr()` for simple moving averages with control over alignment; `TTR::SMA()`, `WMA()`, and `EMA()` for technical-analysis moving averages including weighted and exponential; `forecast::ma()` for centred MAs with seasonal-period defaults; `slider::slide_dbl()` for tidyverse-friendly sliding-window computations.

9.19.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.19.11 See also — labs in this chapter

- Time series basics

9.20 Vector Autoregression (VAR)

9.20.1 Introduction

Vector autoregression (VAR) is the multivariate generalisation of an autoregressive model: each variable in a k-variate time series depends linearly on lags of itself and lags of every other variable in the system. It is the workhorse of empirical macroeconomics, where modelling GDP, inflation, and interest rates jointly produces forecasts and impulse-

response analyses that single-equation models cannot. Outside macro, VAR is the natural tool whenever multiple time series are believed to be jointly determined and forecasting all of them is the goal.

9.20.2 Prerequisites

A working understanding of univariate AR models, multivariate data, and the concept of system-wide forecasting and identification.

9.20.3 Theory

A VAR(p) model is

$$\mathbf{y}_t = \mathbf{c} + A_1 \mathbf{y}_{t-1} + A_2 \mathbf{y}_{t-2} + \cdots + A_p \mathbf{y}_{t-p} + \mathbf{u}_t,$$

where $\mathbf{y}_t$ is a k-vector, A_i are $k \times k$ coefficient matrices, and $\mathbf{u}_t$ are jointly white-noise innovations with covariance Σ_u. The model is estimated equation-by-equation by OLS (the same as multivariate OLS under spherical errors), and lag length p is chosen by AIC, BIC, HQ, or sequential LRT.

The VAR feeds three key downstream analyses: multi-step forecasts with system-coherent uncertainty bands, impulse-response functions that trace the dynamic effect of a shock to one variable through the system, and Granger-causality tests for predictive relationships among the variables.

9.20.4 Assumptions

Stationary series (difference first if not, or use VECM for cointegrated I(1) series); independent, homoscedastic, jointly white-noise innovations; correctly specified lag order.

9.20.5 R Implementation

```r
library(vars)

data(Canada)   # four-variate quarterly macroeconomic series
VARselect(Canada, lag.max = 10, type = "const")$selection

fit ← VAR(Canada, p = 2, type = "const")
summary(fit)

# Forecast
fc ← predict(fit, n.ahead = 8)
plot(fc)
```

9.20.6 Output & Results

`VARselect()` returns suggested lag orders by four information criteria; differing recommendations across criteria are common. `summary()` returns equation-by-equation coefficient tables. `predict()` produces multi-step forecasts with system-coherent confidence bands.

9.20.7 Interpretation

A reporting sentence: "VARselect identified $p = 2$ as the AIC- and HQ-optimal lag order for the four-variate Canada series; the VAR(2) model produced eight-quarter forecasts with prediction intervals that widen from ± 0.5 % at one quarter to ± 1.8 % at eight, with the multivariate covariance structure preserving the empirical co-movements between employment, GDP, and inflation." Always state the lag selection criterion and the deterministic specification (constant, trend, or both).

9.20.8 Practical Tips

- Select lag order by AIC, BIC, or HQ information criteria; differing recommendations are common, and the conservative HQ or BIC choice is usually preferable for forecasting.
- Check residual cross-correlations with `serial.test(fit)`; significant off-diagonals indicate missed cross-series dynamics.
- For cointegrated I(1) series, use VECM (`urca::ca.jo()`) on the levels rather than VAR on differences; differencing throws away the long-run equilibrium relationship.
- Structural VARs impose identifying restrictions (Cholesky, sign, long-run) on Σ_u to give impulse responses a causal interpretation; the standard reduced-form VAR does not.
- Forecast-error variance decompositions (`fevd(fit)`) partition forecast uncertainty across innovations; useful for understanding which variables drive forecast errors at each horizon.
- For high-dimensional VARs ($k > 10$), use Bayesian VARs with Minnesota or stochastic-search variable-selection priors; reduced-form OLS becomes unstable.

9.20.9 R Packages Used

`vars::VAR()`, `predict()`, `irf()`, `fevd()`, `causality()` for the canonical VAR workflow; `urca::ca.jo()` for cointegration tests as the precursor to VECM; `tsDyn::lineVar()` for an alternative implementation; `BVAR` for Bayesian VARs with hierarchical priors; `fable::VAR()` for the modern tidyverts interface.

9.20.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.20.11 See also — labs in this chapter

- Time series basics

9.21 Phillips-Perron Test

9.21.1 Introduction

The Phillips-Perron (PP) test is a unit-root test that modifies the Dickey-Fuller statistic to account for autocorrelation and heteroscedasticity in the errors non-parametrically. Where the augmented Dickey-Fuller test handles serial correlation by including lagged differences in the regression, PP applies a Newey-West long-run variance correction directly to the test statistic. The two approaches usually agree; PP is preferable when the lag specification of ADF is uncertain or when heteroscedasticity is suspected.

9.21.2 Prerequisites

A working understanding of the augmented Dickey-Fuller test, the Newey-West long-run variance estimator, and the unit-root vs stationarity distinction.

9.21.3 Theory

PP fits the unit-root regression $\Delta y_t = \alpha + \gamma y_{t-1} + \varepsilon_t$ (without lagged differences) and computes a test statistic that is non-parametrically corrected for residual autocorrelation and heteroscedasticity using the Newey-West estimator of the long-run variance. Two flavours exist: $Z(\alpha)$ (based on the OLS coefficient) and $Z(t)$ (based on the t-statistic). The asymptotic distribution under the unit-root null is the same Dickey-Fuller distribution as ADF, so critical values are tabulated.

The non-parametric correction sidesteps the ADF lag-order choice but requires a bandwidth choice for the long-run variance estimator; defaults work for most series.

9.21.4 Assumptions

Errors are stationary after the unit-root transformation; the long-run variance bandwidth is appropriate for the autocorrelation horizon.

9.21.5 R Implementation

```r
library(tseries)

x ← cumsum(rnorm(100))

pp.test(x)                      # default
pp.test(x, type = "Z(alpha)")
pp.test(diff(x))

# Compare with ADF
adf.test(x)
```

9.21.6 Output & Results

`pp.test()` returns the test statistic and a p-value computed from the Dickey-Fuller distribution. Comparing PP and ADF on the same series is a useful robustness check; large discrepancies signal autocorrelation or heteroscedasticity issues that one test handles better than the other.

9.21.7 Interpretation

A reporting sentence: "The Phillips-Perron test on the random-walk series gave $p = 0.58$, failing to reject the unit-root null; the augmented Dickey-Fuller agreed ($p = 0.52$). After first-differencing both tests rejected the null at $p < 0.01$, confirming the series is integrated of order 1." Always pair PP with ADF and ideally with KPSS for the most robust unit-root verdict.

9.21.8 Practical Tips

- PP and ADF usually agree; if they disagree, examine the residual autocorrelation structure to identify which test is more appropriate.
- PP's non-parametric correction avoids the ADF lag-order sensitivity but introduces a bandwidth choice; the default Newey-West truncation is usually fine.
- Both PP and ADF have low power against near-unit-root stationary processes (e.g., AR(1) with coefficient 0.95); for borderline series, no test gives a confident answer.
- Combine PP with KPSS for the two-sided perspective: PP / ADF test the unit-root null while KPSS tests the stationarity null.
- For series with structural breaks, PP is invalid; use Zivot-Andrews (`urca::ur.za()`) instead.
- For panel data, panel unit-root tests (Im-Pesaran-Shin, Levin-Lin-Chu) extend the framework to multiple series simultaneously.

9.21.9 R Packages Used

`tseries::pp.test()` for the canonical Phillips-Perron implementation; `urca::ur.pp()` for richer output with multiple critical values; `tseries::adf.test()` and `urca::ur.df()` for the complementary ADF; `urca::ur.za()` for Zivot-Andrews when structural breaks are plausible.

9.21.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.21.11 See also — labs in this chapter

- Time series basics

9.22 Rolling-Origin Cross-Validation

9.22.1 Introduction

Standard k-fold cross-validation shuffles observations across folds, which violates time ordering and leaks future information into the training data. For time series, this invalidates predictive performance estimates entirely. Rolling-origin cross-validation (also called time-series CV or walk-forward validation) preserves temporal ordering by training up to a cutoff and forecasting the subsequent horizon, then sliding the cutoff forward and repeating. It is the only valid CV scheme for time-series forecast evaluation.

9.22.2 Prerequisites

A working understanding of cross-validation, time-series forecasting, and the difference between in-sample fit and out-of-sample predictive performance.

9.22.3 Theory

For a series of length T, choose an initial training-window size t_0, a forecast horizon h, and (optionally) a step size for cutoff advancement. For each cutoff $t = t_0, t_0 + s, \ldots, T - h$:

1. Train on $\{y_1, \ldots, y_t\}$.
2. Forecast $\hat{y}_{t+1}, \ldots, \hat{y}_{t+h}$.
3. Compute the error against observed $y_{t+1}, \ldots, y_{t+h}$.

Average errors across cutoffs for each horizon $1, 2, \ldots, h$ to obtain horizon-specific accuracy estimates. Two windowing variants exist: expanding window (training set grows) and rolling window (fixed-size training set slides forward). Expanding is more efficient when the underlying process is stable; rolling is more robust to regime changes.

9.22.4 Assumptions

Sufficient history that the initial training window estimates the model adequately; the model can be refit at each cutoff (or parameters updated efficiently for computational savings).

9.22.5 R Implementation

```r
library(forecast)

x ← AirPassengers
h ← 12

# tsCV: rolling-origin with one-step expanding window
f_ets ← function(y, h) forecast(ets(y), h = h)$mean
err_ets ← tsCV(x, f_ets, h = h)

# RMSE by horizon
sqrt(colMeans(err_ets^2, na.rm = TRUE))

# Compare ARIMA
f_arima ← function(y, h) forecast(auto.arima(y), h = h)$mean
err_arima ← tsCV(x, f_arima, h = h)
sqrt(colMeans(err_arima^2, na.rm = TRUE))
```

9.22.6 Output & Results

`tsCV()` returns a matrix of forecast errors with one row per cutoff and one column per horizon. Column-wise summaries give horizon-specific RMSE, MAE, or other metrics. Comparing matrices across models identifies where each performs best.

9.22.7 Interpretation

A reporting sentence: “Rolling-origin CV with one-step expanding window and forecast horizon 12 months showed ARIMA outperforming ETS at short horizons (1–3 months: RMSE 22 vs 28) but ETS was more accurate at longer horizons (10–12 months: RMSE 38 vs 45). The horizon-specific differences justify reporting CV-RMSE per horizon rather than a single averaged metric.” Report horizon-specific errors; collapsing them hides the structure that informs model choice.

9.22.8 Practical Tips

- `forecast::tsCV()` implements expanding-window CV with one-step cutoff advancement; for rolling fixed-window, write a small loop.
- For computational cost, increase the initial window and skip cutoffs (advance by more than one observation per step); this trades estimation precision for speed.
- Always report horizon-specific errors; a single averaged RMSE hides the structure that determines which model wins at which horizon.
- Refitting the model at every cutoff is expensive; for ARIMA, fixed-parameter updates (`forecast(Arima(y, model = fit), h = h)`) are faster but ignore changing dynamics.
- `fable::stretch_tsibble()` provides tidy rolling-CV splits that integrate with the modern tidyverts forecasting workflow.
- For nested or hierarchical forecasts, run rolling-origin CV at the bottom level and aggregate; never CV at the aggregated level alone.

9.22.9 R Packages Used

`forecast::tsCV()` for expanding-window time-series CV; `fable::stretch_tsibble()` and `accuracy()` for tidyverts CV workflows; `tibbletime::rollify()` for general rolling functions; `slider::slide_dbl()` for sliding-window operations.

9.22.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.22.11 See also — labs in this chapter

- Time series basics

9.23 Seasonal ARIMA (SARIMA)

9.23.1 Introduction

Seasonal ARIMA (SARIMA) extends the standard ARIMA framework to series with periodic seasonality. The standard notation is $\mathrm{ARIMA}(p,d,q)(P,D,Q)_s$ where the lower-case letters are non-seasonal orders, the upper-case are seasonal orders, and s is the seasonal period (12 for monthly with annual seasonality, 4 for quarterly, 24 for hourly with daily seasonality). SARIMA is the workhorse parametric forecasting tool for economic, financial, and many biomedical time series with stable seasonal patterns.

9.23.2 Prerequisites

A working understanding of ARIMA models, differencing, the backshift operator, and seasonal decomposition.

9.23.3 Theory

The full SARIMA equation is

$$\Phi_P(B^s)\phi_p(B)(1-B)^d(1-B^s)^D y_t = \Theta_Q(B^s)\theta_q(B)\varepsilon_t,$$

where B is the backshift operator ($By_t = y_{t-1}$). The non-seasonal AR, differencing, and MA components compose multiplicatively with their seasonal counterparts at lag s. The model accommodates trends through differencing (d) and seasonal patterns through both seasonal differencing (D) and seasonal AR/MA terms.

The famous "airline model" $(0,1,1)(0,1,1)_{12}$ for monthly airline passengers — Box and Jenkins's headline example — demonstrates that even highly trending and strongly seasonal series can fit into the SARIMA family with modest order.

9.23.4 Assumptions

The series is stationary after regular and seasonal differencing; residuals are white noise; the seasonal period s is correctly specified by the `frequency()` of the input `ts` object.

9.23.5 R Implementation

```
library(forecast)

x ← AirPassengers

fit ← auto.arima(log(x), seasonal = TRUE)
summary(fit)

fc ← forecast(fit, h = 24)
autoplot(fc)

checkresiduals(fit)
```

9.23.6 Output & Results

`auto.arima()` searches over candidate orders using AICc and returns the chosen SARIMA model with its coefficients and standard errors. `forecast()` produces point forecasts and 80 % / 95 % prediction intervals. `checkresiduals()` reports the Ljung-Box test, ACF, and time-series plot of residuals.

9.23.7 Interpretation

A reporting sentence: "`auto.arima` on log-transformed AirPassengers selected an ARIMA $(0,1,1)(0,1,1)_{12}$ structure (AICc = -483, $\sigma^2 = 0.0014$); residuals passed Ljung-Box at lag 24 ($p = 0.61$). Forecasts for the next 24 months show continued multiplicative growth and stable seasonal pattern, with 95 % prediction intervals widening from $\pm 5\%$ at 1 month to $\pm 18\%$ at 24 months." Always check residual diagnostics before reporting forecasts.

9.23.8 Practical Tips

- Use `auto.arima()` for automatic order selection; double-check the result against ACF/PACF inspection of the differenced series.
- Log-transform for multiplicative seasonality (where seasonal amplitude grows with the level); SARIMA on the log scale is then additive and well-behaved.
- For very long seasonal periods ($s \geq 52$), SARIMA becomes computationally heavy; consider regression with Fourier seasonal terms (`forecast::tslm` with `fourier()`) instead.
- Always use chronological train/test splits; random CV is invalid for time series because it leaks future information into past folds.

- Box-Jenkins residual diagnostics (Ljung-Box, ACF, PACF, normality plot) are essential after fitting; `checkresiduals()` aggregates them.
- For multiple seasonalities (weekly + yearly), use TBATS (`forecast::tbats`) or `mstl()` decomposition with non-seasonal forecasting on the adjusted series.

9.23.9 R Packages Used

`forecast::auto.arima()` and `Arima()` for canonical SARIMA fitting and forecasting; `fable::ARIMA()` for the modern tidyverts interface; `feasts::ACF()` and `PACF()` for diagnostic visualisations; `seasonal::seas()` for X-13ARIMA-SEATS as the official-statistics alternative.

9.23.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.23.11 See also — labs in this chapter

- Time series basics

9.24 Seasonal Decomposition (STL)

9.24.1 Introduction

STL (Seasonal and Trend decomposition using Loess) is the most flexible classical decomposition method for time series. It separates a series into trend, seasonal, and remainder components using iterative locally-weighted regression (loess) smoothers. Three properties distinguish STL from alternatives: it handles any kind of seasonality (not just monthly or quarterly), it allows the seasonal pattern to change over time, and it is robust to outliers when run with the robust option enabled.

9.24.2 Prerequisites

A working understanding of time-series decomposition, the additive vs multiplicative model distinction, and basic familiarity with loess smoothers.

9.24.3 Theory

STL alternates two loess smoothing operations until convergence: a seasonal smoother that operates within each cycle position (all Januaries together, all Februaries together, etc.) and a trend smoother that operates across the de-seasonalised series. Two window parameters control the resulting decomposition:

- `s.window`: seasonal-window length. `"periodic"` forces a constant seasonal pattern across the series; an integer (typical 7, 9, 11) allows the seasonal pattern to change gradually.
- `t.window`: trend-window length. Larger values produce smoother trend estimates.

For multiplicative seasonality (where seasonal amplitude grows with the level), apply STL to the log of the series; the result is interpretable as an additive decomposition on the log scale.

9.24.4 Assumptions

Additive components on the chosen scale; the seasonal cycle is correctly specified (`frequency()` of the `ts` object); sufficient series length to estimate the decomposition (at least two full seasonal cycles).

9.24.5 R Implementation

```r
library(forecast)

# log-transform for multiplicative data
x ← log(AirPassengers)

# STL with fixed seasonal
stl_fit ← stl(x, s.window = "periodic")
autoplot(stl_fit)

# Seasonal adjustment
seasadj ← seasadj(stl_fit)
plot(seasadj)

# Forecast via STL + ETS
fc ← forecast(stl_fit, method = "ets", h = 24)
autoplot(fc)
```

9.24.6 Output & Results

`stl()` returns an `stl` object with the decomposition. `autoplot()` produces a four-panel display: original series, trend, seasonal, and remainder. `seasadj()` extracts the seasonally adjusted series (original minus seasonal); `forecast(stl_fit)` applies a chosen method (ETS, ARIMA) to the seasonally adjusted component and adds back the seasonal pattern.

9.24.7 Interpretation

A reporting sentence: “STL decomposition of log-AirPassengers showed a smooth upward trend (annual growth approximately 5 %), stable multiplicative seasonality (constant on the log scale, multiplicative on the original), and a remainder series consistent with white noise (Ljung-Box $p = 0.32$); the seasonal pattern peaked in July–August, consistent with summer holiday travel.” Always describe trend and seasonality separately and check that the remainder is approximately white noise.

9.24.8 Practical Tips

- `s.window = "periodic"` forces a constant seasonal pattern across the series; integer values (7, 11) allow the seasonal pattern to evolve. Use the integer when seasonality has visibly shifted over the series.
- Set `robust = TRUE` for outlier-resistant fitting; the standard STL is sensitive to single extreme observations.
- For forecasting, `stlf()` is a one-call wrapper that applies STL + an automatic ETS or ARIMA forecast on the seasonally adjusted series.
- STL handles any seasonal period; for multiple seasonalities (e.g. weekly and yearly), use `forecast::mstl()` which decomposes into multiple seasonal layers.
- Residuals (remainder) should look like white noise; systematic structure indicates the trend or seasonal windows need tuning.
- Compare STL to classical seasonal decomposition (`decompose`) and X-13 (`seasonal::seas`); STL is generally more flexible but X-13 is the regulatory standard for official statistics.

9.24.9 R Packages Used

Base R `stl()` for the canonical implementation; `forecast::stlf()` for STL-based forecasting; `forecast::mstl()` for multi-seasonal decomposition; `seasonal::seas()` for X-13 ARIMA-SEATS as the official-statistics alternative; `feasts::STL()` for the modern tidyverts interface.

9.24.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.24.11 See also — labs in this chapter

- Time series basics

9.25 Spectral Analysis

9.25.1 Introduction

Spectral analysis examines a time series in the frequency domain rather than the time domain. Periodicities appear as peaks in the power spectrum, slow trends dominate the low-frequency end, and white noise has a flat spectrum. The view is complementary to autocorrelation-based time-domain analysis: the spectral density and the autocovariance function are Fourier-transform pairs, so neither view contains information the other lacks, but each makes different features easier to see.

9.25.2 Prerequisites

A working understanding of Fourier analysis, the autocovariance function, and the time-frequency duality of stationary processes.

9.25.3 Theory

For a stationary process with autocovariance function γ_k, the spectral density is

$$S(f) = \sum_{k=-\infty}^{\infty} \gamma_k e^{-2\pi i f k},$$

defined on $f \in [-1/2, 1/2]$ for unit-spaced data. The periodogram is the sample-based estimator, computed as the squared magnitude of the FFT of the centred series divided by the series length. Raw periodograms are inconsistent (variance does not decrease with n), so smoothing — Bartlett, Welch, kernel-based, or multi-taper — is applied to produce stable spectral estimates.

A peak at frequency f indicates a cyclic component with period $1/f$ time units; the area under a frequency band is the variance contribution from that band.

9.25.4 Assumptions

The series is approximately stationary (or has been made so by detrending and differencing); sufficient length for frequency resolution $1/n$ to distinguish nearby peaks.

9.25.5 R Implementation

```
# Simulated series with two cycles
set.seed(2026)
t <- 1:500
y <- sin(2 * pi * t / 12) + 0.5 * sin(2 * pi * t / 4) + rnorm(500, 0, 0.3)

# Periodogram
spec_p <- spectrum(y, plot = FALSE)
plot(spec_p, main = "Raw periodogram")

# Smoothed
spec_s <- spectrum(y, spans = c(5, 5), plot = FALSE)
plot(spec_s, main = "Smoothed periodogram")
abline(v = 1/12, col = "red", lty = 2)
abline(v = 1/4,  col = "red", lty = 2)
```

9.25.6 Output & Results

`spectrum()` returns the periodogram with frequencies and spectral-density estimates, plus the chosen smoothing-window structure. The plot shows distinct peaks at $f = 1/12$ and $f = 1/4$, matching the simulated cycles.

9.25.7 Interpretation

A reporting sentence: "Spectral analysis of the simulated series with two embedded cycles produced peaks at frequencies $f = 0.083$ (period 12) and $f = 0.25$ (period 4) on the smoothed periodogram with spans = c(5, 5); the relative peak heights match the relative amplitudes of the simulated components." Always state the smoothing window in the figure caption; raw periodograms are noisy and often misread.

9.25.8 Practical Tips

- Smoothing (`spans = c(5, 5)` or similar) trades variance for bias; tune to balance peak detection against resolution.
- Frequency resolution is $1/n$; for closely spaced periodicities, longer series are essential.
- Use a log-log plot of frequency vs spectral density to expose power-law behaviour and broadband features.
- Multi-taper estimation (`multitaper::spec.mtm`) provides better bias-variance trade-off than spans-based smoothing for short series.
- Spectral analysis complements ACF / PACF; use both, especially when looking for cyclic versus trend structure.
- For non-stationary series, time-frequency methods (short-time Fourier, wavelets) generalise the stationary spectral framework.

9.25.9 R Packages Used

Base R `spectrum()` for periodograms with smoothing; `multitaper::spec.mtm()` for multi-taper estimation; `astsa::mvspec()` for multivariate spectral analysis; `WaveletComp` for time-frequency localisation when stationarity fails.

9.25.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.25.11 See also — labs in this chapter

- Time series basics

9.26 State-Space Models

9.26.1 Introduction

State-space models describe a time series via a latent state vector that evolves over time, with observations as a noisy function of the state. This general framework encompasses essentially every classical time-series model — ARIMA, exponential smoothing (ETS), structural time series, hidden Markov models, dynamic linear models — as special cases. The unified framework provides a consistent inference machinery (the Kalman filter for linear Gaussian models; particle and extended-Kalman filters for non-linear or non-Gaussian extensions) and a flexible specification language for custom models.

9.26.2 Prerequisites

A working understanding of linear algebra, Bayesian or likelihood inference, and basic familiarity with classical ARIMA / ETS forecasting.

9.26.3 Theory

The standard linear Gaussian state-space model has two equations:

Observation equation: $y_t = F_t x_t + v_t, \quad v_t \sim \mathcal{N}(0, V_t).$

State transition: $x_t = G_t x_{t-1} + w_t, \quad w_t \sim \mathcal{N}(0, W_t).$

The matrices (F_t, G_t, V_t, W_t) specify the model — they are typically constant in time but can vary deterministically. Inference proceeds in three modes: **filtering** estimates $x_t \mid y_{1:t}$ in real time; **smoothing** estimates $x_t \mid y_{1:T}$ using the full series; **prediction** extends the state forward beyond the data.

The Kalman filter computes both filtered and smoothed states by recursive Gaussian updates; non-linear or non-Gaussian extensions use the extended or unscented Kalman filter, the particle filter, or fully Bayesian MCMC.

9.26.4 Assumptions

For the standard Kalman filter: linear Gaussian observation and state equations. Non-linearity or non-Gaussianity requires extended methods.

9.26.5 R Implementation

```
library(dlm)

# Local level model (random walk + noise)
build_dlm ← function(theta)
  dlmModPoly(order = 1, dV = exp(theta[1]), dW = exp(theta[2]))

set.seed(2026)
y ← Nile
fit ← dlmMLE(y, parm = c(0, 0), build = build_dlm)
dlm_mod ← build_dlm(fit$par)

# Filtering
filt ← dlmFilter(y, dlm_mod)
smo  ← dlmSmooth(filt)

plot(y)
lines(dropFirst(smo$s), col = "red", lwd = 2)
```

9.26.6 Output & Results

`dlmMLE()` estimates the variance components by maximum likelihood. `dlmFilter()` and `dlmSmooth()` produce filtered and smoothed state estimates. The smoothed latent series captures the gradual level evolution; for the Nile, it shows the famous step-down around 1900 corresponding to the Aswan Low Dam construction.

9.26.7 Interpretation

A reporting sentence: “A local-level state-space model of Nile river flow fit by maximum likelihood estimated observation variance 15{,}100 and state-transition variance 1{,}470 (signal-to-noise ratio 0.10), favouring gradual state evolution; the smoothed latent series captured a clear level shift in the late 1890s.” Always state the model structure (local level, local linear trend, etc.) and the estimated variance components.

9.26.8 Practical Tips

- ARIMA and ETS have exact state-space representations; `forecast::Arima()` and `ets()` use them internally.
- Unknown latent states are estimated automatically by the Kalman filter; the user typically only specifies the model structure and estimates variance components by MLE.
- For non-linear or non-Gaussian models, use extended Kalman, unscented Kalman, or particle filters; `bayesforecast` and `bsts` provide Bayesian alternatives.
- Parameter estimation by MLE (`dlmMLE`) or Bayesian MCMC (`bsts`); for Bayesian setups, weakly informative priors on variance components stabilise inference when data are weak.
- State-space is the general framework; choose a specific model (local level, local linear trend, structural with seasonal, dynamic regression) based on substantive domain knowledge.

- For diagnostics, examine the standardised innovations (filter residuals); they should be approximately white noise under correct model specification.

9.26.9 R Packages Used

`dlm` for the canonical dynamic linear model framework; `KFAS` for fast Kalman filtering / smoothing including non-Gaussian extensions; `bsts` for Bayesian structural time series; `MARSS` for multivariate ARMA state-space models; `fable::SSM()` for tidyverts state-space modelling.

9.26.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.26.11 See also — labs in this chapter

- Time series basics

9.27 Stationarity Tests

9.27.1 Introduction

Stationarity — constant mean, variance, and autocovariance over time — is a fundamental prerequisite for most classical time-series models (ARIMA, VAR, spectral analysis). Non-stationary series typically need to be made stationary by differencing, detrending, or transformation before model fitting; forgetting this step produces nonsense estimates and invalid forecasts. Formal stationarity testing combines several tests with complementary nulls to give a robust verdict.

9.27.2 Prerequisites

A working understanding of time-series basics, autocorrelation, the difference between strict and weak stationarity, and the role of differencing in inducing stationarity.

9.27.3 Theory

The standard test battery comprises:

- **Augmented Dickey-Fuller (ADF)**: null = unit root (non-stationary), alternative = stationary or trend-stationary. Rejection supports stationarity.
- **KPSS**: null = stationarity (level or trend), alternative = unit root. Rejection supports non-stationarity.

- **Phillips-Perron (PP)**: non-parametric variant of ADF, robust to autocorrelation and heteroscedasticity in errors.

Pair them: stationary if ADF rejects *and* KPSS does not; non-stationary if ADF fails to reject *and* KPSS rejects. Disagreement (both reject, neither rejects) is ambiguous and may indicate near-unit-root behaviour, structural breaks, or fractional integration.

For series with structural breaks, standard tests are invalid; use Zivot-Andrews (`urca::ur.za()`) which extends the framework to allow one endogenous break.

9.27.4 Assumptions

Choice of test depends on the alternative — level vs trend stationarity for KPSS, drift vs trend specification for ADF. Adequate sample size for asymptotic critical values.

9.27.5 R Implementation

```r
library(tseries); library(urca)

x ← AirPassengers

# ADF
adf.test(log(x))

# KPSS
kpss.test(log(x))

# Phillips-Perron
pp.test(log(x))

# Differencing to achieve stationarity
ndiffs(log(x))   # forecast::ndiffs
```

9.27.6 Output & Results

Each test returns a statistic and p-value. `forecast::ndiffs()` and `nsdiffs()` automate the differencing decision by running KPSS or other tests at each candidate differencing order until stationarity is supported.

9.27.7 Interpretation

A reporting sentence: "On the log-transformed AirPassengers series, ADF failed to reject the unit-root null ($p = 0.99$) and KPSS rejected level stationarity ($p < 0.05$); both agree the series is non-stationary. After first-differencing plus seasonal differencing (order $(0,1,0)(0,1,0)_{12}$), all three tests supported stationarity, justifying ARIMA modelling on the doubly-differenced series." Always pair ADF with KPSS and report both verdicts.

9.27.8 Practical Tips

- Run ADF and KPSS together; their conclusions should agree, and disagreement is itself diagnostic information.

- For series with a clear trend, test trend-stationarity explicitly (`kpss.test(x, null = "Trend")` and `adf.test` with trend specification).
- Structural breaks invalidate standard tests; use Zivot-Andrews (`urca::ur.za()`) when a break is plausible.
- `forecast::ndiffs()` and `nsdiffs()` automate the choice of regular and seasonal differencing orders for ARIMA model preparation.
- Visual inspection of the series — plotting it and looking for trends, level shifts, or changing variance — usually suggests the right transformation before any test is run.
- For panel data (multiple time series), use Im-Pesaran-Shin or Levin-Lin-Chu panel unit-root tests; running univariate tests on each series ignores the panel structure.

9.27.9 R Packages Used

`tseries::adf.test()`, `kpss.test()`, `pp.test()` for the standard battery; `urca::ur.df()`, `ur.kpss()`, `ur.pp()`, `ur.za()` for richer output and structural-break extensions; `forecast::ndiffs()` and `nsdiffs()` for automated differencing decisions; `feasts::unitroot_kpss()` and `unitroot_ndiffs()` for the modern tidyverts interface.

9.27.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.27.11 See also — labs in this chapter

- Time series basics

9.28 Time Series: Introduction

9.28.1 Introduction

A time series is a sequence of observations indexed by time. Time-series analysis decomposes the series into four interpretable components — trend, seasonality, cyclicality, and noise — and models each. The decomposition framework underpins virtually every classical method: ARIMA models the noise after differencing out trend and seasonality; ETS captures all four components in a state-space form; Bayesian structural time series isolates them as latent processes. Understanding the four-component vocabulary is the foundational step before reaching for any specific model.

9.28.2 Prerequisites

A working understanding of basic statistics, the `ts` class in R, and the difference between cross-sectional and longitudinal data structures.

9.28.3 Theory

The classical additive decomposition is

$$y_t = T_t + S_t + C_t + \varepsilon_t,$$

with components: trend T (long-term level change), seasonal S (regular fixed-period variation, e.g. monthly cycle), cycle C (irregular longer-than-seasonal swings, e.g. business cycles), and noise ε (residual stochastic variation). The multiplicative form $y_t = T_t \cdot S_t \cdot C_t \cdot \varepsilon_t$ applies when seasonal amplitude grows with the level; it is additive on the log scale.

In practice, the cycle is often combined with the trend (giving "trend-cycle" as one component), leaving three components in most decomposition implementations: trend-cycle, seasonal, and remainder.

9.28.4 Assumptions

Time order is preserved (no shuffling); regular sampling intervals (irregular sampling requires `zoo` / `xts` or modelling the spacing); the chosen decomposition (additive vs multiplicative) matches the data.

9.28.5 R Implementation

```r
library(forecast)

data(AirPassengers)
plot(AirPassengers, main = "Monthly airline passengers")

# Classical decomposition
dec ← decompose(AirPassengers, type = "multiplicative")
plot(dec)

# STL: loess-based decomposition
stl_dec ← stl(log(AirPassengers), s.window = "periodic")
plot(stl_dec)
```

9.28.6 Output & Results

`decompose()` returns the classical additive or multiplicative decomposition; `stl()` returns the loess-based seasonal-trend decomposition. The four-panel plot shows observed, trend, seasonal, and remainder components stacked vertically.

9.28.7 Interpretation

A reporting sentence: "The monthly air-passenger series exhibited a clear upward trend (approximately 5 % per year), a stable annual seasonal cycle peaking in summer, and noise that grew with the level (motivating the log transformation). STL decomposition on the log scale produced trend, seasonal, and approximately white-noise remainder components." When introducing a time series, describe each component visible in the data.

9.28.8 Practical Tips

- Log-transform series whose variance grows with the level; the resulting decomposition on the log scale is additive and well-behaved.
- STL (`stl()`) is more robust than classical `decompose()` ; the latter is sensitive to outliers and assumes fixed seasonal patterns.
- After decomposition, examine the remainder series with ACF / PACF plots and Ljung-Box; substantial autocorrelation indicates missed structure.
- The modern tidy workflow uses `tsibble` for time-aware tibbles, `feasts` for decomposition and diagnostics, and `fable` for forecasting; together they replace much of the older `forecast` workflow.
- For irregular sampling intervals, use `zoo` or `xts` time series; classical decomposition assumes regular sampling.
- For multiple seasonalities (e.g. weekly + yearly in retail data), `forecast::mstl()` produces a multi-layer decomposition that single-seasonal STL cannot.

9.28.9 R Packages Used

Base R `ts()` , `decompose()` , and `stl()` ; `forecast` for the broader classical workflow including `auto.arima` , `forecast` , `mstl` ; `tsibble` , `feasts` , `fable` for the modern tidyverts ecosystem; `zoo` and `xts` for irregularly-spaced time series.

9.28.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.28.11 See also — labs in this chapter

- Time series basics

9.29 VECM and Cointegration

9.29.1 Introduction

Two or more non-stationary time series are *cointegrated* if some linear combination of them is stationary. The classic example is interest rates at different maturities: each rate is non-stationary (random-walk-like), but spreads are mean-reverting. A vector error-correction model (VECM) captures the short-run dynamics of cointegrated series while respecting the long-run equilibrium relationship; the Johansen procedure tests how many cointegrating relations exist and estimates them jointly with the dynamic parameters.

9.29.2 Prerequisites

A working understanding of vector autoregressive (VAR) models, unit roots, and the I(1)/I(0) terminology for integrated and stationary series.

9.29.3 Theory

The VECM is

$$\Delta y_t = \Pi y_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + \varepsilon_t.$$

The matrix Π encodes the long-run relationships; its rank equals the number of cointegrating relations. Johansen's approach tests the rank via two statistics:

- **Trace test**: $\lambda_{\text{trace}}(r) = -T \sum_{i=r+1}^{k} \log(1 - \hat{\lambda}_i)$ tests $H_0 : \text{rank} \le r$.
- **Maximum eigenvalue test**: $\lambda_{\max}(r) = -T \log(1 - \hat{\lambda}_{r+1})$ tests $H_0 : \text{rank} = r$ vs $H_1 : \text{rank} = r + 1$.

The two often agree; when they disagree, the trace test is generally preferred. Once the rank is determined, the cointegrating vectors and adjustment speeds can be estimated.

9.29.4 Assumptions

Each series is individually I(1) (verifiable by ADF and KPSS); k-variate Gaussian innovations; the lag length p is correctly specified.

9.29.5 R Implementation

```
library(urca); library(vars)

data(Canada)
jo ← ca.jo(Canada, ecdet = "const", type = "eigen", K = 2, spec = "transitory")
summary(jo)

# Fit VECM with rank r = 1 (if supported by test)
vecm ← cajorls(jo, r = 1)
vecm
```

9.29.6 Output & Results

`ca.jo()` returns the eigenvalue and trace test statistics per candidate rank, along with critical values at 10/5/1 % levels. `cajorls()` fits the VECM at the chosen rank and returns the cointegrating vector and adjustment coefficients.

9.29.7 Interpretation

A reporting sentence: "Johansen's trace test supported one cointegrating relationship in the Canada macroeconomic data (test statistic 32.8 vs 5 % critical value 25.3, $p < 0.05$); the corresponding cointegrating vector linked money demand and interest rates with

adjustment speeds of -0.10 for money demand toward equilibrium." Always report both trace and max-eigenvalue results, and the chosen lag length.

9.29.8 Practical Tips

- Confirm each series is I(1) via ADF / KPSS before cointegration testing; cointegration is meaningful only between integrated series.
- The `ecdet` argument specifies whether deterministic terms (constant, trend) enter inside the cointegrating relation; choose based on visible long-run drift.
- Engle-Granger two-step is simpler for two series and gives essentially equivalent conclusions; Johansen generalises to k variables.
- Report both trace and max-eigenvalue tests; they typically agree but can disagree when one or two cointegrating relations are weak.
- For short samples (under 100 observations) and many variables, Johansen loses power; reduce the dimension or use bias-corrected critical values.
- Once the VECM is fitted, impulse-response functions and forecast-error variance decompositions on `vec2var(vecm)` extend the standard VAR diagnostic toolkit.

9.29.9 R Packages Used

`urca` for `ca.jo()`, `cajorls()`, and Johansen-related cointegration tests; `vars` for VAR modelling and `vec2var()` to convert a VECM back to VAR form for IRF analysis; `tsDyn::VECM()` for an alternative VECM implementation; `urca::ca.po()` for the Phillips-Ouliaris cointegration tests.

9.29.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.29.11 See also — labs in this chapter

- Time series basics

9.30 Wavelet Analysis

9.30.1 Introduction

Wavelet analysis decomposes a time series into components localised in both time and frequency simultaneously. Unlike the Fourier transform, which assumes stationarity and produces a single spectrum for the entire series, wavelets capture cyclic components that appear, shift, or disappear over time — essential for non-stationary signals such as EEG

recordings, climate proxies, or any series whose periodic content evolves. The output is a scalogram, a 2D map of power across time and frequency that reveals the time-varying structure that Fourier methods miss.

9.30.2 Prerequisites

A working understanding of spectral analysis and Fourier transforms, the concept of time-frequency localisation, and basic familiarity with the trade-off between time and frequency resolution.

9.30.3 Theory

The continuous wavelet transform (CWT) convolves the signal with scaled and shifted copies of a mother wavelet ψ:

$$W(a,b) = \int y(t)\,\bar{\psi}\left(\frac{t-b}{a}\right)\frac{\mathrm{d}t}{\sqrt{|a|}}.$$

The scale a controls the wavelet's "width" and corresponds inversely to frequency; b is the time shift. The squared magnitude $|W(a,b)|^2$ is the wavelet power at time b and scale a. Common mother wavelets include Morlet (oscillatory, good time-frequency localisation), Haar (step-function), Daubechies (orthogonal, used in compression). The scalogram visualises $|W|^2$ as a heatmap over the time-frequency plane.

The cone of influence delimits regions of the scalogram free of edge artefacts; power within the cone is reliable, power outside should be discounted.

9.30.4 Assumptions

Depends on the wavelet choice; minimal beyond a sufficiently long series. The discrete wavelet transform (DWT) requires series length to be a power of 2; the CWT does not.

9.30.5 R Implementation

```
library(WaveletComp)

set.seed(2026)
t ← 1:500
y ← sin(2 * pi * t / 50) * (t < 250) + sin(2 * pi * t / 20) * (t ⩾ 250) +
     rnorm(500, 0, 0.3)

df ← data.frame(date = t, x = y)
wt ← analyze.wavelet(df, "x", loess.span = 0, dt = 1,
                     dj = 1/50, lowerPeriod = 8,
                     upperPeriod = 128, make.pval = FALSE, verbose = FALSE)

wt.image(wt, main = "Wavelet power spectrum")
```

9.30.6 Output & Results

`analyze.wavelet()` returns a list containing the wavelet power, phase, and statistical-significance information. `wt.image()` plots the scalogram as a 2D heatmap with the cone of influence overlaid.

9.30.7 Interpretation

A reporting sentence: "The wavelet power spectrum revealed a 50-time-step cycle in the first half of the series and a 20-time-step cycle in the second half — a non-stationary feature invisible to a global Fourier analysis. The cone of influence excluded the first and last 30 time points from interpretation due to edge effects." Always describe the time-frequency structure visible and acknowledge the cone of influence.

9.30.8 Practical Tips

- Choose the wavelet to suit the signal: Morlet for oscillatory components, Mexican-hat for localised peaks, Daubechies for compression and denoising.
- Power outside the cone of influence is unreliable; mask it or interpret with caution.
- Statistical significance via red-noise surrogates (`make.pval = TRUE` in `WaveletComp`) tests power against an AR(1) null at each scale and time.
- The discrete wavelet transform (DWT, in `wavethresh`) supports compression and denoising; CWT is for visualisation and analysis.
- For EEG, fMRI, and other neurophysiological data, wavelets are the standard time-frequency tool; specialised packages (`eegkit`, `fmri.wavelets`) extend the framework.
- For multi-channel time-frequency analysis, cross-wavelet coherence and wavelet phase quantify time-varying co-movements between series.

9.30.9 R Packages Used

`WaveletComp` for `analyze.wavelet()`, `wt.image()`, and biwavelet coherence; `wavelets` and `wavethresh` for discrete wavelet transforms and threshold-based denoising; `dplR` for tree-ring-style wavelet analysis; `Rwave` for additional wavelet families and inversion.

9.30.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.30.11 See also — labs in this chapter

- Time series basics

9.31 White Noise Tests

9.31.1 Introduction

After fitting a time-series model — ARIMA, ETS, regression with ARIMA errors — the residuals should look like white noise: zero mean, constant variance, and no autocorrelation. White-noise tests check these properties formally and are the standard residual-diagnostic battery for time-series modelling. Significant rejection indicates the model has missed structure that should motivate refinement; failure to reject supports model adequacy.

9.31.2 Prerequisites

A working understanding of ARIMA models, residual diagnostics, and the autocorrelation function (ACF) as the visual counterpart to formal autocorrelation tests.

9.31.3 Theory

Three test families dominate practice:

- **Portmanteau tests** (Box-Pierce, Ljung-Box) aggregate squared sample autocorrelations over the first h lags into a single chi-squared statistic. Under the white-noise null, $Q \sim \chi^2_{h-p-q}$ where $p+q$ is the number of fitted ARMA parameters. The Ljung-Box variant has a small-sample correction and is preferred over Box-Pierce.
- **Turning-points test**: counts the number of local maxima and minima in the residual series; under iid, has known mean $2(n-2)/3$ and variance, giving an approximate Normal test for independence.
- **Runs test**: counts runs of consecutive same-sign residuals; tests independence without distributional assumptions.

A failure on Ljung-Box at any lag indicates missed autocorrelation that the model should address; a failure on the runs or turning-points test indicates more general non-randomness.

9.31.4 Assumptions

Null: residuals are iid white noise. The tests have different power against autocorrelation, heteroscedasticity, and non-linearity; no single test detects all departures, so multiple tests are run as a battery.

9.31.5 R Implementation

```r
library(forecast)

set.seed(2026)
fit <- arima(AirPassengers, order = c(0, 1, 1), seasonal = c(0, 1, 1))
resid_fit <- residuals(fit)

# Ljung-Box
Box.test(resid_fit, lag = 24, type = "Ljung-Box", fitdf = 2)

# Visual + ARIMA diagnostic
checkresiduals(fit)
```

9.31.6 Output & Results

`Box.test()` returns the test statistic and p-value; `checkresiduals()` aggregates the Ljung-Box test with ACF, histogram, and time-series plots into a single diagnostic display.

9.31.7 Interpretation

A reporting sentence: "Ljung-Box at lag 24 on the residuals from the ARIMA(0,1,1)(0,1,1)

$$12$$

fit gave $Q = 28.1$ on 22 degrees of freedom ($p = 0.17$); the ACF showed no isolated significant lags, the histogram was approximately Normal, and the residual time-series plot showed no remaining structure. Residuals are consistent with white noise." Always pair the numerical test with the ACF plot.

9.31.8 Practical Tips

- Choose the maximum lag h around $\log n$ to a small multiple thereof; for seasonal models, set h at least one full seasonal period to detect missed seasonal autocorrelation.
- Use `fitdf = p + q` for ARMA(p, q) fitted models; the test ignores the parameter cost otherwise and is anti-conservative.
- A significant Ljung-Box after model fitting indicates missed structure; add AR or MA terms, seasonal terms, or consider a different model family.
- Pair the formal Ljung-Box test with a visual ACF inspection; isolated significant lags can be substantively meaningful even if the omnibus test passes.
- `forecast::checkresiduals()` is the convenient one-call diagnostic battery; use it routinely after every time-series fit.
- For non-Normal residuals, a Jarque-Bera or Shapiro-Wilk test on the residuals complements the ACF-based tests.

9.31.9 R Packages Used

Base R `Box.test()` for Ljung-Box and Box-Pierce; `forecast::checkresiduals()` for the integrated diagnostic; `tseries::runs.test()` for the runs test; `randtests::turning.point.test()` for the turning-points test; `feasts::ljung_box()` for

the modern tidyverts implementation.

9.31.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.31.11 See also — labs in this chapter

- Time series basics

9.32 X-13ARIMA-SEATS Decomposition

9.32.1 Introduction

X-13ARIMA-SEATS is the seasonal-adjustment workhorse of official statistics. It is the Census Bureau standard, used by Eurostat, the Bank of England, and most national statistical agencies for producing officially published seasonally adjusted economic time series. The combination of RegARIMA-based pre-adjustment (handling outliers, trading-day effects, Easter effects) with either X-11 filter-based or SEATS model-based decomposition makes it the most sophisticated decomposition tool routinely available — and the standard against which simpler methods (STL, classical decomposition) are benchmarked.

9.32.2 Prerequisites

A working understanding of seasonal decomposition, SARIMA models, and the role of calendar effects (trading days, leap years, moving holidays) in economic time series.

9.32.3 Theory

X-13 has two main computational stages:

1. **RegARIMA pre-adjustment**: fit a regression-with-ARIMA-errors model that absorbs outliers (additive, level shift, temporary change), trading-day effects, leap-year effects, and Easter effects. The pre-adjustment removes these from the original series.

2. **Decomposition**: either X-11-style iterative moving averages or SEATS (Signal Extraction in ARIMA Time Series), a model-based decomposition derived from the RegARIMA spectral structure. SEATS is preferred for series with stable seasonal patterns; X-11 is more robust to deviations from the fitted ARIMA.

The output includes the seasonally adjusted series, the trend-cycle, the seasonal component, and the irregular component, plus diagnostics for stability, residual autocorrelation, and seasonal-adjustment quality.

9.32.4 Assumptions

Monthly or quarterly data with several years of history (typically at least 3–5 years for stable seasonal-factor estimation); the RegARIMA pre-adjustment captures the calendar effects that matter for the series.

9.32.5 R Implementation

```
library(seasonal)

x ← AirPassengers
m ← seas(x)
summary(m)

plot(m)
series(m, "s10")   # seasonal factors
series(m, "s12")   # seasonally adjusted
series(m, "s11")   # trend-cycle
```

9.32.6 Output & Results

`seas()` runs the X-13 binary in the background and returns a `seas` object with the RegARIMA model summary, the various decomposed series accessible via `series()`, and a panel of diagnostics. `view(m)` (or `inspect(m)`) opens an interactive dashboard for diagnostic exploration.

9.32.7 Interpretation

A reporting sentence: "X-13ARIMA-SEATS seasonal adjustment of monthly air-passenger counts identified a $(0,1,1)(0,1,1)_{12}$ RegARIMA structure with no significant outliers; the SEATS-based decomposition produced a smooth seasonally adjusted trend with stable seasonal factors that gradually amplified across the series, consistent with the multiplicative seasonality." Always state the chosen ARIMA structure and the decomposition method (SEATS or X-11).

9.32.8 Practical Tips

- The `seasonal` package automatically installs the X-13 binary on first use; verify with `seasonal::checkX13()`.
- X-13 is the official standard for seasonal adjustment of economic series; required by some regulators and statistical agencies.
- `view(m)` opens an interactive HTML dashboard with all diagnostics — extremely useful for exploring complex decompositions.
- Trading-day, Easter, and outlier regressors are added automatically in default mode but can be customised; well-chosen regressors substantially improve fit for retail

and financial series.
- For non-economic series or series with non-standard seasonal periods, STL (`stl()` or `mstl()`) is a simpler alternative without the calendar-effect machinery.
- Compare X-13 against STL and classical decomposition; large discrepancies signal calendar or outlier effects that one method handles better than another.

9.32.9 R Packages Used

`seasonal` for `seas()` , `series()` , and the X-13 interface; `seasonalview` for the interactive HTML dashboard; `forecast::stl()` and `mstl()` for simpler decomposition alternatives; `feasts::X_13ARIMA_SEATS()` for the modern tidyverts wrapper.

9.32.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

9.32.11 See also — labs in this chapter

- Time series basics

9.33 Week 2, Session 5 — Time series basics

Inference lab using the five-step template: Hypothesis → Visualise → Assumptions → Conduct → Conclude.

9.33.1 Learning objectives

- Decompose a seasonal time series into trend, seasonal, and residual components.
- Fit an ARIMA model automatically with `forecast::auto.arima` .
- Detect a mean shift using `changepoint::cpt.mean` .

9.33.2 Prerequisites

Regression; basic familiarity with the `ts` class.

9.33.3 Background

A time series differs from cross-sectional data in one crucial way: observations close in time are correlated with one another, so standard errors based on independence are wrong. Classical decomposition splits a series into a slowly varying trend, a periodic seasonal component, and a residual. ARIMA models capture the residual autocorrelation with autoregressive and moving-average terms, possibly after differencing to remove a unit root.

Change-point detection asks a different question: given a series, when (if ever) did the underlying mean or variance shift? The `changepoint` package implements several methods; `cpt.mean` with a penalised likelihood criterion is a common starting point. In epidemiology, change-points flag outbreaks, policy changes, and data-collection transitions.

A seasonally adjusted series is not the same as a detrended series. Seasonal adjustment removes only the regular period; detrending removes the slow evolution. Most anomalies you care about (an outbreak, an intervention) live in what remains.

9.33.4 Setup

```
library(tidyverse)
library(forecast)
library(changepoint)
set.seed(42)
theme_set(theme_minimal(base_size = 12))
```

9.33.5 1. Hypothesis

A simulated monthly series has a linear trend, a yearly seasonal cycle, and a mean shift at month 80. Decomposition, ARIMA, and change-point detection should each recover an interpretable piece.

9.33.6 2. Visualise

```
months ← 144
t  ← seq_len(months)
trend ← 0.08 * t
season ← 4 * sin(2 * pi * t / 12)
shift  ← if_else(t ⩾ 80, 5, 0)
y      ← 20 + trend + season + shift + rnorm(months, 0, 1.5)
ts_y   ← ts(y, frequency = 12, start = c(2014, 1))

autoplot(ts_y) +
  labs(x = "Year", y = "Simulated rate")
```

9.33.7 3. Assumptions

STL decomposition assumes the seasonal period is fixed and known. ARIMA assumes a linear, time-invariant generating process after differencing. `cpt.mean` assumes the variance is approximately constant — change-points in variance would need `cpt.var`.

9.33.8 4. Conduct

```
decomp ← stl(ts_y, s.window = "periodic")
autoplot(decomp)
```

```
fit_arima ← auto.arima(ts_y)
fit_arima
```

```
fc ← forecast(fit_arima, h = 12)
autoplot(fc) +
  labs(x = "Year", y = "Simulated rate")
```

```
cpt ← cpt.mean(as.numeric(ts_y), method = "PELT")
cpts(cpt)
```

```
tibble(t = t, y = as.numeric(ts_y)) ▷
  ggplot(aes(t, y)) +
  geom_line() +
  geom_vline(xintercept = cpts(cpt), linetype = "dashed",
             colour = "firebrick") +
  labs(x = "Month index", y = "Rate")
```

9.33.9 5. Concluding statement

STL cleanly separated a linear trend, a 12-month seasonal cycle, and a residual containing the simulated mean shift. `auto.arima` selected `fit_arima$arma ▷ paste(collapse = ", ")` (p, q, P, Q, frequency, d, D). PELT detected change-points at months `paste(cpts(cpt), collapse = ", ")`, close to the simulated truth of 80.

9.33.10 Common pitfalls

- Using daily ARIMA on weekly-seasonal data without telling the model the period.
- Calling every bump a change-point; penalise harder and re-run.
- Using `lm()` on a time series as if residuals were independent.
- Forecasting far beyond the observed period without widening the intervals in the report.

9.33.11 Further reading

- Hyndman RJ, Athanasopoulos G. *Forecasting: Principles and Practice*.
- Killick R, Fearnhead P, Eckley IA (2012), *Optimal detection of changepoints with a linear computational cost*.
- Box GEP, Jenkins GM. *Time Series Analysis: Forecasting and Control*.

9.33.12 Session info

```
sessionInfo()
```

9.33.13 See also — chapter index

- Methods in this chapter
- Find your method (Chapter 0)