STAT 443 Report

Chow Sheng Liang & Eric Li

2025-07-20

Part 1

A fundamental part of forecasting is understanding how a variable relates to its own past, this is because most real-world processes are not memoryless. For example, daily temperature tends to evolve smoothly over time, electricity demand shows strong daily and weekly cycles, and stock market volatility clusters over consecutive days.

Stationarity is a core concept that provides the theoretical framework to model such temporal relationships. Formally we define a time series as :

$$\{X_t\}_{t\in\mathbb{Z}} \quad \text{or} \quad \{X_1, X_2, \ldots, X_T\}$$

where each X_t is a random variable representing the value of a process at time t. The model we are trying to approximate here would differ slightly from linear regression:

$$\hat{f}(\{X_1, \cdots, X_t\}) \approx f(\{X_1, \cdots, X_t\}) = X_{t+1}$$

Stationarity is satisfied by the following 3 conditions:

1. $E(|X_t|^2) < \infty \quad$ for all $t \in T$
2. $E(X_t) = \mu \quad$ for all $t \in T$
3. $Cov(X_r, X_s) = Cov(X_{r+t}, X_{s+t}) \quad$ for all $\; r, s, r+t, s+t \in T$

These conditions combine to ensure that the statistical properties of a process remain **consistent over time**, allowing us to estimate future values based on past patterns without worrying about structural drift. *(Theoretically, stationarity by itself is not enough, we would also need causality that ensures a variable is* ***only influenced by its past****)*

Just as linear regression builds on linearity, time series analysis builds on stationarity. Under this assumption, we can fit structured models like AR, MA, or $ARMA$, which linearly relate a variable to its past values, much like how linear models relate outcomes to transformed inputs.

In practice, as you may have noticed with linear regression, we often do not observe data that adhere to the linearity/normality assumption and we typically transform y or X. In the same spirit, we transform a time series with differencing and lag transformations:

- Backward operator: $BX_t = X_{t-1}$
- d^{th} order differencing: $Y_t := \nabla^d X_t = (1-B)^d X_t$
- Seasonal differencing: $\nabla^s X_t := (1-B^s)X_t$

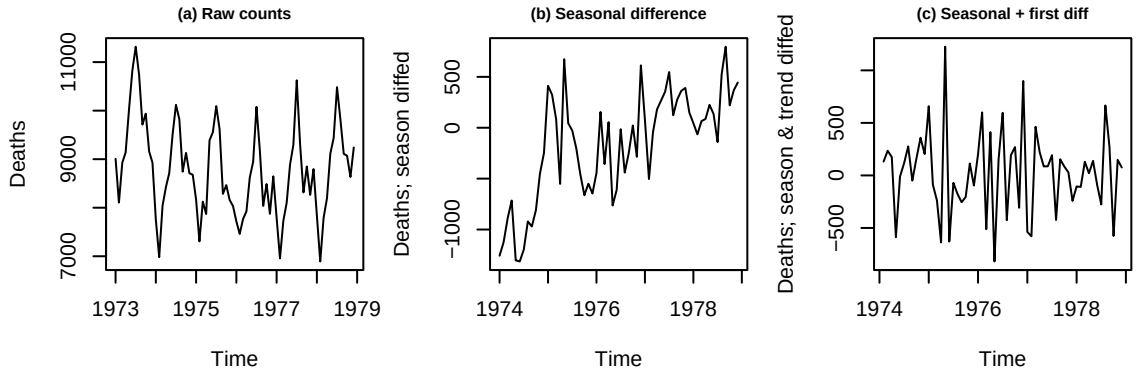

Figure 1: Transformation of US accidental-death counts

So in summary, just as linearity is crucial for linear regression, stationarity is a core assumption in forecasting as it restricts the function space so we can meaningfully fit and extrapolate a process' temporal relationships independent of external changes.

Part 2

```r
project_data <- readRDS("projectdata.rds")
pred_num_ahead <- 100
conf_int_mult <- 1.96

# Model fitting
ar1_model <- Arima(project_data, order = c(1, 0, 0))
ar1_prediction <- predict(ar1_model, n.ahead = pred_num_ahead)

arima_model <- Arima(project_data, order = c(0, 1, 0))
arima_prediction <- predict(arima_model, n.ahead = pred_num_ahead)

# Get prediction time range
freq <- frequency(project_data)
last_time <- time(project_data)[length(project_data)]
future_time <- seq(from = last_time + 1/freq, by = 1/freq, length.out = pred_num_ahead)

# Construct vectors for CI
ar1_upper <- ar1_prediction$pred + conf_int_mult * ar1_prediction$se
ar1_lower <- ar1_prediction$pred - conf_int_mult * ar1_prediction$se

arima_upper <- arima_prediction$pred + conf_int_mult * arima_prediction$se
arima_lower <- arima_prediction$pred - conf_int_mult * arima_prediction$se

data_mean = mean(project_data)
data_var_pos = data_mean + conf_int_mult * sd(project_data)
data_var_neg = data_mean - conf_int_mult * sd(project_data)

# Layout
par(mfrow = c(1, 2), mar = c(4, 4, 2, 2))

# AR(1)
plot(project_data, xlim = c(start(project_data)[1], future_time[pred_num_ahead]),
     ylim = c(-20, 10), main = "(a)", xlab = "Time", ylab = "Value")
lines(future_time, ar1_prediction$pred, col = "red")
lines(future_time, ar1_upper, col = "blue")
lines(future_time, ar1_lower, col = "blue")
abline(h = data_mean, col = "red", lty = 2)
abline(h = data_var_pos, col = "blue", lty = 2)
abline(h = data_var_neg, col = "blue", lty = 2)

# ARIMA(0,1,0)
plot(project_data, xlim = c(start(project_data)[1], future_time[pred_num_ahead]),
     ylim = c(-20, 10), main = "(b)", xlab = "Time", ylab = "Value")
lines(future_time, arima_prediction$pred, col = "red")
lines(future_time, arima_upper, col = "blue")
lines(future_time, arima_lower, col = "blue")
abline(h = data_mean, col = "red", lty = 2)
abline(h = data_var_pos, col = "blue", lty = 2)
abline(h = data_var_neg, col = "blue", lty = 2)
```

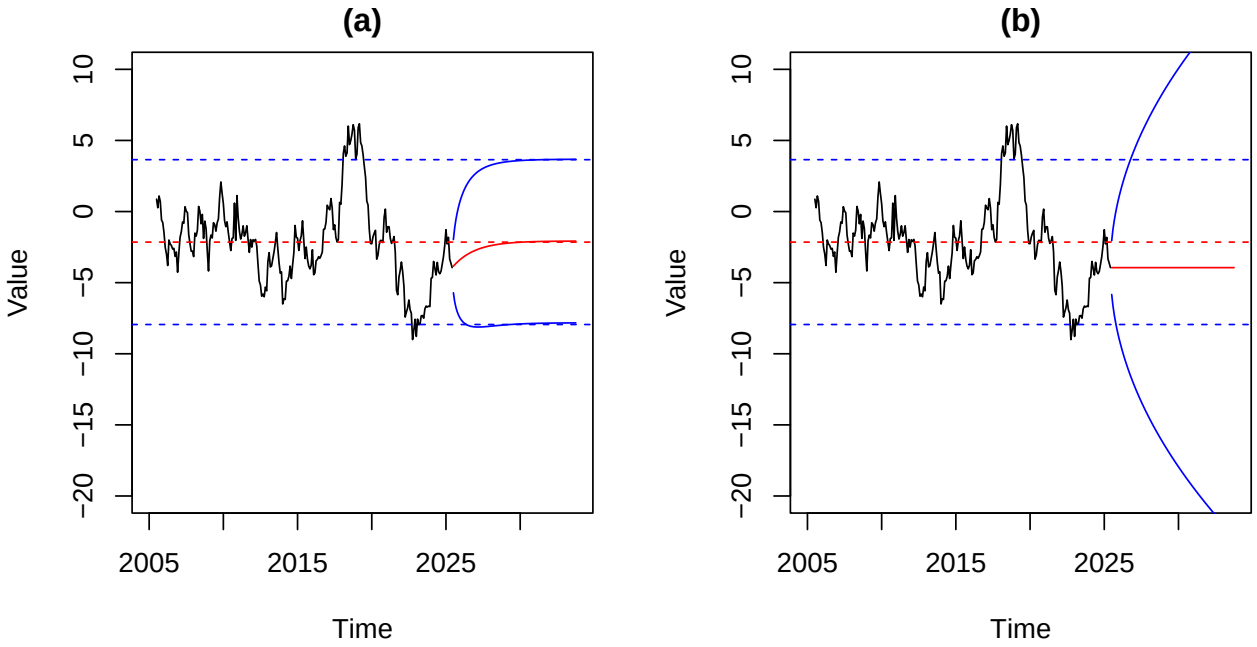

Figure 2: Panel (a) shows long term forecasts using an AR(1) model. Panel (b) shows long term forecasts using an ARIMA(0, 1, 0) model.

Through the Box-Jenkins methodology, we would first examine the time-series plot, ACF, and PACF plots to visually check for any reason to be suspicious of our stationary assumption. We can also numerically test for this using the Phillips-Perron test, where the null hypothesis is of a unit root → non-stationarity; which we hope to reject.

If we see a clear trend from the time-series plot or if the Phillips-Perron test suggests non-stationarity we may consider differencing it and re-evaluating. If we see a seasonal trend in the ACF(i.e. ACF spikes at regular periods) we may consider differencing with a lag that matches the period and re-evaluating.

Once the process is stationary, Box-Jenkins requires us to identify the model by fitting an appropriate order and validating with a criterion, then reiterating until the one with the best metric is found. However this entire process can be much easily done with `auto.arima()` through numerical evaluation. We can optionally provide `auto.arima()` with a ceiling for its search to optimize speed, `auto.arima()` will then automate the identification process by iteratively minimizing AIC across the possible `(p,d,q)(P,D,Q)` combinations with options to include/exclude the seasonal components as well as options to exhaustively search all permutations or do a stepwise search.

We will then visually validate this model by looking at its residual plot and residual ACF plot which we should expect to see evidence of a white noise process. If the residuals suggests otherwise, we should go back to step 1 and re-evaluate our model selection, and maybe even consider completely different models like ARCH, GARCH or simpler linear models like Holt-Winters.

Part 3

Overview

The goal of this simulation study is to assess the performance of two tools commonly used in time series modeling:

1. `auto.arima()`: Automatically selects and fits (S)ARIMA models by minimizing information criterias, aiming to recover the correct model structure and generate accurate forecasts.
2. `pp.test()`: Performs the Phillips-Perron test to detect the presence of a unit root (i.e. non-stationarity).

These tools are evaluated on their reliability in identifying and forecasting a known stationary, non-seasonal AR(1) process with $\phi = 0.95$ and $\sigma = 0.5$, using a Monte Carlo simulation design. Forecast accuracy is assessed on a holdout test set of 20% for each realization.

Background

We consider an AR(1) process where each value depends on its immediate past and a normally distributed noise term: $X_t = 0.95X_{t-1} + Z_t$, with $Z_t \sim \mathcal{N}(0, 0.5^2)$. A stationary process' outcomes only depend on historic data, not the time sampled. Stationarity can be assessed by examining the roots of the characteristic polynomial (See [Brockwell and Davis, 2016]); in this case, the root is 1.05 which is close to 1, this complicates detection. So although this process is technically stationary, its high dependence on the previous term makes it behave almost non-stationarily, often drifting from the mean in finite samples. We use this process to evaluate the performance of `auto.arima()` and `pp.text()` in identifying and modeling such ambitious cases.

Design

To best reflect real-world scenarios where users may not know whether a series is stationary or seasonal, we purposefully do not restrict the model search space in `auto.arima()` by setting:

- `stationary = FALSE` (allow but not forcing differencing)
- `seasonal = TRUE` (allow but not forcing seasonal terms)

to explore how robust the model selection is under loose assumptions.

We investigate how performance varies across three axis:

- Information criteria: AIC, AICc, and BIC
- Search strategies: stepwise vs exhaustive
- Series lengths: 50, 100, and 250 observations

Evaluation Metrics

Our evaluation uses two metrics averaged over 1000 simulations:

- Model identification accuracy:
  - `auto.arima()`: proportion of simulations where the correct ARIMA order (1,0,0) is selected
  - `pp.test()`: proportion of correct stationarity classifications (rejecting unit root, $p \leq 0.05$)
- Forecasting accuracy (for `auto.arima()` only):
  - Root Mean Square Error (RMSE) on the test set (20% of each realization)

Results

Table 1: auto.arima() Performance Grouped by Series Length and Search Strategy

	Series Length = 50		Series Length = 100		Series Length = 250	
Criterion	50_Accuracy	50_RMSE	100_Accuracy	100_RMSE	250_Accuracy	250_RMSE
Stepwise						
AICc	0.194	0.977	0.190	1.267	0.128	1.639
AIC	0.175	0.983	0.182	1.269	0.124	1.638
BIC	0.216	0.958	0.236	1.250	0.193	1.635
Exhaustive						
AICc	0.165	0.985	0.165	1.270	0.108	1.631
AIC	0.133	0.997	0.156	1.272	0.105	1.631
BIC	0.208	0.960	0.235	1.249	0.192	1.636

Table 2: pp.test() Performance across Series Lengths

Series Length	PP Accuracy
50	0.054
100	0.114
250	0.281

Analysis

The performance of `auto.arima()` in identifying the correct model order is generally underwhelming coming in at ~20%, which is not entirely unexpected given the inherent difficulty of the task. Among the information criteria, AICc consistently outperforms AIC in both order identification and forecast accuracy, which aligns with expectations due to AICc's robustness in accounting for smaller samples.

Surprisingly, exhaustive search performs worse than stepwise search across all metrics. This contradicts the intuition that a global optimal would yield better results, and may suggest potential overfitting on a large search space or some inefficiency within the exhaustive search algorithm. Even more unexpectedly, `auto.arima()` did not improve in identification accuracy as the sample size increases, this is unlikely to be due to random chance as we ran 1000 simulations.

BIC emerges as the strongest criterion overall, outperforming AIC and AICc in both identification and forecast accuracy across both search strategies. Despite differences in order accuracy, RMSE appears similar across all criteria and search strategies as the series length increases, suggesting a convergence effect in forecast error that is not reflected in model selection accuracy.

For unit root testing, the `pp.test()` improves with longer series, which is consistent with the statistical theory it is based on(SLLN).

In conclusion, several counter-intuitive results were observed, such as exhaustive search under-performing and order accuracy not improving while forecast accuracy becomes more stable with more data. Future work could investigate why RMSE stabilizes even when model identification does not improve, and whether using `auto.arima()` for model selection but deferring to `pp.test()` to validate differencing(if any) can yield better overall performance.

Part 4

```r
# Deals with weird auto.arima() object
get_order <- function(auto_arima_fit) {
  ord <- arimaorder(auto_arima_fit)

  safe_index <- function(vec, i) {
    if (i <= length(vec)) vec[i] else 0
  }

  p <- safe_index(ord, 1)
  d <- safe_index(ord, 2)
  q <- safe_index(ord, 3)
  P <- safe_index(ord, 4)
  D <- safe_index(ord, 5)
  Q <- safe_index(ord, 6)
  s <- safe_index(ord, 7)

  # Check if there is any seasonal component
  if (P != 0 || D != 0 || Q != 0) {
    return(paste0("ARIMA(", p, ",", d, ",", q, ")(", P, ",", D, ",", Q, ")[", s, "]"))
  } else {
    return(paste0("ARIMA(", p, ",", d, ",", q, ")"))
  }
}

ar1_simulation_experiment <- function(n_sim = 1000, series_length = 100) {
  # Configs
  set.seed(443)
  phi <- 0.95
  sigma <- 0.5
  train_test_split <- 0.8

  # Define model configs (default is stationary = FALSE and seasonal = TRUE),
  #  Parallel is used for multi-core processing, only for exhaustive search
  configs <- list(
    stepwise_true_aicc  = list(ic = "aicc", stepwise = TRUE, parallel = FALSE),
    stepwise_true_aic   = list(ic = "aic", stepwise = TRUE, parallel = FALSE),
    stepwise_true_bic   = list(ic = "bic", stepwise = TRUE, parallel = FALSE),
    stepwise_false_aicc = list(ic = "aicc", stepwise = FALSE, parallel = TRUE),
    stepwise_false_aic  = list(ic = "aic", stepwise = FALSE, parallel = TRUE),
    stepwise_false_bic  = list(ic = "bic", stepwise = FALSE, parallel = TRUE)
  )
  config_keys <- names(configs)

  # Initialize storage
  pp_pvalues <- numeric(n_sim)
  arima_order <- setNames(vector("list", length(config_keys)), config_keys)
  arima_rsme  <- setNames(vector("list", length(config_keys)), config_keys)
  for (key in config_keys) {
    arima_order[[key]] <- character(n_sim)
    arima_rsme[[key]] <- numeric(n_sim)
  }
```

```r
  # Simulation loop
  for (i in 1:n_sim) {
    # Train-test split
    x <- arima.sim(model = list(ar = phi), n = series_length, sd = sigma)
    split_index <- floor(train_test_split * series_length)
    x_train <- x[1:split_index]
    x_test <- x[(split_index + 1):length(x)]

    # pp.test()
    pp_pvalues[i] <- suppressWarnings(pp.test(x_train)$p.value)

    # auto.arima()
    for (key in config_keys) {
      config <- configs[[key]]
      fit <- auto.arima(
        x_train, ic = config$ic, stepwise = config$stepwise,
        stationary = FALSE, seasonal = TRUE, parallel = config$parallel
      )

      # Order
      arima_order[[key]][i] <- get_order(fit)

      # RSME
      pred <- forecast(fit, h = length(x_test))$mean
      arima_rsme[[key]][i] <- sqrt(mean((x_test - pred)^2))
    }
  }

  return(list(
    pp_test_pvalues = pp_pvalues,
    arima_orders = arima_order,
    arima_rmse = arima_rsme
  ))
}

sml_sample <- ar1_simulation_experiment(1000,50)
med_sample <- ar1_simulation_experiment(1000,100)
lg_sample <- ar1_simulation_experiment(1000,250)
saveRDS(sml_sample, file = "ar1_sim_sml_sample.rds")
saveRDS(med_sample, file = "ar1_sim_med_sample.rds")
saveRDS(lg_sample, file = "ar1_sim_lg_sample.rds")
```

```r
summarize_results <- function(sim_result) {
  # Config
  true_order <- "ARIMA(1,0,0)"

  # Extract
  order <- sim_result$arima_orders
  rmse <- sim_result$arima_rmse
  pval <- sim_result$pp_test_pvalues

  # Set lists
  criterion <- c("AICc", "AIC", "BIC", "AICc", "AIC", "BIC")
  search_strat <- c("Stepwise", "Stepwise", "Stepwise", "Exhaustive", "Exhaustive", "Exhaustive")
```

```
  accuracy <- c(mean(order[["stepwise_true_aicc"]] == true_order),
                mean(order[["stepwise_true_aic"]] == true_order),
                mean(order[["stepwise_true_bic"]] == true_order),
                mean(order[["stepwise_false_aicc"]] == true_order),
                mean(order[["stepwise_false_aic"]] == true_order),
                mean(order[["stepwise_false_bic"]] == true_order))
  avg_rmse <- c(round(mean(rmse[["stepwise_true_aicc"]]), 3),
                round(mean(rmse[["stepwise_true_aic"]]), 3),
                round(mean(rmse[["stepwise_true_bic"]]), 3),
                round(mean(rmse[["stepwise_false_aicc"]]), 3),
                round(mean(rmse[["stepwise_false_aic"]]), 3),
                round(mean(rmse[["stepwise_false_bic"]]), 3))

  # Data frame for auto.arima summary
  auto_arima_summary <- data.frame(
    Criterion = criterion,
    Search_Strat = search_strat,
    Accuracy = accuracy,
    RMSE = avg_rmse,
    stringsAsFactors = FALSE
  )

  # Data frame for pp.test summary
  pp_summary <- data.frame(
    Accuracy = round(mean(pval < 0.05), 3)
  )

  return(list(auto_arima = auto_arima_summary, pp_test = pp_summary))
}

sml_sample <- readRDS("ar1_sim_sml_sample.rds")
med_sample <- readRDS("ar1_sim_med_sample.rds")
lg_sample <- readRDS("ar1_sim_lg_sample.rds")

sml_summary <- summarize_results(sml_sample)
med_summary <- summarize_results(med_sample)
lg_summary <- summarize_results(lg_sample)
```

Part 5

Generative AI tools were used solely for minor clarity improvements and resolving code syntax or type-related issues. All research, design decisions, and written content are original, and have been carefully reviewed and verified.

A lot of the theory is from [Marriott, 2025], and we used [R Core Team, 2025] as an example dataset.

References

Peter J Brockwell and Richard A Davis. Stationary processes. In *Introduction to time series and forecasting*, pages 39–71. Springer, 2016.

Paul Marriott. Stat 443: Forecasting, spring 2025. Distributed on Waterloo LEARN, 2025. Course notes, University of Waterloo.

R Core Team. USAccDeaths dataset. https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/USAccDeaths.html, 2025. Available in base R's datasets package.