Name: Your name here
Due: 2024/11/06

Day 16 - Lab: Regression harmonics, GLS, and state-space models

Introduction

In this assignment, you will simulate a time series using some of the regression techniques discussed in class. Then, you will fit an OLS, GLS, and state-space representation of the time series and compare and contrast the results. For this assignment, assume that we are simulating daily measurements over a 25 week period.

Regression harmonics

Sometimes, it can be difficult to put your finger on exactly what the is causing a seasonal effect. In such case, it is useful to model the seasonal component as a sum of harmonic functions, like `sin` and `cos`. The **harmonic seasonal model** for time series $\{x_t\}$ with frequency f is defined by:

$$x_t = m_t + \sum_{i=1}^{[f/2]} (s_i \sin(2\pi it/f) + c_i \cos(2\pi it/f)) + z_t$$

where $[f/2]$ denotes the integer component of $f/2$ (sometimes called the floor function).

For example, a harmonic seasonal model for daily observations over the course of weeks might look like:

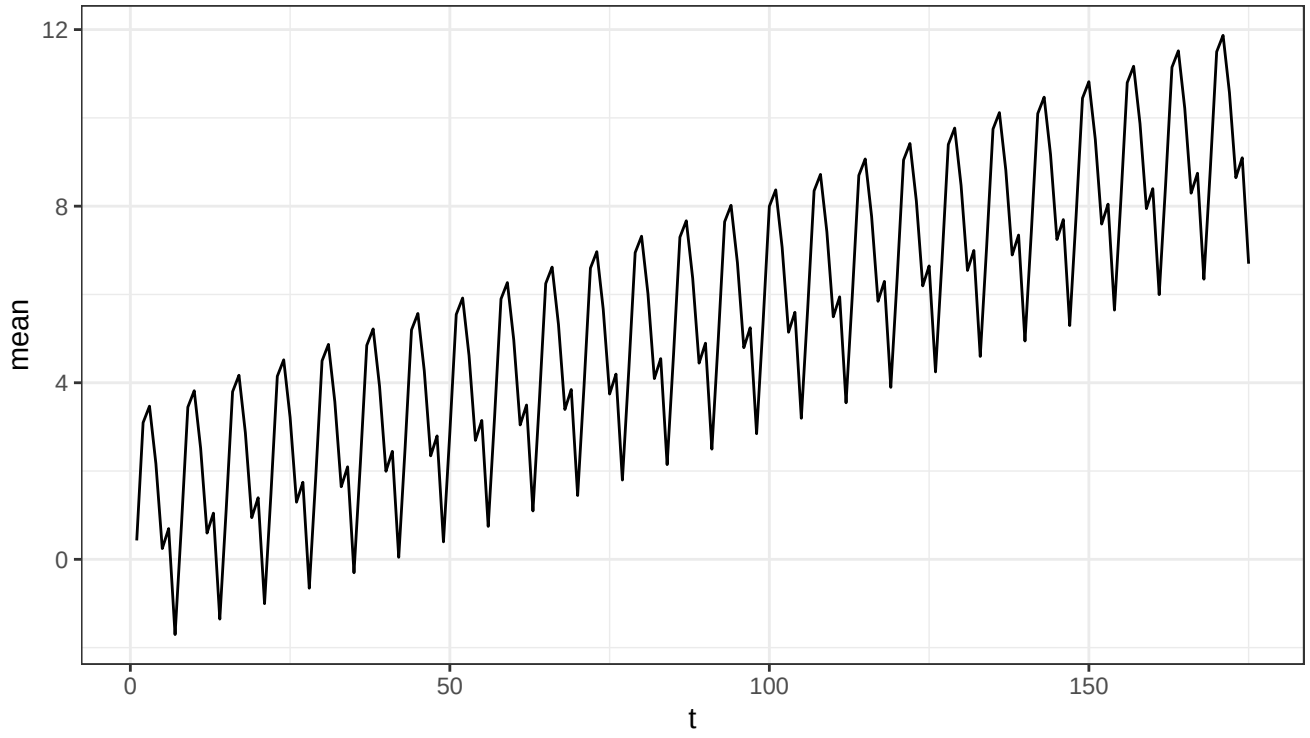
$$x_t = \beta_0 + \beta_1 t + \beta_2 \sin\left(\frac{2\pi t}{7}\right) + \beta_3 \sin\left(\frac{4\pi t}{7}\right) + \beta_4 \sin\left(\frac{6\pi t}{7}\right) + \beta_5 \cos\left(\frac{2\pi t}{7}\right) + \beta_6 \cos\left(\frac{4\pi t}{7}\right) + \beta_7 \cos\left(\frac{6\pi t}{7}\right) + z_t$$

where z_t is the error series. By writing the model in this way, we can use regression to estimate all model parameters. In the following lab, we will simulate a time series using this method, then compare and contrast parameter estimates from OLS, GLS, and state-space models.

Simulating a time series

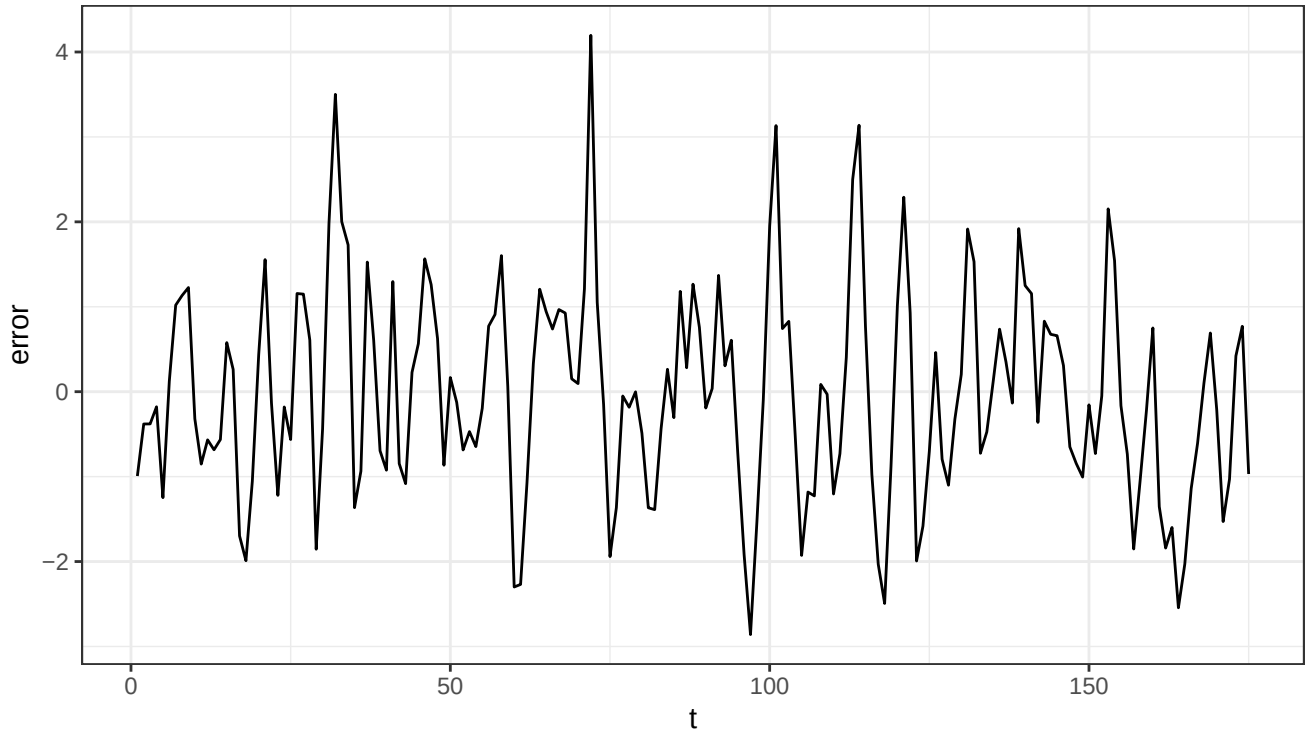
1. [3 pt] Let us start by generating the mean function for our series. Simulate the mean function for daily measurements from a 25-week long series with a linear trend and a harmonic seasonal component. You may choose the values used to generate the mean function, but the resulting function should have clear linear trend and seasonality. Note that your seasonal effects should have 3 harmonic cycles. Plot the mean function.

```
mean_func <- function(seed = NULL, n, freq, beta = rnorm(8)){
  # function to simulate mean of ts using random coefficients
  if(!is.null(seed)) set.seed(seed)
  tmp <- tibble(
    t = 1:n
  ) %>%
    mutate(
      sint = sin(2*pi*t/freq),
      cost = cos(2*pi*t/freq),
      sin2t = sin(2*pi*2*t/freq),
      cos2t = cos(2*pi*2*t/freq),
      sin3t = sin(2*pi*3*t/freq),
      cos3t = cos(2*pi*3*t/freq)
    )

  X <- model.matrix(~ t + sint + cost + sin2t + cos2t + sin3t + cos3t, tmp)
  tmp$mean <- c(X %*% beta)
  out <- list(
    df = tmp,
    mean = tmp$mean,
    beta = beta
  )
  return(out)
}

mean <- mean_func(
  seed = 10292024, n = 7*25, freq = 7,
  beta = c(1, .05, 1, -2, runif(4, -1, 1))
)
mean$df %>%
  ggplot() +
  geom_line(aes(x = t, y = mean))+
  theme_bw()
```



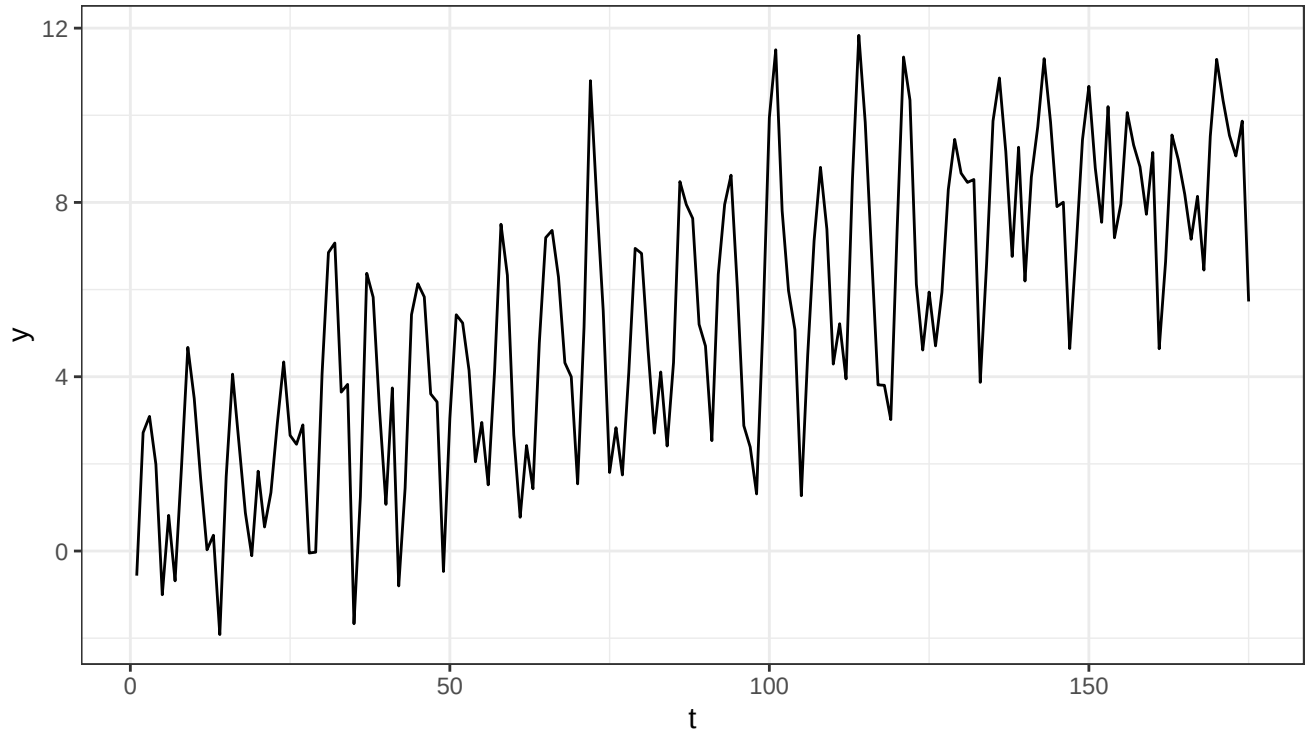
2. [2 pt] Simulate a (stationary) AR(2) process to add to the mean function created in question 1. You may choose the values of the parameters α_1, α_2, and σ, but ensure that the simulated error series is stationary, zero-mean, and exhibits meaningful serial autocorrelation (i.e. do not set the α's to be super small). Plot the error series.

```
set.seed(10292024)
error <- w <- rnorm(7*25)
for(t in 3:(7*25)) error[t] <- .7*error[t-1] - .3*error[t-2] + w[t]
mean$df$error <- error

mean$df %>%
  ggplot() +
  geom_line(aes(x = t, y = error))+
  theme_bw()
```



3. [2 pt] Combine the mean function with the error series to create a complete time series and plot the result.

```
ts_df <- mean$df %>%
  mutate(y = mean + error)

ts_df %>%
  ggplot() +
  geom_line(aes(x = t, y = y)) +
  theme_bw()
```



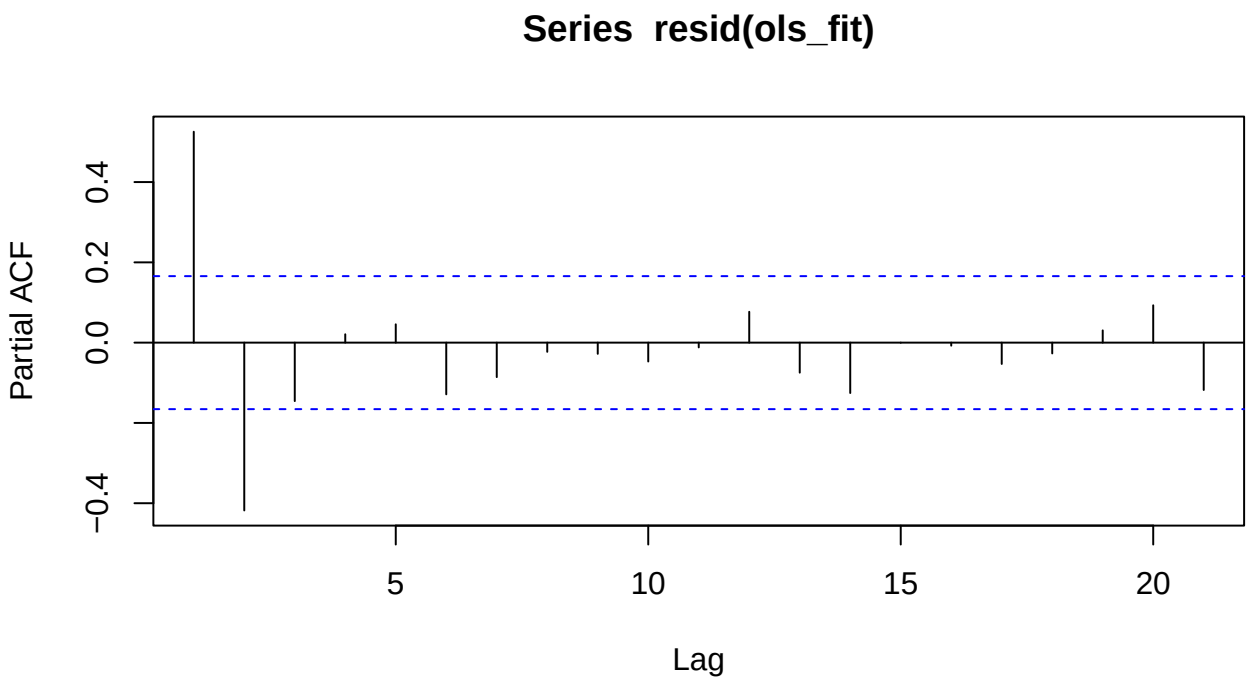
4. [1 pt] Finally, create two data sets: `obs_df` and `forecast_df`, comprised of the first 140 and last 35 time points, respectively. Henceforth, the `obs_df` data set will represent the data that we observe, and the `forecast_df` data set will represent unobserved values that we hope to forecast.

```
obs_df <- ts_df %>% filter(t <= 140)
forecast_df <- ts_df %>% filter(t >140)
```

OLS

5. [2 pt] Fit an ordinary least squares regression model to the `obs_df` time series, including `t` and all 6 harmonic terms, and create an PACF plot of the residuals. Comment on what you see.

```
ols_fit <- lm(y ~ t + sint + sin2t + sin3t + cost + cos2t + cos3t, obs_df)
pacf(resid(ols_fit))
```



> There is evidence of an AR(2) process in the residual error series - which makes sense, because that is what we simulated!

Confidence vs prediction intervals

When forecasting future values, we make the distinction between estimating the *mean response* and predicting a future *individual* response; the former is called a confidence interval and the latter is called a prediction interval. The estimated value remains the same between the two intervals, but the standard error of the value changes (which affects the width of the confidence interval).

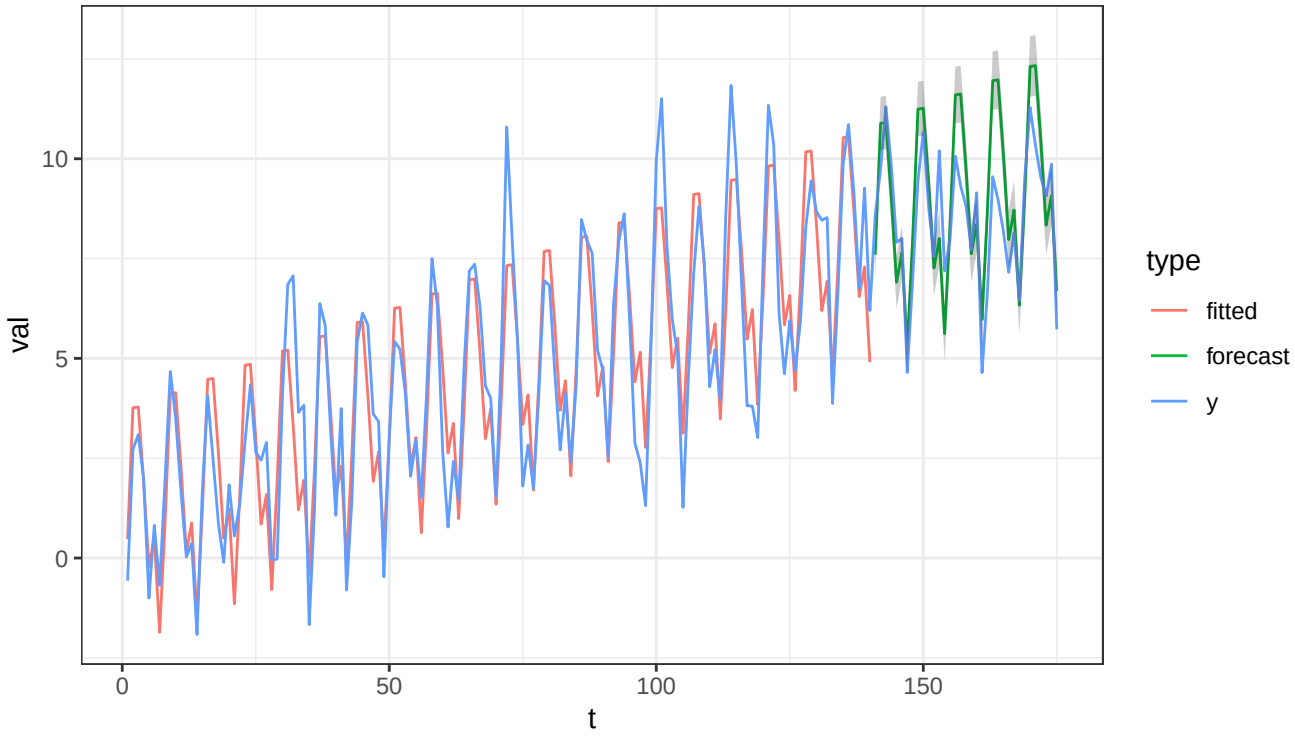
As an example, consider the `AirPassengers` data. When forecasting one time point ahead, a confidence interval would provide a range of plausible values for the average number of air passengers on all planes one time point ahead, while a prediction interval would provide a range of plausible values for the number of air passengers on a single plane. There is much more uncertainty in the latter than the former.

The formulas to calculate confidence intervals and predictions intervals for regression models are quite complicated. Fortunately, the `predict` function in R will do it for you by specifying the `interval` type in the call to predict.

6. [2 pt] Forecast the values of the series in the `forecast_df` data frame, and plot the observed series, fitted series, and forecasted series on a single plot, including 95% **confidence** intervals for the forecast. (see `?predict.lm` for examples)

```
pred <- predict(ols_fit, newdata = forecast_df, interval = "confidence")

# plot
ts_df %>%
  mutate(
    fitted = c(fitted(ols_fit), rep(NA, nrow(forecast_df))),
    forecast = c(rep(NA, nrow(obs_df)), pred[,1])
  ) %>%
  dplyr::select(t, y, fitted, forecast) %>%
  pivot_longer(y:forecast, names_to = "type", values_to = "val") %>%
  ggplot() +
  geom_line(aes(x = t, y = val, col = type)) +
  geom_ribbon(
    data = tibble(
      t = forecast_df$t,
      lwr = pred[,2],
      upr = pred[,3]
    ),
    aes(x = t, ymin = lwr, ymax = upr), alpha = .25
  ) +
  theme_bw()
```

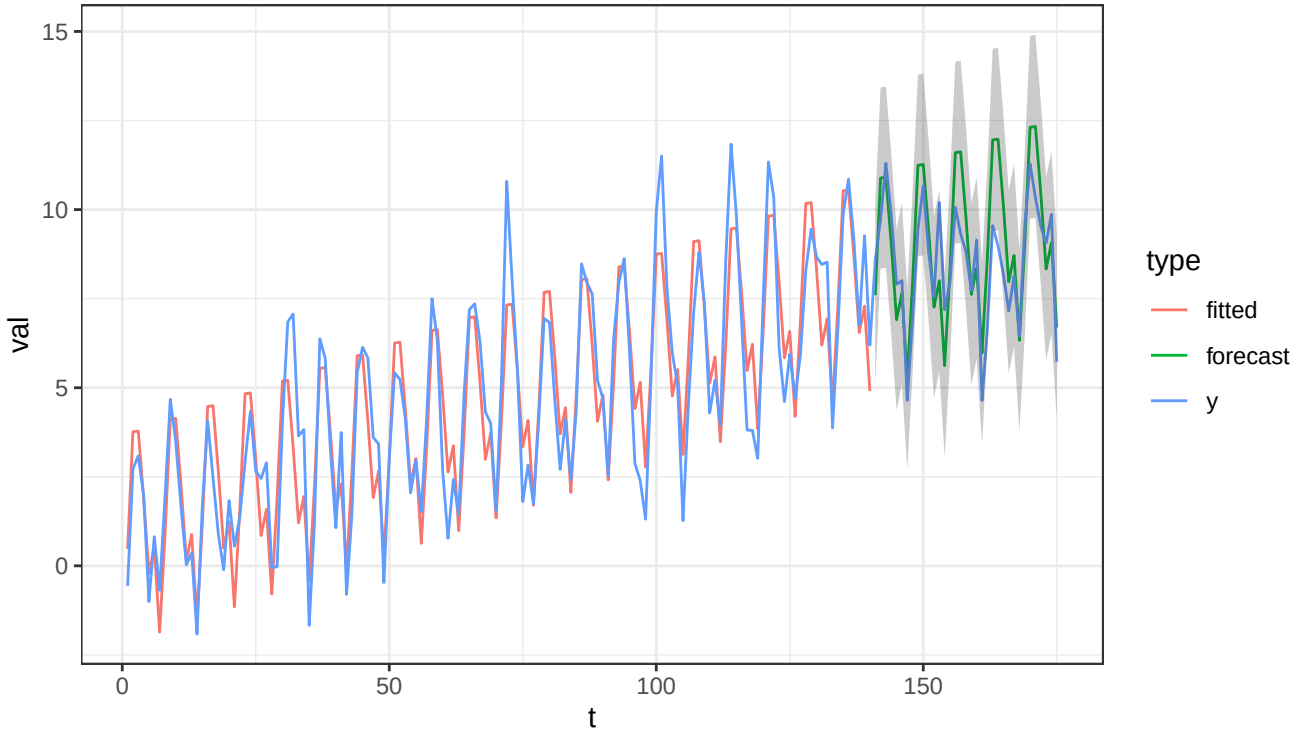
```
Warning: Removed 175 rows containing missing values or values outside the scale range
(`geom_line()`).
```



7. [2 pt] Do the same thing again, but instead including a **prediction** interval. Comment on which intervals are wider: the confidence or the prediction.

```
pred <- predict(ols_fit, newdata = forecast_df, interval = "prediction")

# plot
ts_df %>%
  mutate(
    fitted = c(fitted(ols_fit), rep(NA, nrow(forecast_df))),
    forecast = c(rep(NA, nrow(obs_df)), pred[,1])
  ) %>%
  dplyr::select(t, y, fitted, forecast) %>%
  pivot_longer(y:forecast, names_to = "type", values_to = "val") %>%
  ggplot() +
  geom_line(aes(x = t, y = val, col = type)) +
  geom_ribbon(
    data = tibble(
      t = forecast_df$t,
      lwr = pred[,2],
      upr = pred[,3]
    ),
    aes(x = t, ymin = lwr, ymax = upr), alpha = .25
  ) +
  theme_bw()
```

```
Warning: Removed 175 rows containing missing values or values outside the scale range
(`geom_line()`).
```



The prediction intervals are much wider!

GLS

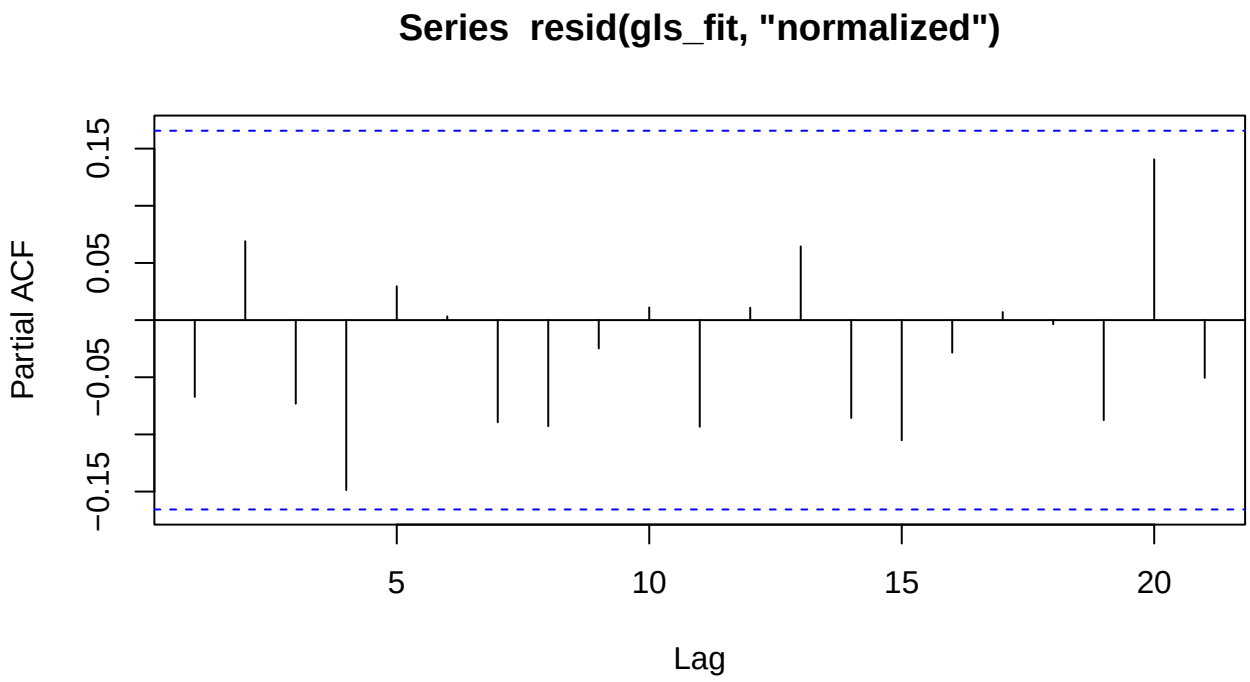
8. [2 pt] Fit a GLS model with an AR(2) correlation structure to the `obs_df` time series and create a pacf plot of the normalized residuals. What are the estimated values of α_1 and α_2? Does the GLS model solve the issues with autocorrelation in the residuals?

```
gls_fit <- gls(
  y ~ t + sint + sin2t + sin3t + cost + cos2t + cos3t,
  correlation = corARMA(p = 2),
  obs_df
)
summary(gls_fit)
```

```
Generalized least squares fit by REML
  Model: y ~ t + sint + sin2t + sin3t + cost + cos2t + cos3t
  Data: obs_df
       AIC     BIC    logLik
  427.4901 459.201 -202.7451

Correlation Structure: ARMA(2,0)
 Formula: ~1
 Parameter estimate(s):
      Phi1       Phi2
 0.7529017 -0.4175322

Coefficients:
                 Value  Std.Error    t-value p-value
(Intercept)  1.0071120 0.24542066   4.103615   1e-04
t            0.0506610 0.00302207  16.763683   0e+00
sint         1.4155960 0.23954130   5.909611   0e+00
sin2t       -0.8744745 0.09473622  -9.230625   0e+00
sin3t       -0.3630460 0.05594810  -6.488978   0e+00
cost        -2.0954303 0.23851517  -8.785313   0e+00
cos2t       -0.5484726 0.09489234  -5.779947   0e+00
cos3t       -0.5796086 0.05647311 -10.263444   0e+00

 Correlation:
      (Intr) t      sint   sin2t  sin3t  cost   cos2t
t     -0.868
sint  -0.031  0.033
sin2t -0.012  0.009 -0.005
sin3t -0.005  0.004  0.000  0.002
cost   0.010 -0.016 -0.005 -0.010 -0.004
cos2t  0.007 -0.012 -0.004 -0.007 -0.003 -0.008
cos3t  0.015 -0.017 -0.006 -0.011 -0.004  0.002  0.013
```

```
Standardized residuals:
        Min          Q1         Med          Q3         Max
-2.23300812 -0.64339440 -0.09924543  0.65822488  2.98414034

Residual standard error: 1.243035
Degrees of freedom: 140 total; 132 residual
```

```
pacf(resid(gls_fit, "normalized"))
```



$\hat{\alpha}_1 = .75$ and $\hat{\alpha}_2 = -.41$, which are very similar to the values used to generate the series. Based on the PACF plot, there does not appear to be autocorrelation left in the residual error series!

No more questions on GLS in the lab - tune in for more in the homework. :)

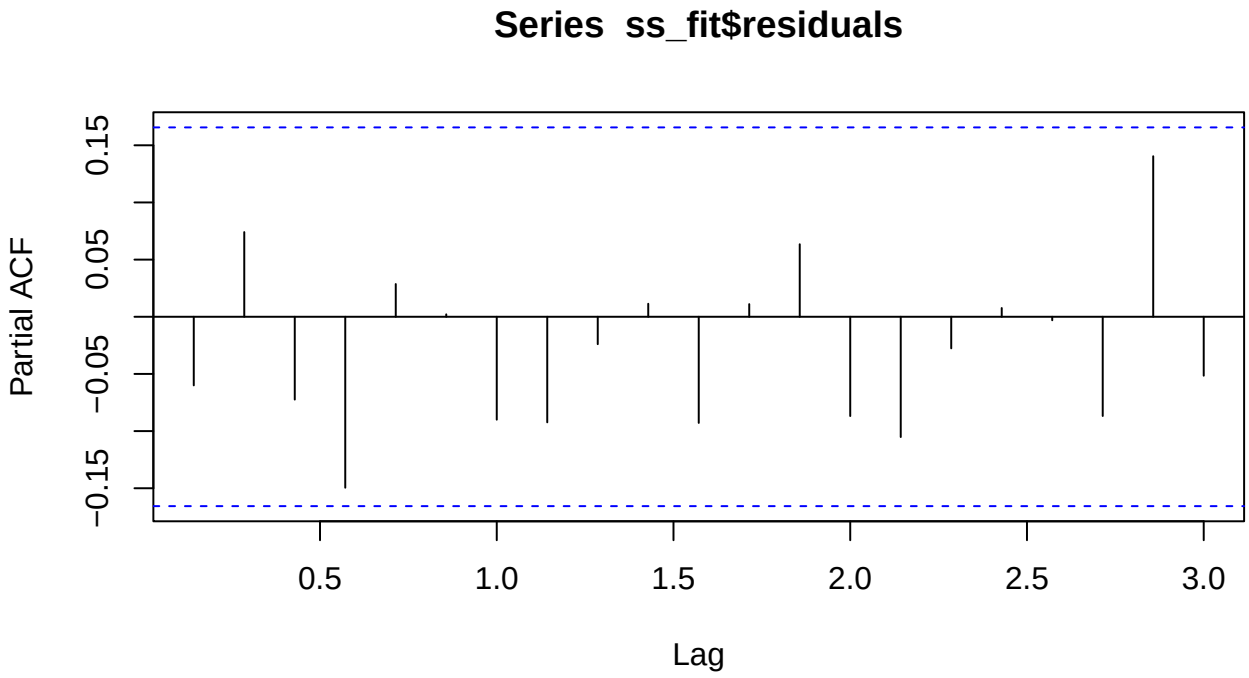
State-space

9. [2 pt] Fit a state-space model with an AR(2) correlation structure to `obs_df` time series and create a pacf plot of the residuals. What are the estimated values of α_1 and α_2? Does the SS model solve the issues with autocorrelation in the residuals?

```
# a few things
sim_ts <- ts(
  obs_df$y,
  freq = 7
)
X <- model.matrix(~ t + sint + sin2t + sin3t + cost + cos2t + cos3t, obs_df)

ss_fit <- arima(
  sim_ts,
  order = c(2, 0, 0),
  xreg = X,
  include.mean = F
)
ss_fit
```

```
Call:
arima(x = sim_ts, order = c(2, 0, 0), xreg = X, include.mean = F)

Coefficients:
         ar1      ar2  (Intercept)       t    sint    sin2t    sin3t     cost
      0.7456  -0.4182       1.0076  0.0507  1.4149  -0.8746  -0.3631  -2.0953
s.e.  0.0766   0.0772       0.2358  0.0029  0.2321   0.0925   0.0545   0.2308
        cos2t    cos3t
      -0.5483  -0.5795
s.e.   0.0927   0.0550

sigma^2 estimated as 0.8635:  log likelihood = -188.73,  aic = 399.46
```

```
pacf(ss_fit$residuals)
```



$\hat{\alpha}_1 = .745$ and $\hat{\alpha}_2 = -.418$, which are very similar to the values used to generate the series (and the GLS values, but not the same). Based on the PACF plot, there does not appear to be autocorrelation left in the residual error series!

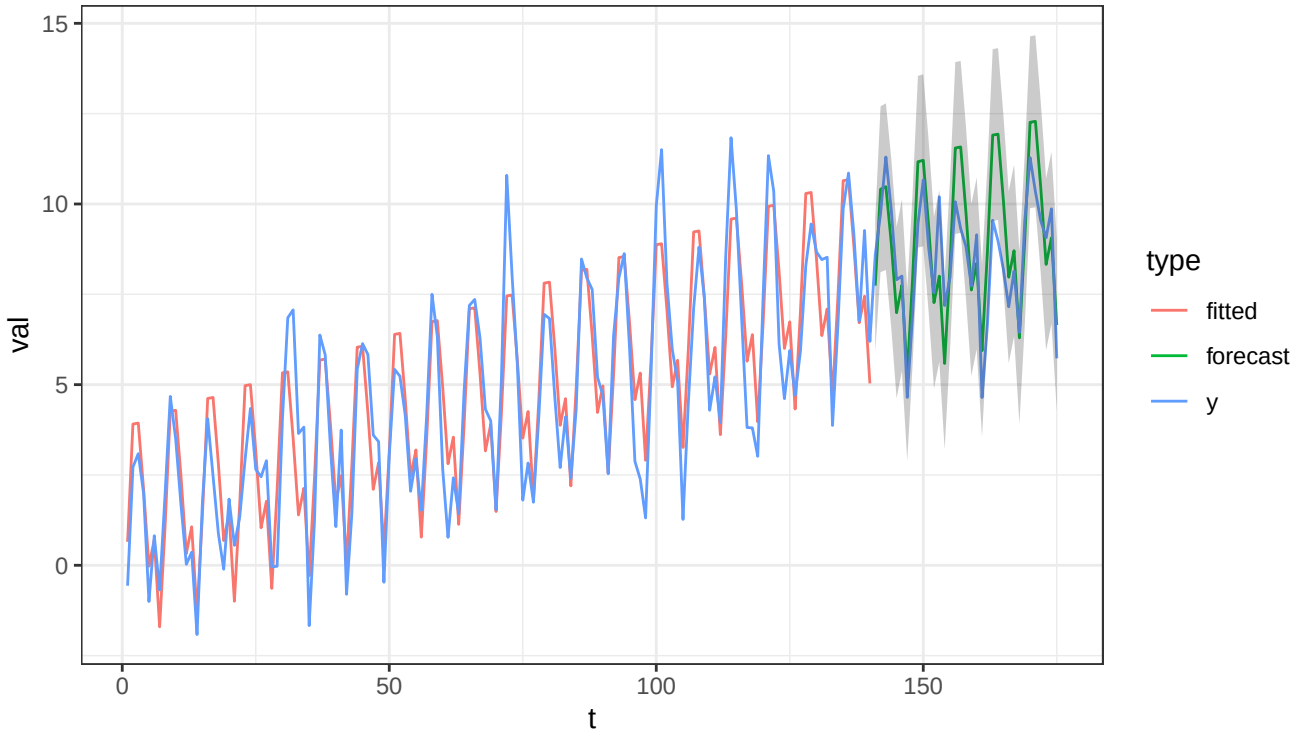
One of the main advantages of the state-space representation of the GLS model is that we are able to obtain standard errors for predictions. Sadly, the `predict` function cannot generate confidence intervals and prediction intervals for forecasts from an `arima`, as there is no notion of confidence vs prediction with the state-space approach. Instead, we can generate an approximate 95% prediction interval by using the standard errors and t-distribution.

10. [4 pt] Forecast the last 35 observations and construct approximate 95% prediction intervals for the forecasted values. Plot the observed, fitted, and forecasted values on a single plot. To create the intervals, you should use a multiplier obtained from `qt(.975, 140-10)` (the degrees of freedom are determined by the sample size minus the number of estimated parameters).

```
# fitted
fitted <- predict(
  ss_fit,
  newxreg = model.matrix(~ t + sint + sin2t + sin3t + cost + cos2t + cos3t, obs_df)
)

# pred
pred <- predict(
  ss_fit,
  n.ahead = 35,
```

```
  newxreg = model.matrix(~ t + sint + sin2t + sin3t + cost + cos2t + cos3t, forecast_df)
)

# plot
ts_df %>%
  mutate(
    fitted = c(fitted$pred, rep(NA, nrow(forecast_df))),
    forecast = c(rep(NA, nrow(obs_df)), pred$pred)
  ) %>%
  dplyr::select(t, y, fitted, forecast) %>%
  pivot_longer(y:forecast, names_to = "type", values_to = "val") %>%
  ggplot() +
  geom_line(aes(x = t, y = val, col = type)) +
  geom_ribbon(
    data = tibble(
      t = forecast_df$t,
      lwr = pred$pred - qt(.975, 130)*pred$se,
      upr = pred$pred + qt(.975, 130)*pred$se
    ),
    aes(x = t, ymin = lwr, ymax = upr), alpha = .25
  ) +
  theme_bw()
```

```
Warning: Removed 175 rows containing missing values or values outside the scale range
(`geom_line()`).
```



11. [4 pt] The state-space representation of the model does not provide t-statistics and p-values by default. However, we can calculate approximate tests using the same t-distribution with 130 degrees of freedom. Create a matrix that returns the estimated regression coefficients, standard errors, t-statistics, and two-sided p-values for the estimates from the state-space model.

```
tibble(
  param = names(ss_fit$coef[-c(1:2)]),
  est = ss_fit$coef[-c(1:2)],
  se = sqrt(diag(ss_fit$var.coef)[-c(1:2)])
) %>%
  mutate(
    t = est/se,
    p_val = 2*pt(-abs(t), 130)
  )
```

```
# A tibble: 8 x 5
  param           est      se      t    p_val
  <chr>         <dbl>   <dbl>  <dbl>    <dbl>
1 (Intercept)  1.01   0.236     4.27 3.68e- 5
2 t            0.0507 0.00291  17.4  7.89e-36
3 sint         1.41   0.232     6.10 1.15e- 8
4 sin2t       -0.875  0.0925   -9.46 1.82e-16
5 sin3t       -0.363  0.0545   -6.66 6.95e-10
6 cost        -2.10   0.231    -9.08 1.53e-15
7 cos2t       -0.548  0.0927   -5.92 2.72e- 8
```

```
8 cos3t       -0.580  0.0550  -10.5  3.90e-19
```