Aggregation fallacy worksheet

Joachim Vandekerckhove

Step 1: Simulate Learning Curves

- We generate learning curves for 10 participants, each with a random change point.

```r
# Simulate 10 participants' learning curves
n_participants <- 10
time_points <- 1:100
change_points <- sample(1:100, n_participants,
                        replace = TRUE)
learning_curves <- sapply(change_points,
    function(cp) ifelse(time_points < cp, 0, 1))
```

Step 1: Simulate Learning Curves

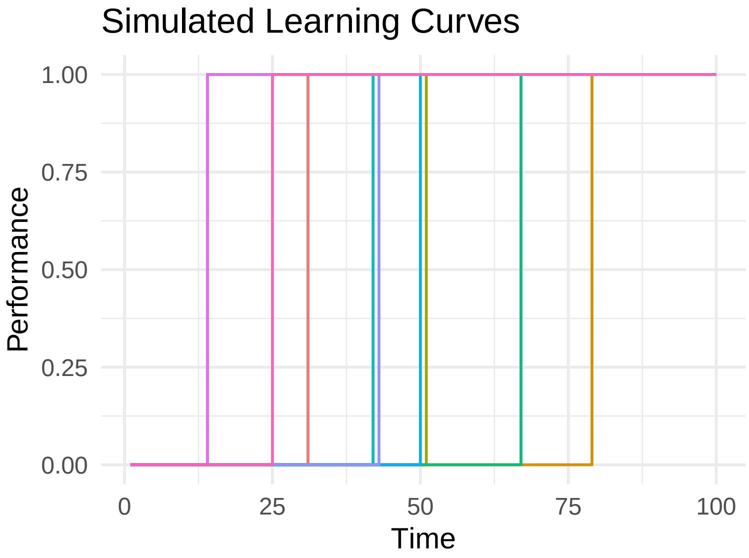
- We generate learning curves for 10 participants, each with a random change point.
- We use a step function to represent the learning curves.

```r
# Simulate 10 participants' learning curves
n_participants <- 10
time_points <- 1:100
change_points <- sample(1:100, n_participants,
                        replace = TRUE)
learning_curves <- sapply(change_points,
    function(cp) ifelse(time_points < cp, 0, 1))
```

```r
# Create a data frame for plotting
learning_data <- data.frame(
  Time = rep(time_points, n_participants),
  Performance = as.vector(learning_curves),
  Participant = rep(1:n_participants, each = 100))

# Plot the simulated learning curves
p1 <- ggplot(learning_data,
             aes(x = Time,
                 y = Performance,
                 group = Participant,
                 color = as.factor(Participant))) +
  geom_step() +
  labs(title = "Simulated Learning Curves",
       x = "Time", y = "Performance") +
  theme_minimal() +
  theme(legend.position = "none")
```

```r
print(p1)
```


Simulated Learning Curves
Performance
1.00
0.75
0.50
0.25
0.00
0
25
50
75
100
Time

Step 2: Estimate Smooth Curve

To fit a smooth curve to the average performance, we'll use a logistic regression model.

- We calculate the summed performance at each time point across all participants.

```r
# Calculate the summed performance at each time point
summed_performance <- rowSums(learning_curves)
```

Step 2: Estimate Smooth Curve

To fit a smooth curve to the average performance, we'll use a logistic regression model.

- We calculate the summed performance at each time point across all participants.
- We fit a logistic regression model using JAGS.

```r
# Calculate the summed performance at each time point
summed_performance <- rowSums(learning_curves)
```

```
# Define the JAGS model for the smooth curve
curve_model <- "
model {
  for (i in 1:N) {
    y[i] ~ dbin(p[i], P)
    logit(p[i]) <- alpha + beta * (time[i] - midpoint)
  }
  alpha ~ dnorm(0, 0.01)
  beta ~ dnorm(0, 0.01)
  midpoint ~ dunif(1, 100)
}
"
```

```r
# Prepare data for JAGS
data_list <- list(
  y    = summed_performance,
  time = time_points,
  P    = n_participants,
  N    = length(time_points)
)

# Initial values
inits <- function() {
  list(alpha = 0, beta = 1, midpoint = 50)
}

# Parameters to monitor
params <- c("alpha", "beta", "midpoint")
```

```r
# Run the JAGS model
jags_curve <- jags.model(
  textConnection(curve_model), data = data_list,
  inits = inits, n.chains = 8, n.adapt = 1250)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 3
##    Total graph size: 609
##
## Initializing model
```

```r
update(jags_curve, n.iter = 1250)
samples_curve <- coda.samples(
  jags_curve, variable.names = params, n.iter = 2500)

# Extract the posterior means for the parameters
posterior_means <- summary(samples_curve)$
  statistics[, "Mean"]
alpha_hat <- posterior_means["alpha"]
beta_hat <- posterior_means["beta"]
midpoint_hat <- posterior_means["midpoint"]
```

Diagnostic check.

```r
print(gelman.diag(samples_curve))
```

```
## Potential scale reduction factors:
##
##          Point est. Upper C.I.
## alpha          1.45       2.02
## beta           1.00       1.00
## midpoint       1.45       2.02
##
## Multivariate psrf
##
## 1.47
```

Print some summary statistics.

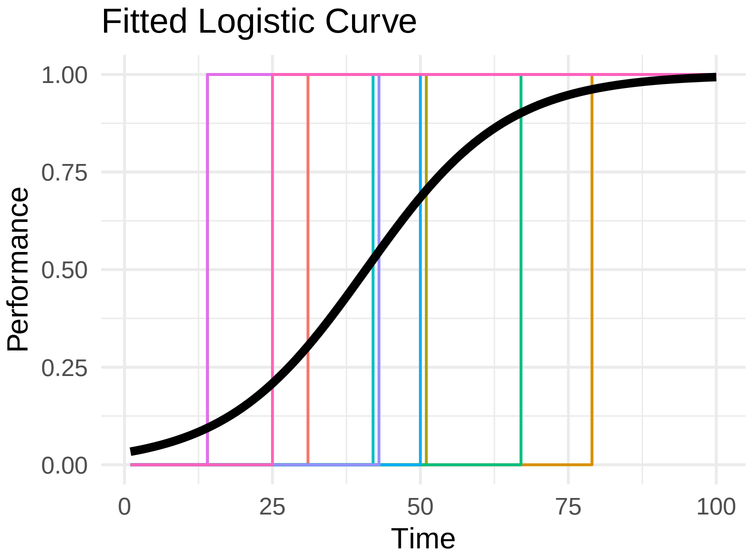
```r
summary(samples_curve)[1]$
  statistics[, c("Mean", "SD", "Time-series SE")] %>%
  print()
```

```
##                 Mean          SD Time-series SE
## alpha     0.77049111  2.416956069   0.4604354447
## beta      0.08446575  0.005075513   0.0001122589
## midpoint 49.90721205 28.592735240   5.4656869730
```

```r
# Calculate the fitted curve
fitted_curve <- 1 / (1 + exp(-(alpha_hat + beta_hat *
                    (time_points - midpoint_hat))))
curve_data <- data.frame(Time = time_points,
                         Performance = fitted_curve)

# Add the fitted curve to the existing plot
p2 <- p1 + geom_line(data = curve_data,
               aes(x = Time, y = Performance),
               inherit.aes = FALSE, color = "black",
               linewidth = 1.5) +
  labs(title = "Fitted Logistic Curve")
```

```r
print(p2)
```


Fitted Logistic Curve
1.00
0.75
0.50
0.25
0.00
Performance
0
25
50
75
100
Time

Step 3: Hierarchical Model

- We define a hierarchical model in JAGS to estimate each participant's change point.

Step 3: Hierarchical Model

- We define a hierarchical model in JAGS to estimate each participant's change point.
- The change point for each participant is modeled as a uniform distribution between 1 and 100.

```
hierarchical_model <- "
model {
  for (j in 1:P) {
    for (i in 1:N) {
      y[i, j] ~ dbern(p[i, j])
      p[i, j] <- ifelse(time[i] < change_point[j], 0, 1)
    }
    change_point[j] ~ dnorm(mu, tau)T(0,100)
  }
  mu ~ dnorm(50, 20)T(0,100)
  tau ~ dnorm(20, 10)T(0,)
  sigma <- pow(tau, -0.5)
}
"
```

```r
# Prepare data for JAGS
data_list_hrcl <- list(
  y    = learning_curves,
  time = time_points,
  P    = n_participants,
  N    = length(time_points)
)

# Initial values
inits_hrcl <- function() {
  list(change_point = change_points)
}

# Parameters to monitor
params_hrcl <- c("mu", "sigma")
```

```r
# Run the JAGS model
jags_hrcl <- jags.model(
  textConnection(hierarchical_model),
  data = data_list_hrcl,
  inits = inits_hrcl,
  n.chains = 3, n.adapt = 1000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1000
##    Unobserved stochastic nodes: 12
##    Total graph size: 3123
##
## Initializing model
```

```r
update(jags_hrcl, n.iter = 1000)
samples_hrcl <- coda.samples(
  jags_hrcl,
  variable.names = params_hrcl,
  n.iter = 5000)
```

Diagnostic check.

```r
print(gelman.diag(samples_hrcl))
```

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## mu           1.0        1.0
## sigma        1.1        1.1
##
## Multivariate psrf
##
## 1
```

```r
# Extract the posterior means for the parameters
posterior_means_hrcl <- summary(samples_hrcl)$
  statistics[, "Mean"]
mu_hat_hrcl    <- posterior_means_hrcl["mu"]
sigma_hat_hrcl <- posterior_means_hrcl["sigma"]

summary(samples_hrcl)[1]$
  statistics[, c("Mean", "SD", "Time-series SE")] %>%
  print()
```

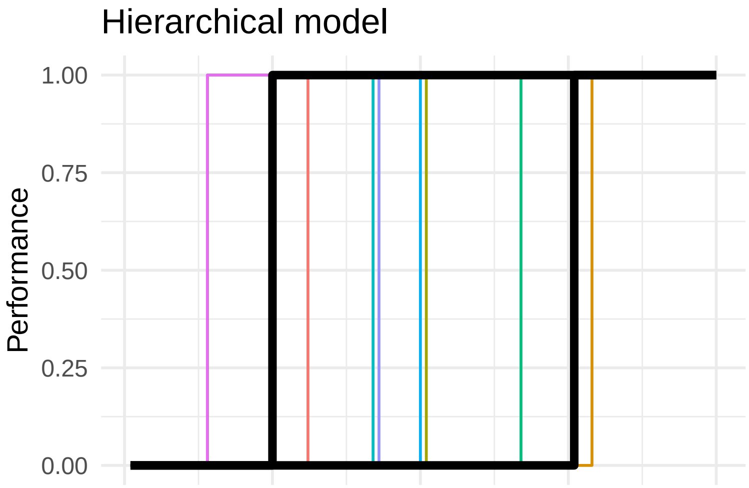
```
##             Mean        SD Time-series SE
## mu    49.99197  0.224002    0.002298043
## sigma 25.88520 21.737210    0.208580970
```

Generate some representative participants from the model parameters.

```r
low  = mu_hat_hrcl - sigma_hat_hrcl
high = mu_hat_hrcl + sigma_hat_hrcl

step_function_data <- data.frame(
  Time = rep(time_points, 2),
  Performance = c(ifelse(time_points < low, 0, 1),
                  ifelse(time_points < high, 0, 1)),
  Function = rep(c("-1SD", "+1SD"),
                 each = length(time_points))
)
```

```r
p1 + geom_step(data = step_function_data,
               aes(x = Time, y = Performance,
                   group = Function), color = "black",
               linewidth = 1.5) +
  labs(title = "Hierarchical model")
```


Hierarchical model
Performance