forecast_example

July 25, 2025

1 HMM Forecasting Walkthrough

1.0.1 This notebook shows the step by step process of generating hmm forecasts along with forecast validation using simulated data.

1.1 Setup

Load hmmTMB and other packages

[15]:
```
library(devtools)
library(ggridges)
set.seed(1)
```

[16]:
```
load_all("../../../hmmTMB")
#library(hmmTMB)
```

```
Loading hmmTMB
```

1.2 Generate Data

Use the true model defined in true_mod.hmm to set the model functions. Set arbitrary fixed effect coefficients.

[17]:
```
# Simulate data -----------------------------------------------------------

# number of time steps
n <- 1000

# Generate training data with random covariates
empty <- data.frame(
  ID = 1,
  pois_obs = rep(NA, n),
  norm_obs = rep(NA, n),
  covariate_1 = rnorm(n, mean = 0, sd = 1),
  covariate_2 = runif(n, min = -1, max = 1),
  covariate_3 = sample(c(-1, 0, 1), n, replace = TRUE)
  #random_effect_id = as.factor(rep(1:2, each = n/2))
)

# create true model
```

```r
hid <- MarkovChain$new(
  data = empty,
  n_states = 2,
  formula = ~covariate_3,
  tpm = matrix(c(0.8, 0.3, 0.2, 0.7), 2, 2)
)
obs <- Observation$new(
  data = empty,
  dists = list(
    pois_obs = "pois",
    norm_obs = "norm"
  ),
  par = list(
    pois_obs = list(rate = c(1, 2)),
    norm_obs = list(mean = c(0, 1), sd = c(1, 1))
  ),
  formula = list(
    pois_obs = list(rate = ~ covariate_1),
    norm_obs = list(mean = ~ covariate_2)
  )
)
true_mod <- HMM$new(
  obs = obs,
  hid = hid
)

# Create a list of random integers for the hidden Markov model coefficients
random_coeff_fe_obs <- replicate(
  length(true_mod$coeff_list()$coeff_fe_obs),
  sample(c(0.9, 1, 1.1, 1.2), 1, replace = TRUE)
)
random_coeff_re_obs <- replicate(
  length(true_mod$coeff_list()$coeff_re_obs),
  sample(c(-0.1, 0, 0.1, 0.2), 1, replace = TRUE)
)
random_log_lambda_obs <- replicate(
  length(true_mod$coeff_list()$log_lambda_obs),
  sample(c(-0.1, 0, 0.1, 0.2), 1, replace = TRUE)
)

# Update the coefficients of the hidden Markov model with the random values
true_mod$obs()$update_coeff_fe(coeff_fe = random_coeff_fe_obs)
true_mod$obs()$update_coeff_re(coeff_re = random_coeff_re_obs)
true_mod$obs()$update_lambda(random_log_lambda_obs)

random_coeff_fe_hid <- replicate(
  length(true_mod$coeff_list()$coeff_fe_hid),
```

```
  sample(c(0.9, 1, 1.1, 1.2), 1, replace = TRUE)
)
random_coeff_re_hid <- replicate(
  length(true_mod$coeff_list()$coeff_re_hid),
  sample(c(-0.1, 0, 0.1, 0.2), 1, replace = TRUE)
)
random_log_lambda_hid <- replicate(
  length(true_mod$coeff_list()$log_lambda_hid),
  sample(c(-0.1, 0, 0.1, 0.2), 1, replace = TRUE)
)

true_mod$hid()$update_coeff_fe(coeff_fe = random_coeff_fe_hid)
true_mod$hid()$update_coeff_re(coeff_re = random_coeff_re_hid)
true_mod$hid()$update_lambda(random_log_lambda_hid)

# View the model parameters
true_mod$coeff_list()
```

$coeff_fe_obs A matrix: 10 × 1 of type dbl

pois_obs.rate.state1.(Intercept)	0.9
pois_obs.rate.state1.covariate_1	1.2
pois_obs.rate.state2.(Intercept)	1.0
pois_obs.rate.state2.covariate_1	1.0
norm_obs.mean.state1.(Intercept)	0.9
norm_obs.mean.state1.covariate_2	1.1
norm_obs.mean.state2.(Intercept)	1.1
norm_obs.mean.state2.covariate_2	0.9
norm_obs.sd.state1.(Intercept)	1.1
norm_obs.sd.state2.(Intercept)	0.9

$log_lambda_obs

$coeff_fe_hid A matrix: 4 × 1 of type dbl

S1>S2.(Intercept)	1.2
S1>S2.covariate_3	1.2
S2>S1.(Intercept)	1.2
S2>S1.covariate_3	1.2

$log_lambda_hid

$log_delta0 A matrix: 1 × 1 of type dbl

ID:1.state1	0

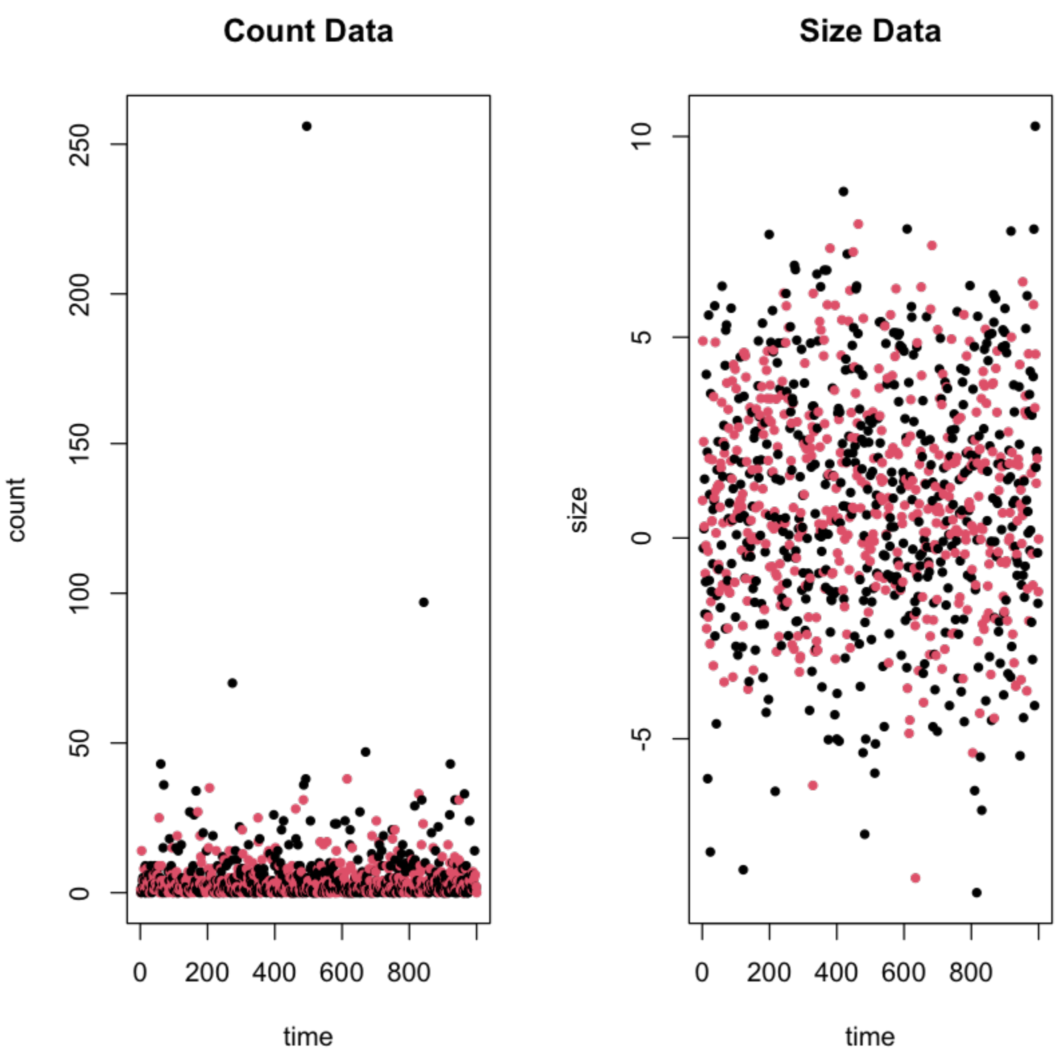
$coeff_re_obs

$coeff_re_hid

[18]:
```
# simulate from true model
training <- true_mod$simulate(n, data = empty)

# update data in true_mod with dat
true_mod$obs()$update_data(training)
```

```r
# plot data
par(mfrow = c(1, 2)) # set outer margins to 0
plot(training$pois_obs, pch = 20, xlab = "time", ylab = "count", col =
  attr(training, "state"), main = "Count Data")
plot(training$norm_obs, pch = 20, xlab = "time", ylab = "size", col =
  attr(training, "state"), main = "Size Data")
par(mfrow = c(1, 1))
```

```
Simulating states… 100%
Simulating pois_obs… 100%
Simulating norm_obs… 100%
```



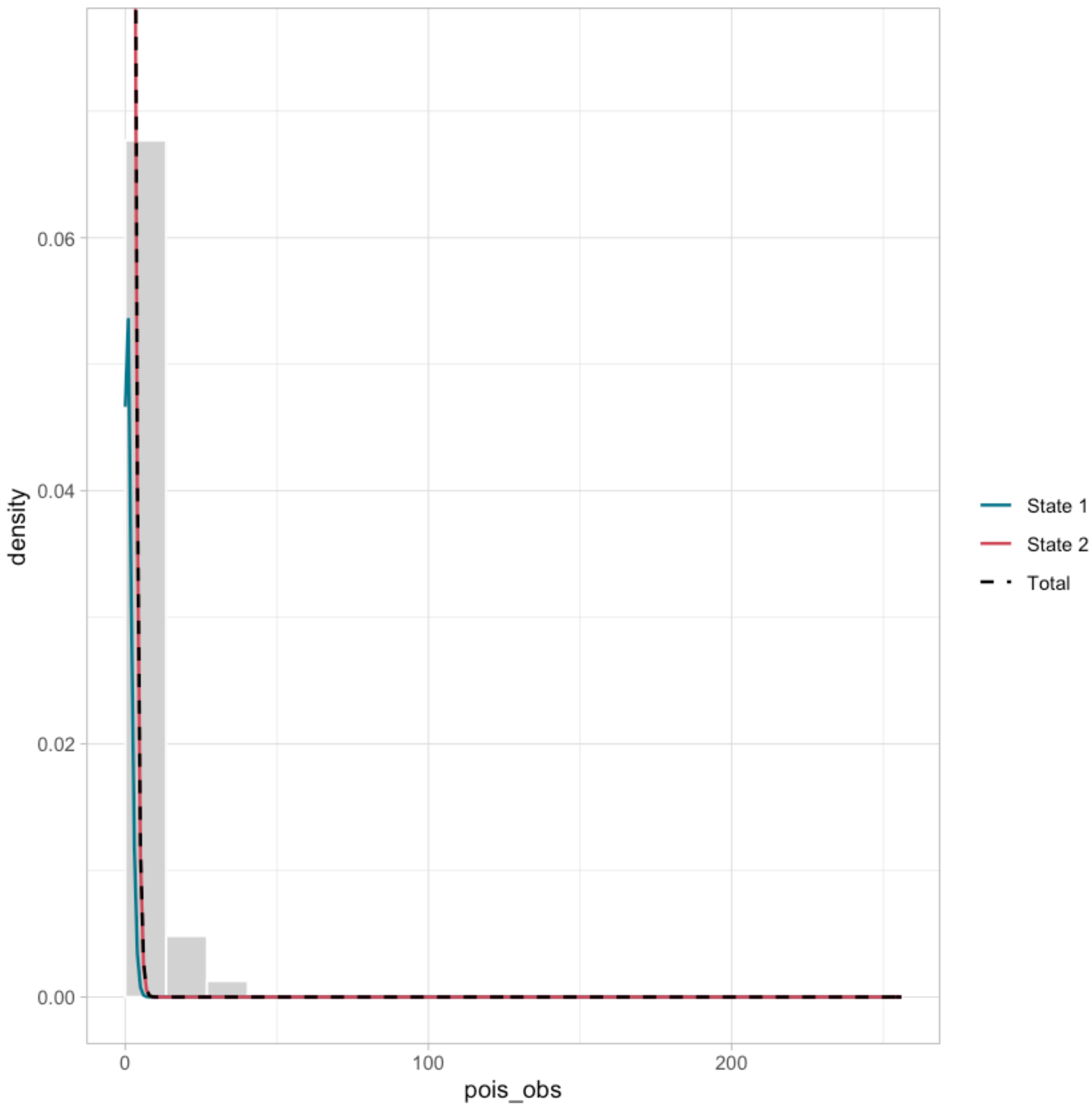
1.3 Fit the model

Fit a model using the model definition and starting parameters defined by mod.hmm

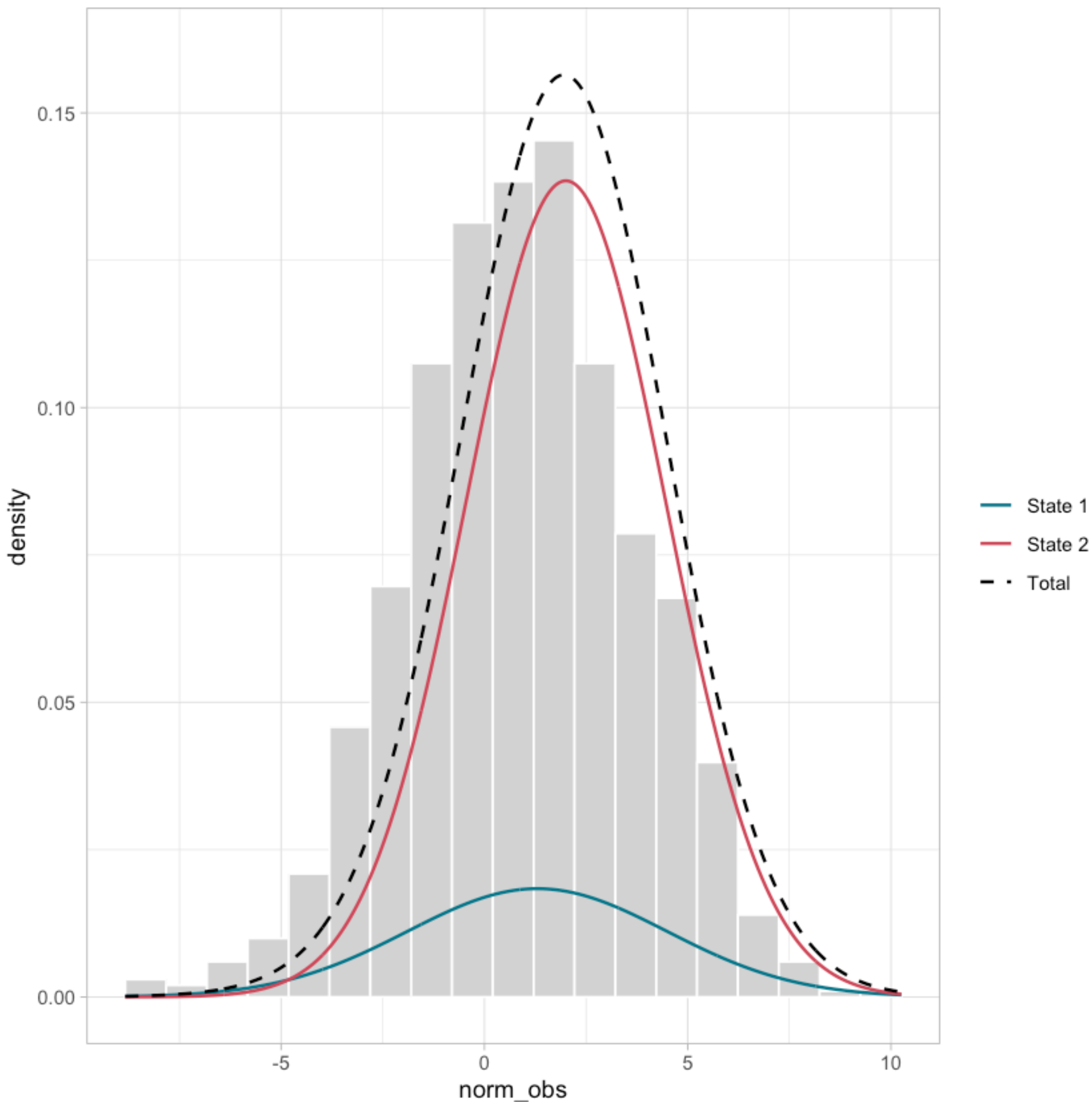
[19]:
```
# Fit model -------------------------------------------------------------

# create model to fit
mod <- HMM$new(hid = hid, obs = obs)

# suggest better starting parameters
ini <- mod$suggest_initial()

# set to new starting parameters (or you could edit the specification file)
mod$obs()$update_par(ini)

# fit model
mod$fit(silent = TRUE)
```

Check that the model has converged and fit properly

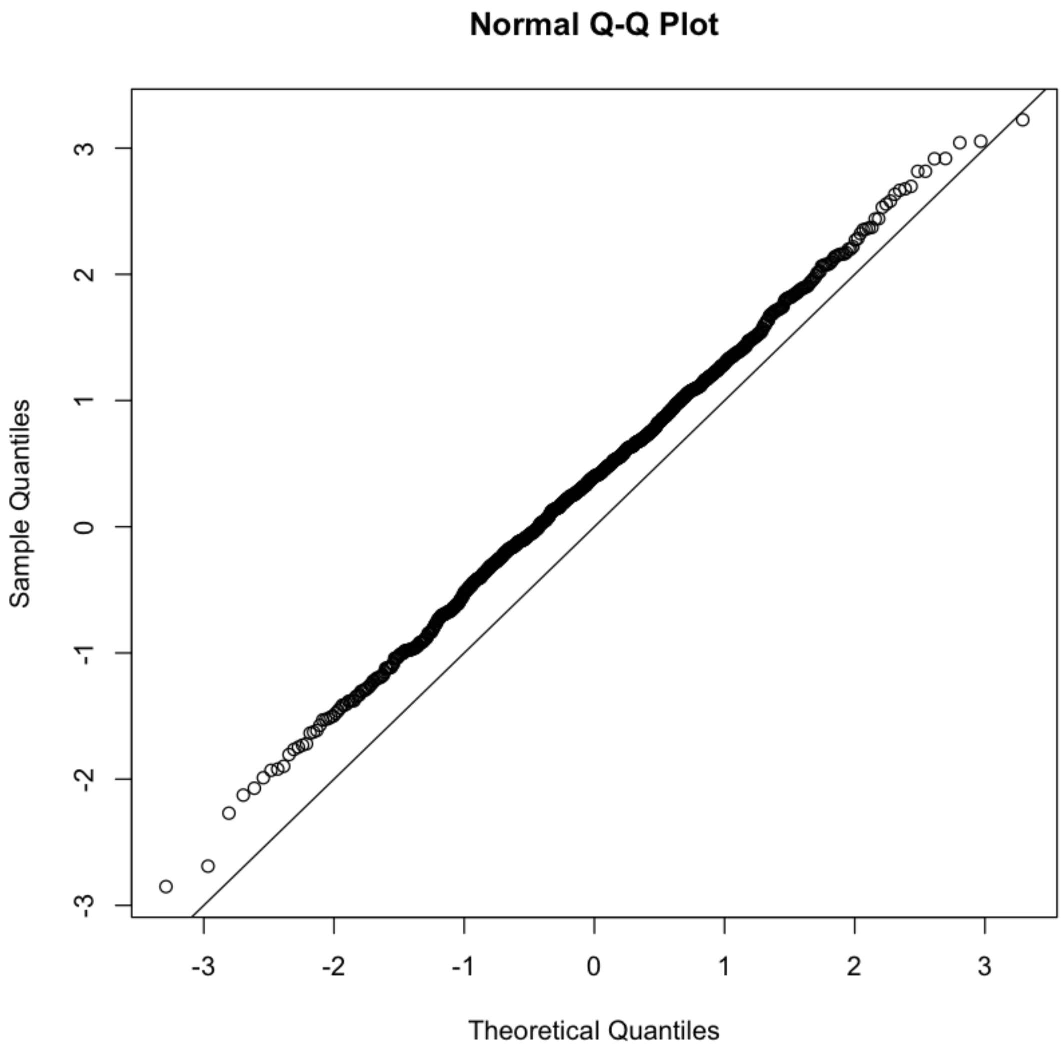
[20]:
```
par(mfrow = c(1, 2))
mod$plot_dist("pois_obs")
mod$plot_dist("norm_obs")
par(mfrow = c(1, 1))

pr <- mod$pseudores()

qqnorm(pr$pois_obs)
abline(0, 1)

qqnorm(pr$norm_obs)
abline(0, 1)
```

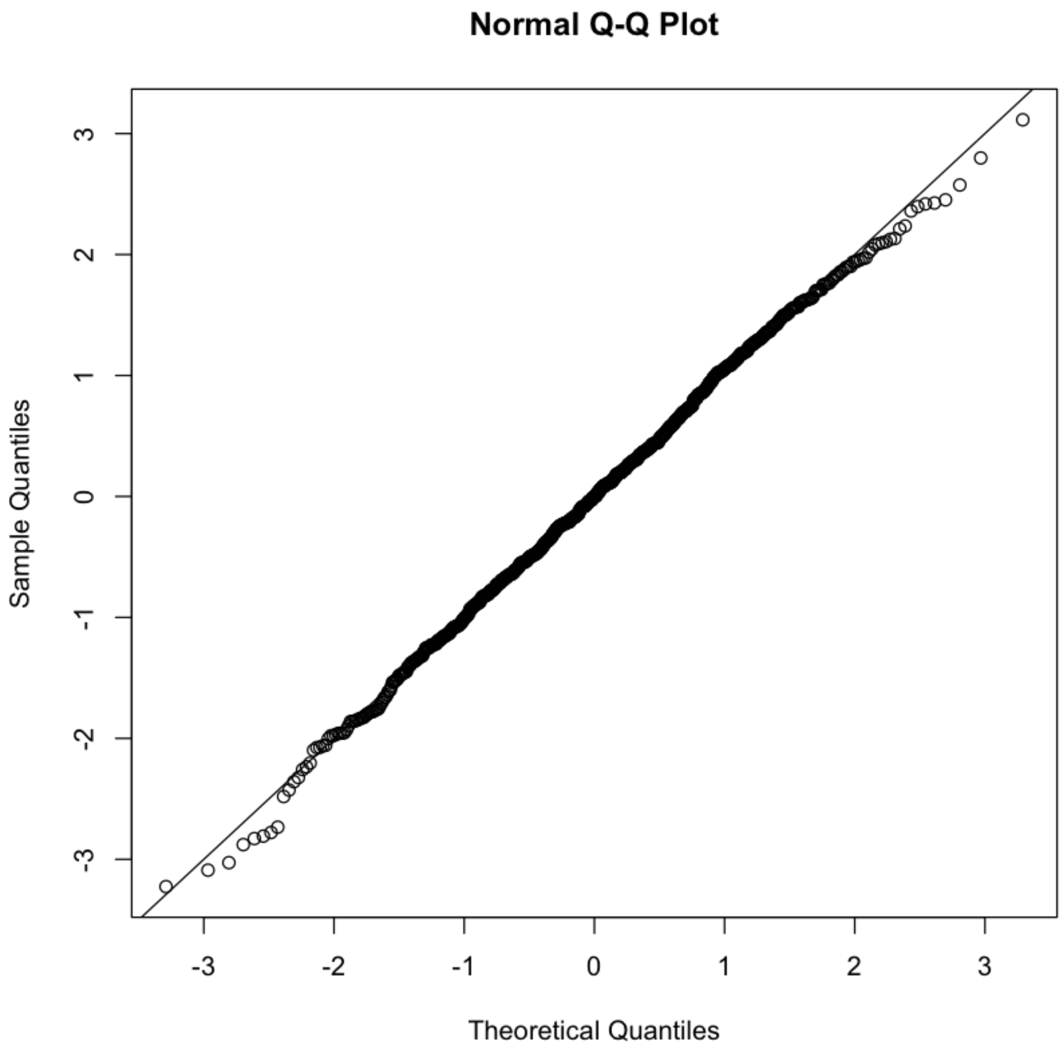



```
Computing CDFs… done
Computing residuals for pois_obs … done
Computing residuals for norm_obs … done
```



0.15
0.10
0.05
0.00
density
-5
0
5
10
norm_obs
State 1
State 2
Total


Normal Q-Q Plot
Sample Quantiles
Theoretical Quantiles
-3
-2
-1
0
1
2
3



1.4 Forecasting

Use the fitted model to generate a probability distribution for each future state.

Set the number of future time steps to predict. Calculate the parameter values at future time steps accounting for covariates using the predict method

[21]:
```
# Set number of observations to predict
n <- 10

# Create new data for prediction, optionally with covariates
forecast_data <- data.frame(
  ID = 1,
```

```
  pois_obs = rep(NA, n),
  norm_obs = rep(NA, n),
  covariate_1 = rnorm(n, mean = 0, sd = 1),
  covariate_2 = runif(n, min = -1, max = 1),
  covariate_3 = sample(c(-1, 0, 1), n, replace = TRUE)
  # random_effect_id = as.factor(rep(1:2, each = n/2))
)

forecast_data
```

A data.frame: 10 × 6

ID <dbl>	pois_obs <lgl>	norm_obs <lgl>	covariate_1 <dbl>	covariate_2 <dbl>	covariate_3 <dbl>
1	NA	NA	-0.27225030	0.2506252	1
1	NA	NA	-0.71720824	0.8678033	0
1	NA	NA	-0.19532618	-0.2374188	1
1	NA	NA	1.31326420	0.5469202	0
1	NA	NA	0.01268243	0.3950817	-1
1	NA	NA	-1.12421159	0.8369966	-1
1	NA	NA	0.85662803	-0.7556111	1
1	NA	NA	0.31418212	0.7315667	-1
1	NA	NA	0.33667190	0.7975774	1
1	NA	NA	0.26202593	-0.1804867	0

[22]:
```
## Generate forecast predictions using forecast module
eval_range <- list(
  pois_obs = 0:50,
  norm_obs = seq(-10, 10, by = 0.1)
)
forecasts <- Forecast$new(
  hmm = mod,
  forecast_data = forecast_data,
  preset_eval_range = eval_range,
  starting_state_distribution = "last"
)
forecasts
```

```
<Forecast>
  Public:
    clone: function (deep = FALSE)
    eval_range: function ()
    forecast_data: function ()
    forecast_dists: function ()
    hidden_state_forecast: function ()
    initialize: function (hmm = NULL, n = NULL, forecast_data = NULL,␣
↪preset_eval_range = NULL,
    obs_par_forecast: function ()
    tpm_forecast: function ()
    update_eval_range: function (eval_range)
```

```
 Private:
   configure_eval_range: function (eval_range = NULL, obs_vars = NULL, data =␣
↪NULL)
   eval_range_: list
   forecast_data_: data.frame
   forecast_dists_: list
   hidden_state_forecast_: 0.308420771001174 0.691579228998976 0.13647823595194␣
↪0.8 …
   hmm_: HMM, R6
   obs_par_forecast_: 1.76463149083606 1.01295621035596 3.18990795839716 2.117 …
   observation_vars_: pois_obs norm_obs
   starting_state_distribution_: last
   tpm_forecast_: 0.369199094883535 0.032692662687036 0.630800905116465 0. …
   validate_params: function (hmm = NULL, n = NULL, forecast_data = NULL,␣
↪preset_eval_range = NULL,
```

1.4.1 Plot the forecasted PDFs

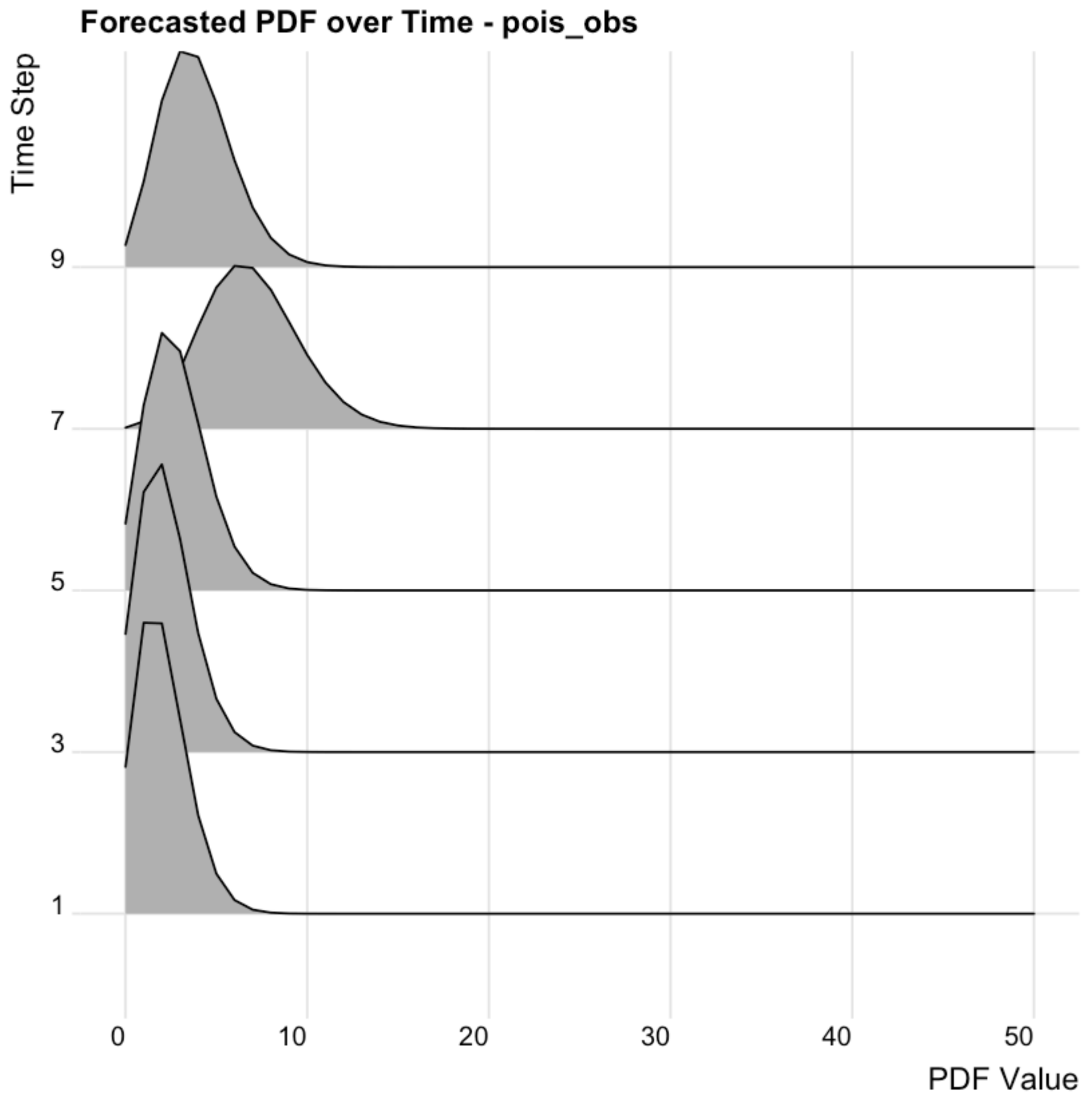
Generate a ridge plot to show the probability distribution for each time step. The plot shows every second time step to help declutter the plot.

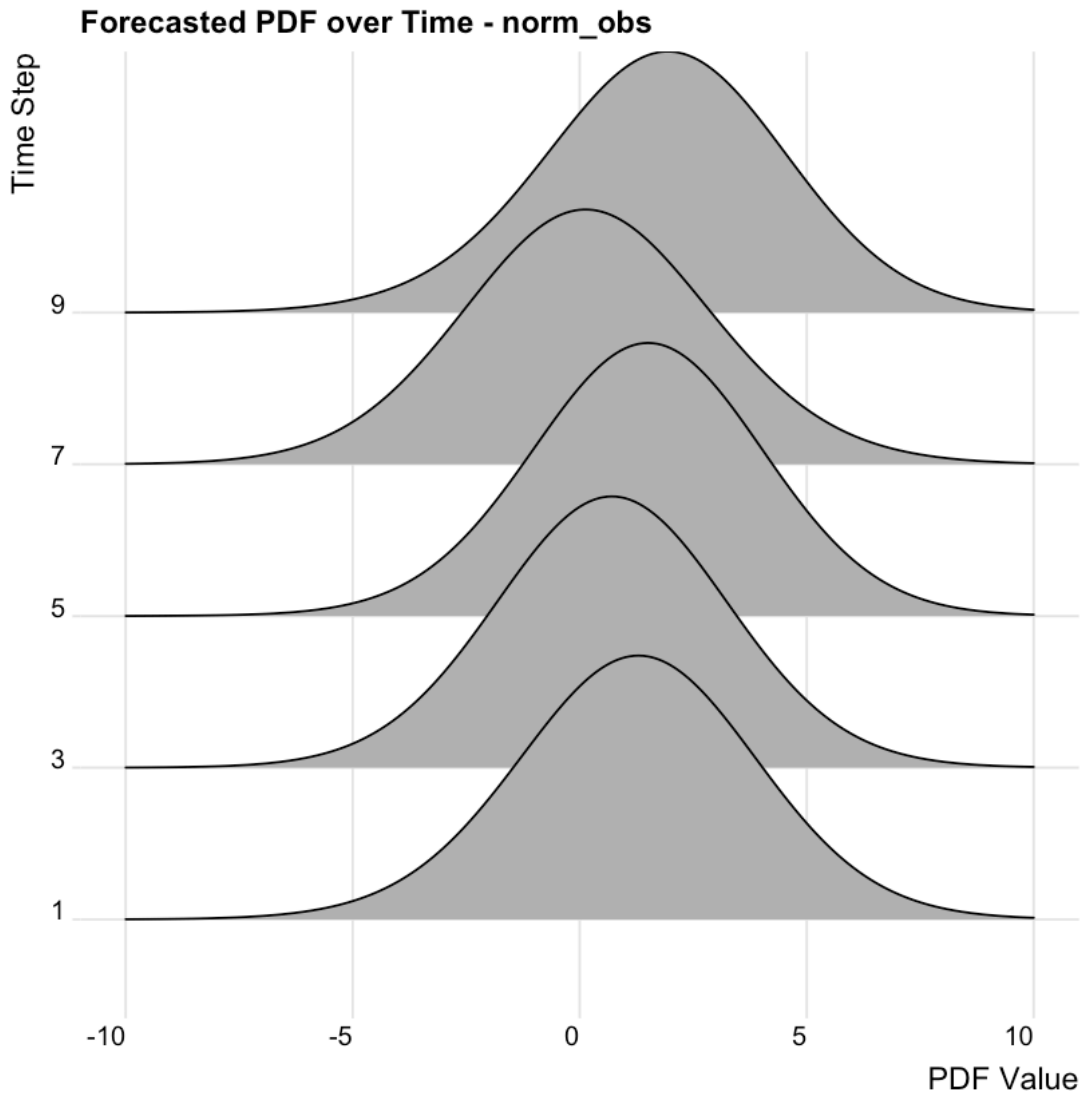
[25]:
```
forecasts$eval_range()
```

$pois_obs 1. 0 2. 1 3. 2 4. 3 5. 4 6. 5 7. 6 8. 7 9. 8 10. 9 11. 10 12. 11 13. 12 14. 13 15. 14 16. 15 17. 16 18. 17 19. 18 20. 19 21. 20 22. 21 23. 22 24. 23 25. 24 26. 25 27. 26 28. 27 29. 28 30. 29 31. 30 32. 31 33. 32 34. 33 35. 34 36. 35 37. 36 38. 37 39. 38 40. 39 41. 40 42. 41 43. 42 44. 43 45. 44 46. 45 47. 46 48. 47 49. 48 50. 49 51. 50

$norm_obs 1. -10 2. -9.9 3. -9.8 4. -9.7 5. -9.6 6. -9.5 7. -9.4 8. -9.3 9. -9.2 10. -9.1 11. -9 12. -8.9 13. -8.8 14. -8.7 15. -8.6 16. -8.5 17. -8.4 18. -8.3 19. -8.2 20. -8.1 21. -8 22. -7.9 23. -7.8 24. -7.7 25. -7.6 26. -7.5 27. -7.4 28. -7.3 29. -7.2 30. -7.1 31. -7 32. -6.9 33. -6.8 34. -6.7 35. -6.6 36. -6.5 37. -6.4 38. -6.3 39. -6.2 40. -6.1 41. -6 42. -5.9 43. -5.8 44. -5.7 45. -5.6 46. -5.5 47. -5.4 48. -5.3 49. -5.2 50. -5.1 51. -5 52. -4.9 53. -4.8 54. -4.7 55. -4.6 56. -4.5 57. -4.4 58. -4.3 59. -4.2 60. -4.1 61. -4 62. -3.9 63. -3.8 64. -3.7 65. -3.6 66. -3.5 67. -3.4 68. -3.3 69. -3.2 70. -3.1 71. -3 72. -2.9 73. -2.8 74. -2.7 75. -2.6 76. -2.5 77. -2.4 78. -2.3 79. -2.2 80. -2.1 81. -2 82. -1.9 83. -1.8 84. -1.7 85. -1.6 86. -1.5 87. -1.4 88. -1.3 89. -1.2 90. -1.1 91. -1 92. -0.9 93. -0.799999999999999 94. -0.699999999999999 95. -0.6 96. -0.5 97. -0.399999999999999 98. -0.299999999999999 99. -0.199999999999999 100. -0.0999999999999996 101. 0 102. 0.100000000000001 103. 0.200000000000001 104. 0.300000000000001 105. 0.4 106. 0.5 107. 0.600000000000001 108. 0.700000000000001 109. 0.800000000000001 110. 0.9 111. 1 112. 1.1 113. 1.2 114. 1.3 115. 1.4 116. 1.5 117. 1.6 118. 1.7 119. 1.8 120. 1.9 121. 2 122. 2.1 123. 2.2 124. 2.3 125. 2.4 126. 2.5 127. 2.6 128. 2.7 129. 2.8 130. 2.9 131. 3 132. 3.1 133. 3.2 134. 3.3 135. 3.4 136. 3.5 137. 3.6 138. 3.7 139. 3.8 140. 3.9 141. 4 142. 4.1 143. 4.2 144. 4.3 145. 4.4 146. 4.5 147. 4.6 148. 4.7 149. 4.8 150. 4.9 151. 5 152. 5.1 153. 5.2 154. 5.3 155. 5.4 156. 5.5 157. 5.6 158. 5.7 159. 5.8 160. 5.9 161. 6 162. 6.1 163. 6.2 164. 6.3 165. 6.4 166. 6.5 167. 6.6 168. 6.7 169. 6.8 170. 6.9 171. 7 172. 7.1 173. 7.2 174. 7.3 175. 7.4 176. 7.5 177. 7.6 178. 7.7 179. 7.8 180. 7.9 181. 8 182. 8.1 183. 8.2 184. 8.3 185. 8.4 186. 8.5 187. 8.6 188. 8.7 189. 8.8 190. 8.9 191. 9 192. 9.1 193. 9.2 194. 9.3 195. 9.4 196. 9.5 197. 9.6 198. 9.7 199. 9.8 200. 9.9 201. 10

[32]:
```
n_steps <- nrow(forecasts$forecast_data())
plot_steps <- seq(1, n_steps, by = 2)

obs_vars <- colnames(mod$obs()$obs_var())
for (obs in obs_vars) {
  # Prepare data for ridge plot
  ridge_data <- data.frame(
    x = rep(
      forecasts$eval_range()[[obs]],
      times = length(plot_steps)
    ),
    y = as.vector(forecasts$forecast_dists()[[obs]][ , plot_steps]),
    time = factor(rep(plot_steps, each = length(forecasts$eval_range()[[obs]])))
  )

  print(ggplot(ridge_data, aes(x = x, y = time, height = y, group = time)) +
    geom_density_ridges(stat = "identity", position = "identity") +
    labs(x = "PDF Value", y = "Time Step", title = paste("Forecasted PDF over␣
 ↪Time -", obs)) +
    theme_ridges() +
    theme(axis.text.x = element_text(angle = 0, hjust = 1)))
}
```


Forecasted PDF over Time - pois_obs
Time Step
9
7
5
3
1
0
10
20
30
40
50
PDF Value



1.5 Validate

Create 1000 simulations of the true model using the ending state and the same covariates used in forecasting.

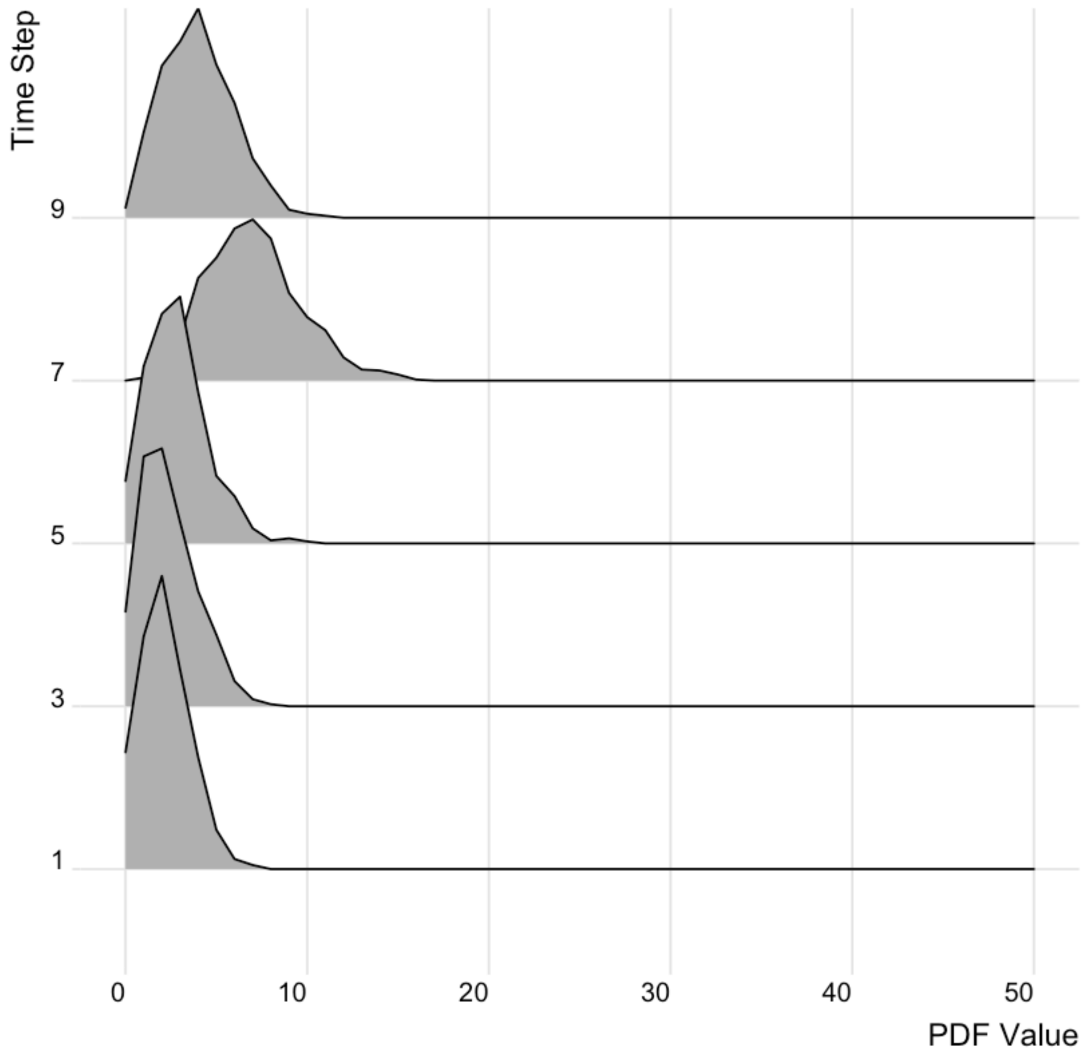
[31]:
```
n_dimensions <- length(obs_vars)
n_simulations <- 1000

# Get last hidden state
true_last_state <- tail(attr(true_mod$obs()$data(), "state"), 1) # last state
# get TPM of last observation
last_tpm <- true_mod$hid()$tpm(nrow(true_mod$obs()$data()))[, , 1]
```

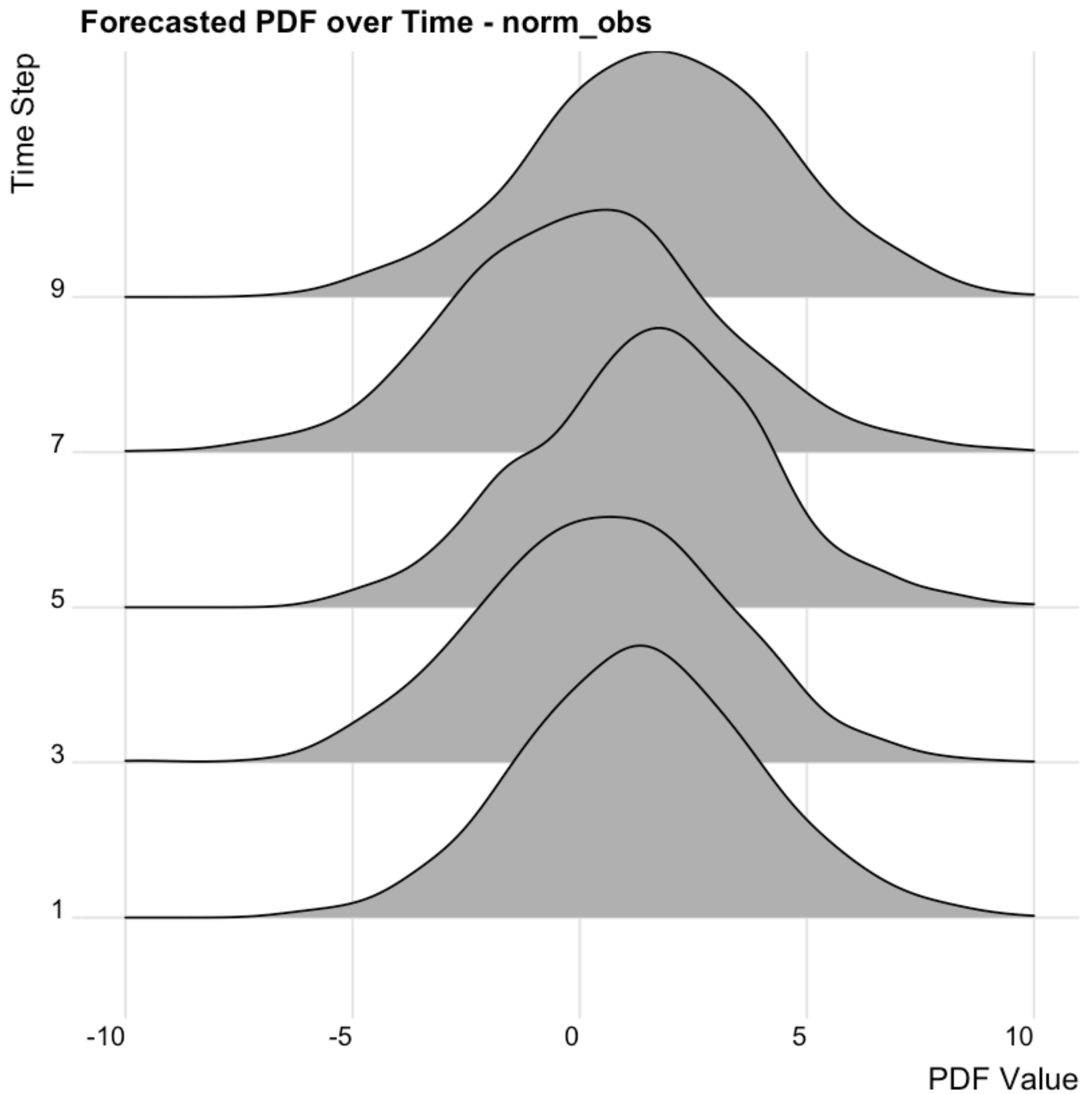
```r
# Update delta0 to use distribution of first forecasted state
true_mod$hid()$update_delta0(last_tpm[true_last_state, ])

# Initialize a named list of 2D arrays to store the simulation results
simulated_data <- list()
for (obs in obs_vars) {
  simulated_data[[obs]] <- array(NA, dim = c(n_simulations, n_steps))
}

# Simulate data and store it in the array
for (i in 1:n_simulations) {
  sim <- true_mod$simulate(n = n_steps, data = forecast_data, silent = TRUE)

  for (dimension in obs_vars) {
    # Get the simulated data for the current dimension
    simulated_data[[dimension]][i, ] <- sim[[dimension]]
  }
}

# Optionally, print a subset of the simulated data
print(simulated_data[[1]][1:5, ]) # Print the first 5 simulations
```

[38]:
```r
# Convert simulated data to a histogram at each time step
discrete_pdf <- function(sample_mat, eval_range) {
  apply(sample_mat, 2, function(col) {
    col <- as.numeric(col)
    counts <- table(factor(col, levels = eval_range))
    as.numeric(counts) / length(col)
  })
}

kde_simple <- function(x, eval_range, bw) {
  x <- as.numeric(x)
  dens <- density(x,
                  from   = min(eval_range),
                  to     = max(eval_range),
                  n      = length(eval_range),
                  bw     = "SJ",
                  cut    = 0,
                  kernel = "gaussian")
  dens$y
}

kde_pdf <- function(sample_mat, eval_range, bw) {
  apply(sample_mat, 2, function(col) {
    col <- as.numeric(col)
    kde_simple(col, eval_range, bw)
```

```r
  })
}

simulated_pdfs <- vector("list", length(obs_vars))
names(simulated_pdfs) <- names(obs_vars)
for (obs in obs_vars) {
  eval_range <- forecasts$eval_range()[[obs]]
  bw <- diff(eval_range)[1]
  if (bw == 1) {
    # discreet case
    simulated_pdfs[[obs]]  <- discrete_pdf(simulated_data[[obs]], eval_range)
  } else {
    simulated_pdfs[[obs]] <- kde_pdf(simulated_data[[obs]], eval_range, bw)
  }

}
# Plot the simulated data as a ridge plot
for (obs in obs_vars) {
  # Prepare data for ridge plot
  ridge_data <- data.frame(
    x = rep(
      forecasts$eval_range()[[obs]],
      times = length(plot_steps)
    ),
    y = as.vector(simulated_pdfs[[obs]][ , plot_steps]),
    time = factor(rep(plot_steps, each = length(forecasts$eval_range()[[obs]])))
  )

  print(ggplot(ridge_data, aes(x = x, y = time, height = y, group = time)) +
    geom_density_ridges(stat = "identity", position = "identity") +
    labs(x = "PDF Value", y = "Time Step", title = paste("Forecasted PDF over␣
↪Time -", obs)) +
    theme_ridges() +
    theme(axis.text.x = element_text(angle = 0, hjust = 1)))
}
```

Forecasted PDF over Time - pois_obs

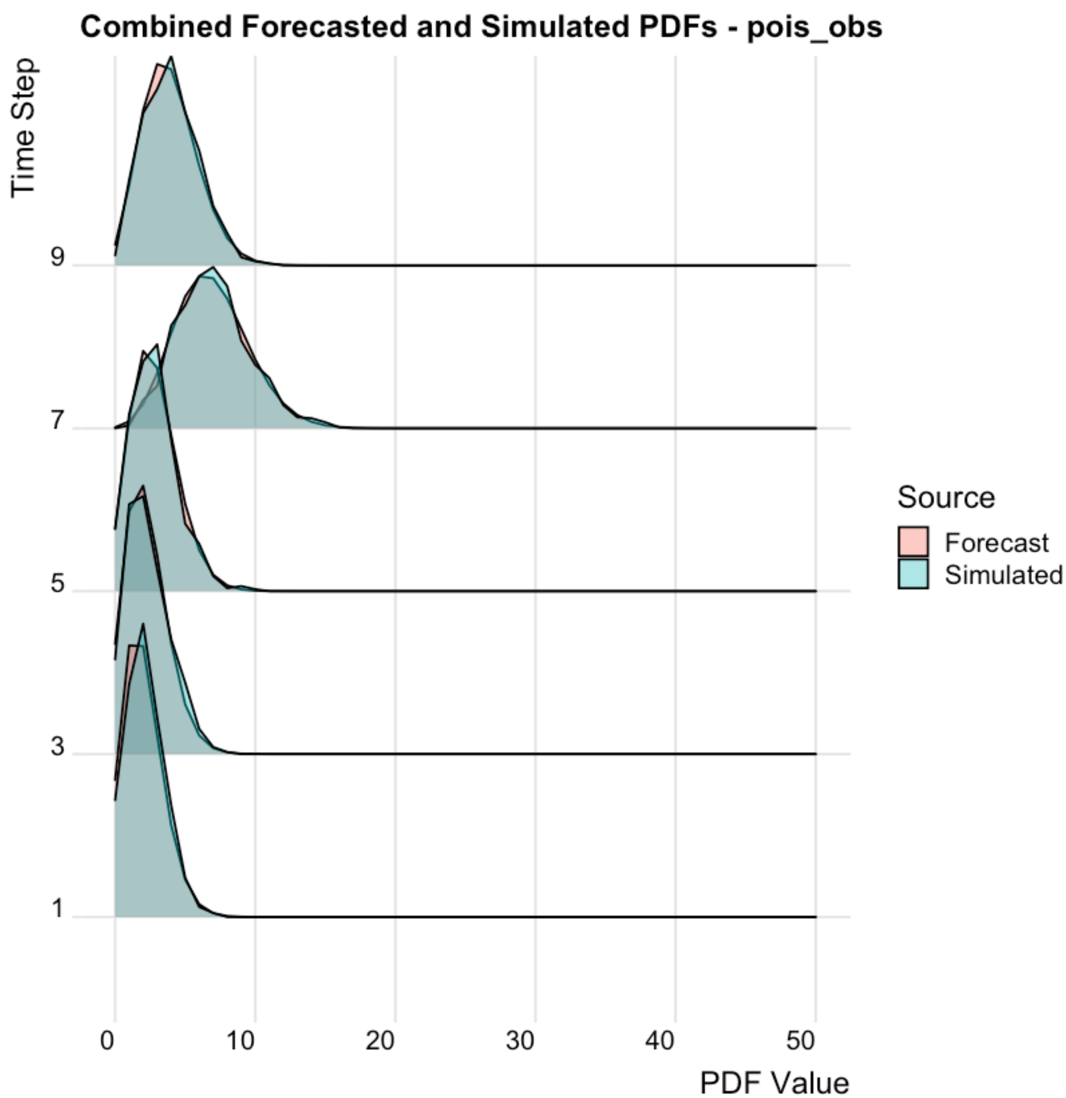




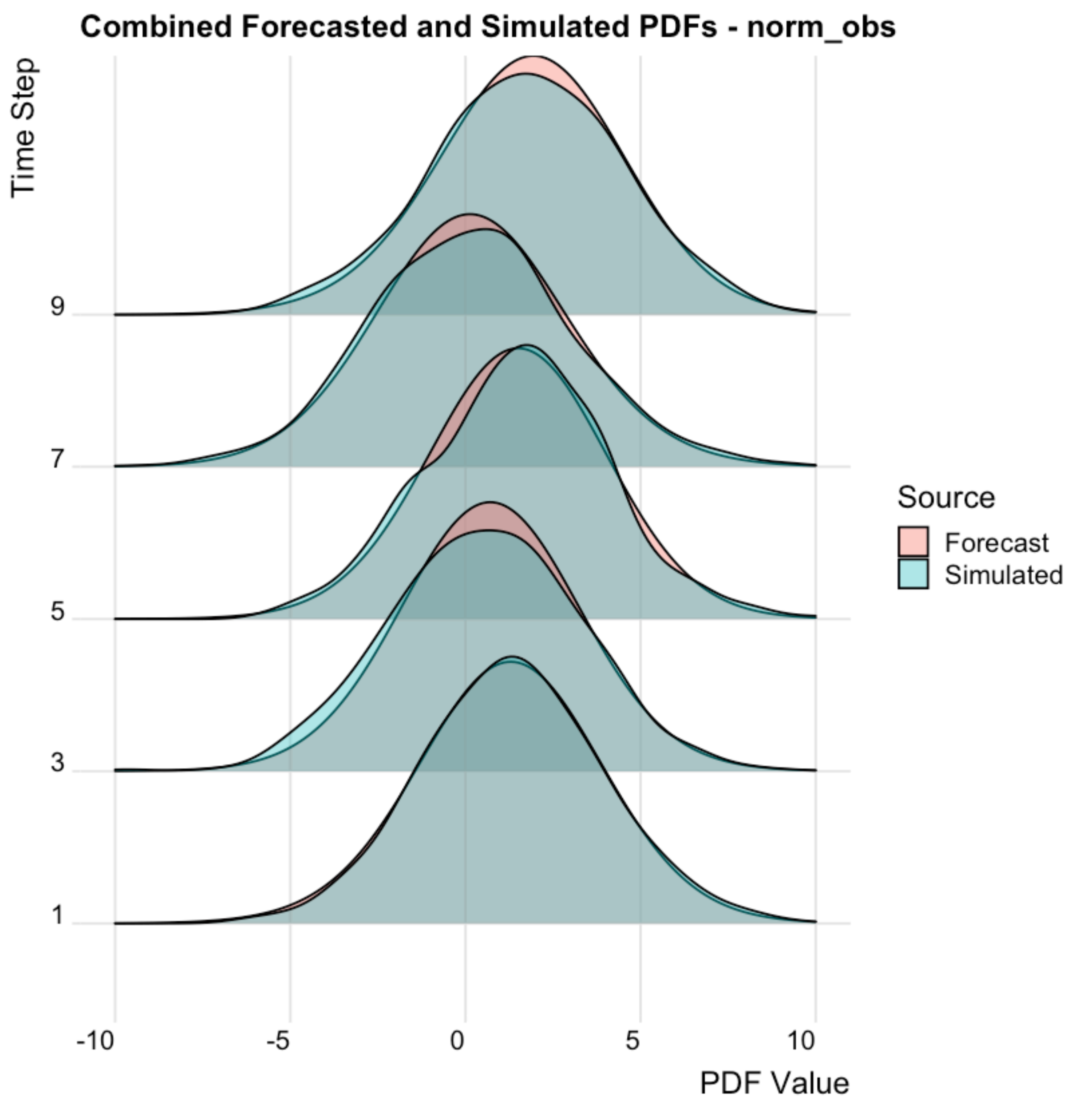
Compare forecasted PDFs with simulated PDFs

[40]:
```
# Define the number of steps and the dimension to plot
n_steps <- nrow(forecasts$forecast_data())
plot_steps <- seq(1, n_steps, by = 2)  # Plot every second step for clarity

for (dimension in obs_vars) {
  # Get forecasted PDFs and simulated PDFs
  eval_range <- forecasts$eval_range()[[dimension]]
  forecast_pdfs <- forecasts$forecast_dists()[[dimension]]
  simulated_pdfs_dim <- simulated_pdfs[[dimension]]

  # Prepare data for ridge plot
```

```
  ridge_data <- data.frame(
    x = rep(eval_range, times = length(plot_steps) * 2),
    y = c(
      as.vector(forecast_pdfs[, plot_steps]),
      as.vector(simulated_pdfs_dim[, plot_steps])
    ),
    time = factor(rep(plot_steps, each = length(eval_range))),
    source = rep(c("Forecast", "Simulated"), each = length(eval_range) *␣
↪length(plot_steps))
  )

  # Create the ridge plot
  print(
    ggplot(ridge_data, aes(x = x, y = time, height = y, group =␣
↪interaction(time, source), fill = source)) +
      geom_density_ridges(stat = "identity", position = "identity", alpha = 0.
↪4) +
      labs(
        x = "PDF Value",
        y = "Time Step",
        title = paste("Combined Forecasted and Simulated PDFs -", dimension),
        fill = "Source"
      ) +
      theme_ridges() +
      theme(axis.text.x = element_text(angle = 0, hjust = 1))
  )
}
```



Combined Forecasted and Simulated PDFs - pois_obs
Time Step
9
7
5
3
1
0
10
20
30
40
50
PDF Value
Source
Forecast
Simulated


Combined Forecasted and Simulated PDFs - norm_obs
Time Step
9
7
5
3
1
-10
-5
0
5
10
PDF Value
Source
Forecast
Simulated