

Four-State HMM Dummy Bear Example

Supplementary Information for: A hidden Markov framework for joint identification of animal activity modes and movement phases

Contents

1	Introduction	3
2	Package Setup and Data Loading	3
2.1	Required Packages	3
2.2	Loading the Example Data	4
3	Data Processing	5
3.1	Trajectory Visualization by Residence Time	5
3.2	Estimating Step Lengths and Turning Angles	6
4	ACHMM Model Specification	7
4.1	Initial Parameter Values	7
4.2	Parameter Constraints	8
4.3	Transition Probability Matrix	10
5	Model Fitting	11
5.1	Building the HMM Object	11
5.2	Model Fitting	12
6	Model Results Visualization	13
6.1	State-Dependent Distributions	13
6.1.1	Step Length Distributions	14
6.1.2	Turning Angle Distributions	15
6.1.3	Residence Time Distributions	17
7	Viterbi Algorithm for State Decoding	18

8	Trajectory Visualization with Decoded States	19
9	Home Ranges	21
9.1	Home Range Estimation	22
9.2	Home Range Visualization	22
10	Conclusion	24
11	Session Information	25

1 Introduction

This supplementary material provides a detailed walkthrough of the Asymmetric Coupled Hidden Markov Model (ACHMM) framework applied to animal movement data. The example uses a dummy bear GPS dataset to demonstrate the workflow for classifying animal activity modes and movement phases.

The analysis employs a HMM with 4-states (Movement Phase - Activity Mode):

- **State 1: Non-Resident - Inactive**
- **State 2: Non-Resident - Moving**
- **State 3: Resident - Inactive**
- **State 4: Resident - Moving**

And 3 data streams:

- **Step lengths** (distance between consecutive locations)
- **Turning angles** (changes in direction)
- **Residence time** (time spent in localized areas)

A detailed description of the approach can be consulted: Cisneros-Araujo, P., Gastón, A., Cubero, D., Pinto, D., Saura, S., & Rodríguez de Rivera, Ó. (2026). A hidden Markov framework for joint identification of animal activity modes and movement phases. *Landscape Ecology*. <https://doi.org/10.1007/s10980-026-02304-3>

2 Package Setup and Data Loading

2.1 Required Packages

The analysis utilizes several specialized R packages for movement analysis, hidden Markov modeling, and spatial visualization:

```
# Load required packages
library(momentuHMM)      # Movement-based HMM functionality
library(hmmTMB)          # Template Model Builder for flexible HMM specification
library(ggplot2)         # Advanced graphics
library(lubridate)       # Date/time handling
library(dplyr)           # Data manipulation
library(amt)             # Animal Movement Tools for home range estimation
library(sf)              # Simple Features for spatial data
library(raster)          # Raster data handling
```

2.2 Loading the Example Data

Data Description:

We begin by loading the pre-processed dummy bear dataset, which contains GPS location data with associated covariates.

The GPS dataset contains the following columns:

- **x, y:** Coordinates (projected in meters)
- **ID:** Individual animal identifier
- **time:** Timestamp of the observation
- **restime:** Residence time (in days)

```
# Load the dummy bear dataset. File path may need to be modified:
dummy.bear <- readRDS(file = "~/Documents/dummy_bear.rds")
```

```
# Examine data structure
head(dummy.bear)
```

```
##           x           y           time           ID  restime
## 1 1901.037 11789.01 2000-01-01 00:00:00 dummy.bear 2.686143
## 2 1979.537 11760.46 2000-01-01 01:00:00 dummy.bear 3.350671
## 3 2014.713 11622.92 2000-01-01 02:00:00 dummy.bear 2.891956
## 4 2200.809 11855.97 2000-01-01 03:00:00 dummy.bear 4.489948
## 5 2029.197 11789.04 2000-01-01 04:00:00 dummy.bear 3.533707
## 6 1983.031 11778.37 2000-01-01 05:00:00 dummy.bear 3.371681
```

```
str(dummy.bear)
```

```
## 'data.frame': 1584 obs. of 5 variables:
## $ x : num 1901 1980 2015 2201 2029 ...
## $ y : num 11789 11760 11623 11856 11789 ...
## $ time : POSIXct, format: "2000-01-01 00:00:00" "2000-01-01 01:00:00" ...
## $ ID : chr "dummy.bear" "dummy.bear" "dummy.bear" "dummy.bear" ...
## $ restime: num 2.69 3.35 2.89 4.49 3.53 ...
```

```
summary(dummy.bear)
```

```
##           x           y           time
## Min.      : 0      Min.      : 0      Min.      :2000-01-01 00:00:00
## 1st Qu.:2578    1st Qu.:10099    1st Qu.:2000-01-17 11:45:00
## Median :2899    Median :12006    Median :2000-02-02 23:30:00
## Mean    :2868    Mean     :11066    Mean     :2000-02-02 23:30:00
```

```
## 3rd Qu.:3229    3rd Qu.:12966    3rd Qu.:2000-02-19 11:15:00
## Max.    :6486    Max.    :16243    Max.    :2000-03-06 23:00:00
##      ID                restime
## Length:1584          Min.    : 0.03823
## Class :character     1st Qu.: 1.62288
## Mode  :character     Median : 5.55727
##                      Mean    : 6.57989
##                      3rd Qu.:10.91737
##                      Max.    :16.18099
```

3 Data Processing

3.1 Trajectory Visualization by Residence Time

Before fitting the HMM, we visualize the raw trajectory data, highlighting areas where the bear spent considerable time (high residence time). Residence time was estimated with the **recurse** R package.

```
p_restime <- ggplot(dummy.bear, aes(x = x, y = y)) +
  geom_path(color = "grey70", size = 0.3) +
  geom_point(
    aes(color = restime),
    size = 1,
    alpha = 0.75
  ) +
  scale_color_gradient(
    name = "Residence Time \n(days)",
    low = "cyan3",
    high = "red"
  ) +
  coord_equal() +
  labs(
    title = "Dummy Bear Trajectory",
    x = "X Coordinate (m)",
    y = "Y Coordinate (m)"
  ) +
  theme_minimal(base_size = 16) +
  theme(
    plot.title = element_text(size = 16, hjust = 0.5),
    legend.title = element_text()
  )
```

```
print(p_restime)
```

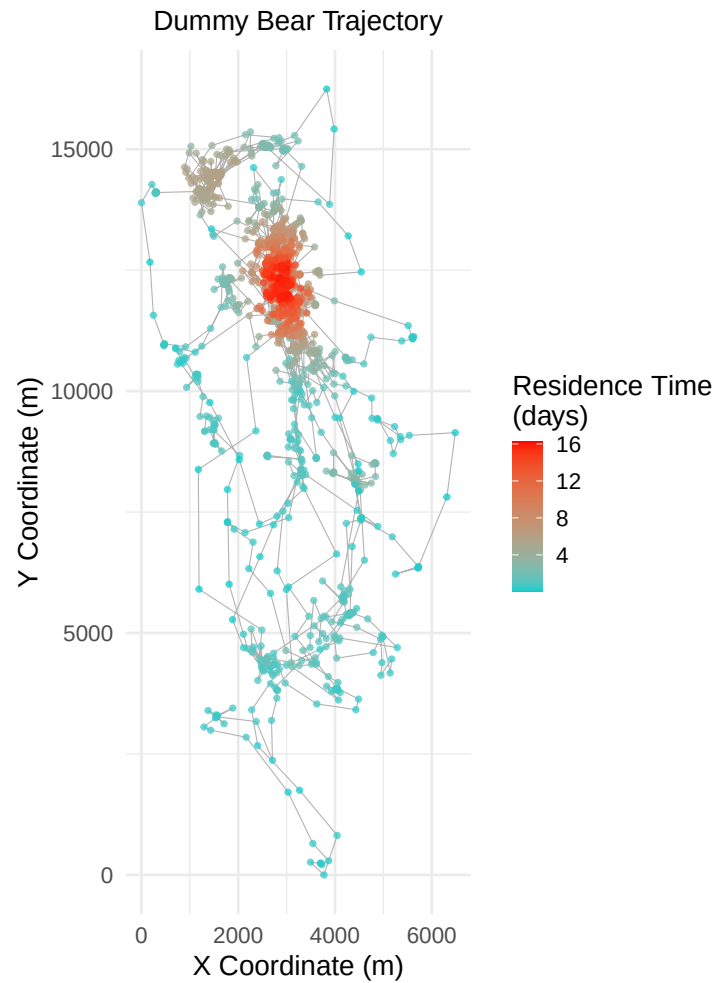


Figure 1: Bear trajectory mapped by residence time. Warmer colors indicate areas where the bear spent extended periods (higher residency), while cooler colors indicate brief visits.

3.2 Estimating Step Lengths and Turning Angles

We extract step lengths (distances between consecutive locations) and turning angles (changes in direction) easily using the `momentuHMM` R package.

Note: GPS data should have a regular sampling rate (e.g., 1 loc./h)

```
# Prepare data by computing step lengths and turning angles
# The prepData function standardizes the movement data for HMM analysis
```

```
data <- prepData(dummy.bear)
```

```
# Examine the processed data
head(data)
```

```
##           ID      step      angle      time  restime      x
## 1 dummy.bear  83.529986      NA 2000-01-01 00:00:00 2.686143 1901.037
## 2 dummy.bear 141.970213 -0.9715916 2000-01-01 01:00:00 3.350671 1979.537
## 3 dummy.bear 298.236509  2.2173765 2000-01-01 02:00:00 2.891956 2014.713
## 4 dummy.bear 184.201735  2.6164905 2000-01-01 03:00:00 4.489948 2200.809
## 5 dummy.bear  47.384576 -0.1445915 2000-01-01 04:00:00 3.533707 2029.197
## 6 dummy.bear   3.333508  1.3763723 2000-01-01 05:00:00 3.371681 1983.031
##           y
## 1 11789.01
## 2 11760.46
## 3 11622.92
## 4 11855.97
## 5 11789.04
## 6 11778.37
```

```
# Convert step lengths to kilometers for interpretability
# (often easier to interpret than raw meters)
data$step <- data$step / 1000
```

4 ACHMM Model Specification

4.1 Initial Parameter Values

We specify starting values for each observed data stream. These initial values work well for the dummy bear, but users applying this framework to other species may wish to modify these starting values and explore several alternative initializations to ensure robust model fitting and that the model does not converge to a local maximum.

```
# Data Stream 1: Step Lengths
# Use gamma2 distribution (Gamma distribution parameterized by mean and sd)
step_mean0 <- c(0.02, 0.5, 0.02, 0.2) # one value for each of the 4 states
step_sd0   <- c(0.02, 0.5, 0.02, 0.2)

# Data Stream 2: Turning Angles
# Use von Mises (vm) distribution, parameterized by mean (mu) & concentration (kappa)
```

```

angle_mu0 <- c(0, 0, 0, 0)
angle_kappa0 <- c(0.5, 0.7, 0.5, 0.3)

# Data Stream 3: Residence Time
# Use Weibull distribution, parameterized by shape and scale
restime_shape0 <- c(8, 8, 2, 2)
restime_scale0 <- c(2, 2, 10, 10)

# Combine into a parameter list
par0 <- list(
  step = list(mean = step_mean0, sd = step_sd0),
  angle = list(mu = angle_mu0, kappa = angle_kappa0),
  restime = list(shape = restime_shape0, scale = restime_scale0)
)

# Define distributions for each data stream
dists <- list(
  step = "gamma2",
  angle = "vm",
  restime = "weibull"
)

```

4.2 Parameter Constraints

We apply movement constraints across states to implement the ACHMM structure and improve identifiability. The constraints enforced are:

- **Inactive states (1, 3)** share movement parameters (step lengths and turning angles) since an inactive mode might have the same characteristics regardless of the movement phase it belongs
- **Non-Resident states (1, 2)** share residence time parameters
- **Resident states (3, 4)** share residence time parameters

We can link parameters with the `hmmTMB` package.

```

fixpar <- list(obs = c(

# Inactive states should not vary much depending on
# the broader movement phase (Resident/Non-Resident).

# So states 1 (Non-Resident - Inactive) & 3 (Resident - Inactive)
# share the same step length and turning angle parameters.

```



```

#Step lengths parameters:

#State 1 & 3 (Inactive) have the same step length mean
"step.mean.state1.(Intercept)" = 1,
"step.mean.state3.(Intercept)" = 1,
#State 1 & 3 (Inactive) have the same step length sd
"step.sd.state1.(Intercept)" = 2,
"step.sd.state3.(Intercept)" = 2,

# Turning angle parameters:

#State 1 & 3 (Inactive) have the same angle mu
"angle.mu.state1.(Intercept)" = 3,
"angle.mu.state3.(Intercept)" = 3,
#State 1 & 3 (Inactive) have the same angle kappa
"angle.kappa.state1.(Intercept)" = 4,
"angle.kappa.state3.(Intercept)" = 4,

# This step is key for the implementation of the ACHMM:

# Non-Resident states (states 1 and 2) share the same residence time parameters,
# different from Resident states (states 3 and 4).

# Resident states (states 3 and 4) share the same residence time parameters,
# different from Non-Resident states (states 1 and 2).

# Residence time parameters:

#States 1 & 2 (Non-Resident) have the same residence time shape
"restime.shape.state1.(Intercept)" = 5,
"restime.shape.state2.(Intercept)" = 5,
#States 3 & 4 (Resident) have the same residence time shape
"restime.shape.state3.(Intercept)" = 6,
"restime.shape.state4.(Intercept)" = 6,
#States 1 & 2 (Non-Resident) have the same residence time scale
"restime.scale.state1.(Intercept)" = 7,
"restime.scale.state2.(Intercept)" = 7,
#States 3 & 4 (Resident) have the same residence time scale
"restime.scale.state3.(Intercept)" = 8,
"restime.scale.state4.(Intercept)" = 8
))

```

Interpretation:

Fix parameter indices (1, 2, 3,..., 8) are used to enforce a structure in the HMM. This constrains certain parameters to be equal across states, reflecting shared movement characteristics. For example, if Non-Resident states (states 1 and 2) share the same residence time shape parameter, they both have the same index (in this case 5).

Note: This does NOT mean that the shape parameter = 5. It just means that those states with a 5 (states 1 & 2) will have the same value in the shape parameter.

4.3 Transition Probability Matrix

The transition probability matrix (TPM) specifies the probability of switching from one movement state to another. These starting values should reflect reasonable assumptions or expected movement patterns of the species:

```
# Define initial transition probability matrix
# Rows = current state; Columns = next state
# Each row must sum to 1.0

# State 1: Non-Resident - Inactive
# State 2: Non-Resident - Moving
# State 3: Resident - Inactive
# State 4: Resident - Moving

#We expect more switches between Inactive Active than between Non-Resident Resident.
tpm0 <- matrix(c(
  0.82, 0.16, 0.01, 0.01, # From State 1 : Mostly remain, some to State 2
  0.16, 0.80, 0.01, 0.03, # From State 2: Mostly remain, some return to State 1
  0.01, 0.01, 0.80, 0.18, # From State 3: Mostly remain, some to State 4
  0.01, 0.02, 0.15, 0.82 # From State 4: Mostly remain, some return to State 3
), ncol = 4, byrow = TRUE)

# Add informative labels
colnames(tpm0) <- paste0("state ", 1:4)
rownames(tpm0) <- paste0("state ", 1:4)

print(tpm0)
```

```
##          state 1 state 2 state 3 state 4
```

```
## state 1    0.82    0.16    0.01    0.01
## state 2    0.16    0.80    0.01    0.03
## state 3    0.01    0.01    0.80    0.18
## state 4    0.01    0.02    0.15    0.82
```

```
# Verify rows sum to 1
rowsum_check <- rowSums(tpm0)
print(rowsum_check)
```

```
## state 1 state 2 state 3 state 4
##      1      1      1      1
```

Interpretation:

- High diagonal values (>0.80) indicate persistence within the same state
- Non-zero off-diagonal values allow transitions between states
- The structure reflects biologically plausible dynamics: switches between Inactive and Active states are expected to be more common than between Non-Resident and Resident states

5 Model Fitting

5.1 Building the HMM Object

We construct the HMM using the `hmmTMB` package:

```
# Create the Observation process object
# This specifies the distributions and parameters for observed data
obs <- Observation$new(
  data = data,
  dists = dists,
  par = par0,
  n_states = 4
)

# Create the Markov Chain object
# This specifies the hidden state process and transitions
hid <- MarkovChain$new(
  data = data,
  n_states = 4,
  tpm = tpm0,
```

```

  initial_state = "stationary" # Start from stationary distribution
)

# Combine into a HMM object
hmm_coupled_4st <- HMM$new(
  obs = obs,
  hid = hid,
  fixpar = fixpar
)

```

5.2 Model Fitting

```

# Fit the HMM
# silent = TRUE suppresses iteration details
hmm_coupled_4st$fit(silent = TRUE)

# Display model summary
print(hmm_coupled_4st)

```

```

## #####
## ## Observation model ##
## #####
## + step ~ gamma2(mean, sd)
##   * mean.state1 ~ 1
##   * mean.state2 ~ 1
##   * mean.state3 ~ 1
##   * mean.state4 ~ 1
##   * sd.state1 ~ 1
##   * sd.state2 ~ 1
##   * sd.state3 ~ 1
##   * sd.state4 ~ 1
##
## + angle ~ vm(mu, kappa)
##   * mu.state1 ~ 1
##   * mu.state2 ~ 1
##   * mu.state3 ~ 1
##   * mu.state4 ~ 1
##   * kappa.state1 ~ 1
##   * kappa.state2 ~ 1
##   * kappa.state3 ~ 1
##   * kappa.state4 ~ 1
##

```

```
## + restime ~ weibull(shape, scale)
## * shape.state1 ~ 1
## * shape.state2 ~ 1
## * shape.state3 ~ 1
## * shape.state4 ~ 1
## * scale.state1 ~ 1
## * scale.state2 ~ 1
## * scale.state3 ~ 1
## * scale.state4 ~ 1
##
## > Estimated observation parameters (t = 1):
##           state 1 state 2 state 3 state 4
## step.mean    0.024  0.500  0.024  0.295
## step.sd       0.018  0.407  0.018  0.206
## angle.mu      3.035  0.159  3.035 -0.571
## angle.kappa   0.475  0.261  0.475  0.155
## restime.shape 1.410  1.410  3.008  3.008
## restime.scale 1.716  1.716 11.343 11.343
##
## #####
## ## State process model ##
## #####
##           state 1 state 2 state 3 state 4
## state 1      .      ~1      ~1      ~1
## state 2      ~1      .      ~1      ~1
## state 3      ~1      ~1      .      ~1
## state 4      ~1      ~1      ~1      .
##
## > Estimated transition probabilities (t = 1):
##           state 1 state 2 state 3 state 4
## state 1    0.673  0.327  0.000  0.000
## state 2    0.135  0.820  0.002  0.042
## state 3    0.000  0.000  0.538  0.462
## state 4    0.002  0.028  0.157  0.813
```

6 Model Results Visualization

6.1 State-Dependent Distributions

Visualizing the estimated state-dependent distributions show the movement characteristics associated to each state:

6.1.1 Step Length Distributions

The step length distributions show the expected distance traveled under each movement state:

```
state_labels <- c(
  "State 1" = "Non-Resident - Inactive",
  "State 2" = "Non-Resident - Moving",
  "State 3" = "Resident - Inactive",
  "State 4" = "Resident - Moving",
  "Total" = "Total"
)

custom_colors <- c(
  "State 1" = "#1E90FF",
  "State 2" = "#1E90FF",
  "State 3" = "#006400",
  "State 4" = "#006400",
  "Total" = "black"
)

custom_lines <- c(
  "State 1" = "longdash",
  "State 2" = "solid",
  "State 3" = "dashed",
  "State 4" = "solid",
  "Total" = "dashed"
)

hmm_coupled_4st$plot_dist("step") +
  coord_cartesian(ylim = c(0, 2), xlim = c(0, 2)) +
  theme(legend.position = c(0.75, 0.8)) +
  labs(
    x = "Step Length (km)",
    title = "State-Dependent Step Length Distributions"
  ) +
  scale_linetype_manual(
    name = "States",
    values = custom_lines,
    labels = state_labels
  ) +
  scale_color_manual(
    name = "States",
    values = custom_colors,
    labels = state_labels
  )
```

```
) +
theme_minimal(base_size = 16) +
theme(
  plot.title = element_text(hjust = 0.5),
  legend.title = element_text()
)
```

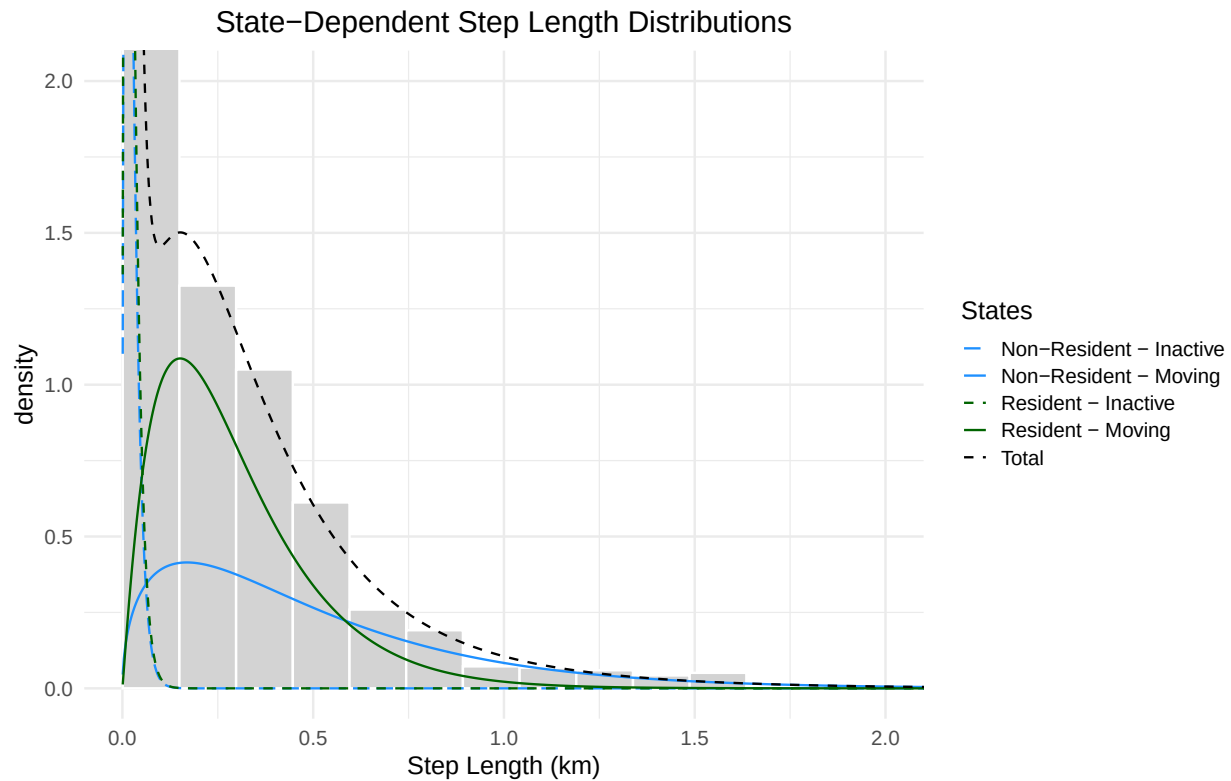


Figure 2: Estimated step length distributions by movement state. State 1 and 3 (inactive) show shorter mean step lengths; states 2 and 4 (moving) show broader distributions at larger distances. Particularly, state 2 (Non-Resident - Moving), which shows longer step lengths than state 4 (Resident - Moving)

6.1.2 Turning Angle Distributions

The turning angle distributions reveal directional persistence in each state:

```
hmm_coupled_4st$plot_dist("angle") +
  theme(legend.position = c(0.75, 0.8)) +
  scale_x_continuous(
    breaks = seq(-pi, pi, by = pi/2),
    labels = expression(-pi, -pi/2, 0, pi/2, pi)
  ) +
  labs(
```

```

x = "Turning Angle (radians)",
title = "State-Dependent Turning Angle Distributions"
) +
scale_linetype_manual(
  name = "States",
  values = custom_lines,
  labels = state_labels
) +
scale_color_manual(
  name = "States",
  values = custom_colors,
  labels = state_labels
) +
theme_minimal(base_size = 16) +
theme(
  plot.title = element_text(hjust = 0.5),
  legend.title = element_text()
)

```

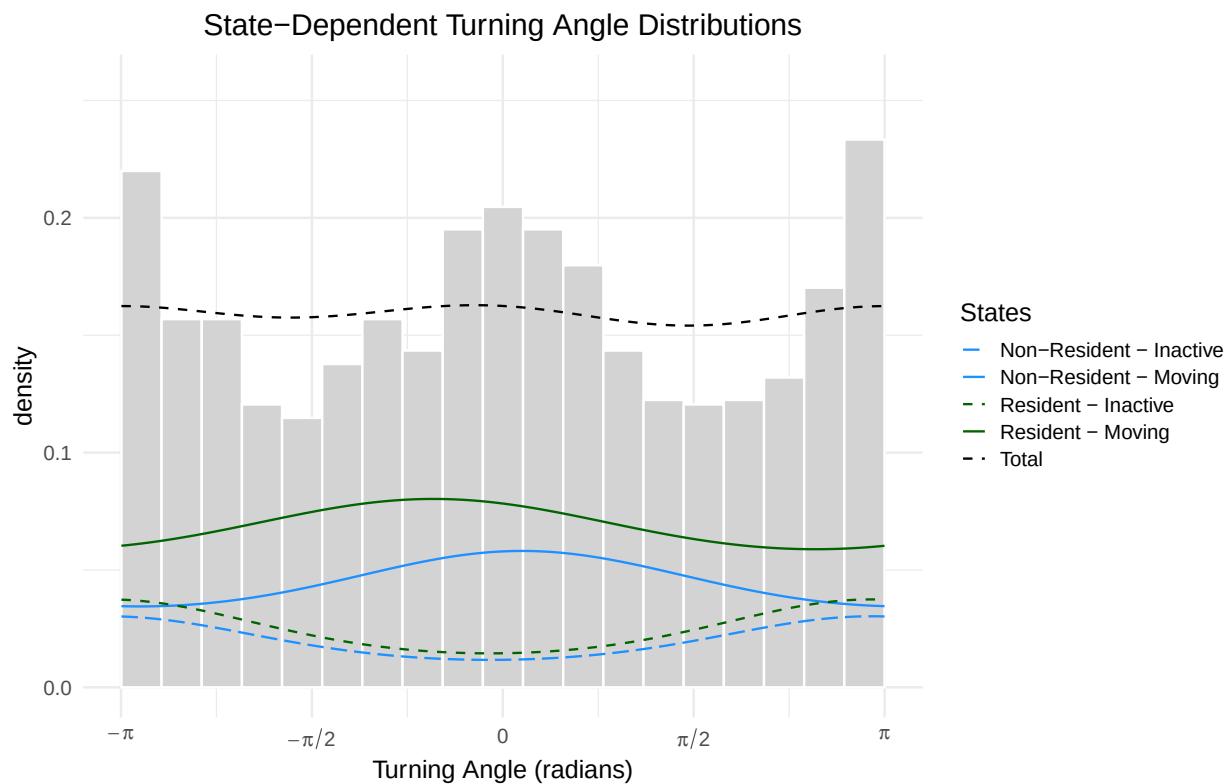


Figure 3: Estimated turning angle distributions (von Mises). Peaked distributions at 0 indicate directional persistence; flat distributions indicate random turning.

6.1.3 Residence Time Distributions

The residence time distributions show expected intensity of space use for the locations of each state:

```
hmm_coupled_4st$plot_dist("restime") +  
  theme(legend.position = c(0.75, 0.8)) +  
  labs(  
    x = "Residence Time (days)",  
    title = "State-Dependent Residence Time Distributions"  
  ) +  
  scale_linetype_manual(  
    name = "States",  
    values = custom_lines,  
    labels = state_labels  
  ) +  
  scale_color_manual(  
    name = "States",  
    values = custom_colors,  
    labels = state_labels  
  ) +  
  theme_minimal(base_size = 16) +  
  theme(  
    plot.title = element_text(hjust = 0.5),  
    legend.title = element_text()  
  )
```

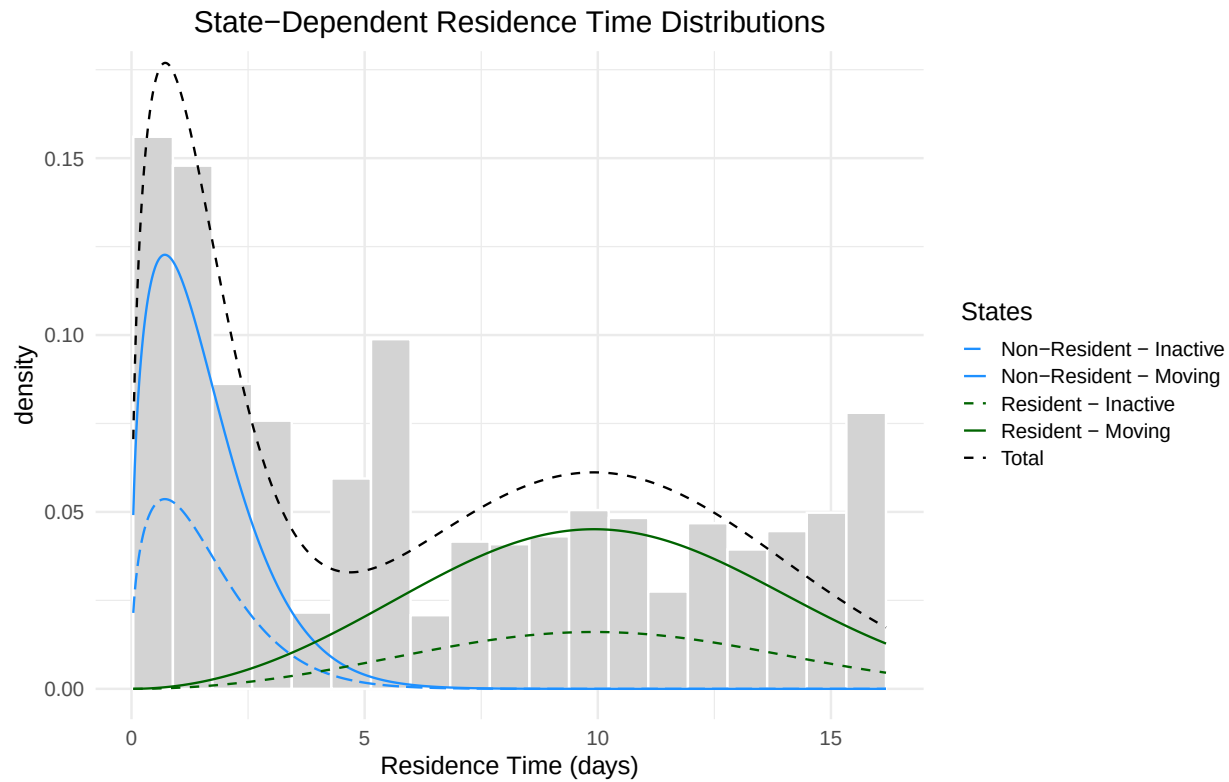


Figure 4: Estimated residence time distributions (Weibull). Resident states distributions concentrate at longer residence times; non-resident states show concentration at shorter durations. Both activity modes share the same residence time within each residency level.

7 Viterbi Algorithm for State Decoding

The Viterbi algorithm finds the most likely sequence of states given the observed data. This produces the “decoded” state sequence that can be mapped in space:

```
# Apply Viterbi algorithm to find most likely state sequence
data$state <- as.factor(hmm_coupled_4st$viterbi())

# Summarize state occurrences
print(table(data$state))
```

```
##
##  1  2  3  4
## 198 453 245 688
```

```
# Proportional distribution
print(round(prop.table(table(data$state)), 3))
```

```
##
##      1      2      3      4
## 0.125 0.286 0.155 0.434
```

8 Trajectory Visualization with Decoded States

Mapping the decoded states reveals activity modes and movement phases:

```
# Define state labels and visual properties
state_labels_decoded <- c(
  "1" = "Non-Resident - Inactive",
  "2" = "Non-Resident - Moving",
  "3" = "Resident - Inactive",
  "4" = "Resident - Moving"
)

custom_colors_decoded <- c(
  "1" = "darkblue",
  "2" = "#1E90FF",
  "3" = "darkgreen",
  "4" = "green"
)

custom_shapes_decoded <- c(
  "1" = 16, # circle
  "2" = 17, # triangle
  "3" = 16, # circle
  "4" = 17  # triangle
)

# Reorder factor levels for visual clarity (moving states overlap less with inactive)
data$state <- factor(data$state, levels = c("2", "4", "1", "3"))

# Create trajectory plot
ggplot(data, aes(x = x, y = y)) +
  geom_path(color = "grey70", size = 0.3, alpha = 0.6) +
  geom_point(
    data = data[order(as.numeric(data$state)), ],
    aes(x = x, y = y, color = state, shape = state),
```

```
    size = 2,
    alpha = 0.75
) +
scale_color_manual(
  values = custom_colors_decoded,
  labels = state_labels_decoded,
  name = "State"
) +
scale_shape_manual(
  values = custom_shapes_decoded,
  labels = state_labels_decoded,
  name = "State"
) +
coord_equal() +
labs(
  title = "Dummy Bear Trajectory with Decoded Movement States",
  x = "X Coordinate (m)",
  y = "Y Coordinate (m)"
) +
theme_minimal(base_size = 16) +
theme(
  plot.title = element_text(size = 16, hjust = 0.5),
  legend.title = element_text(),
  panel.grid.major = element_line(size = 0.2, color = "grey95")
)
```

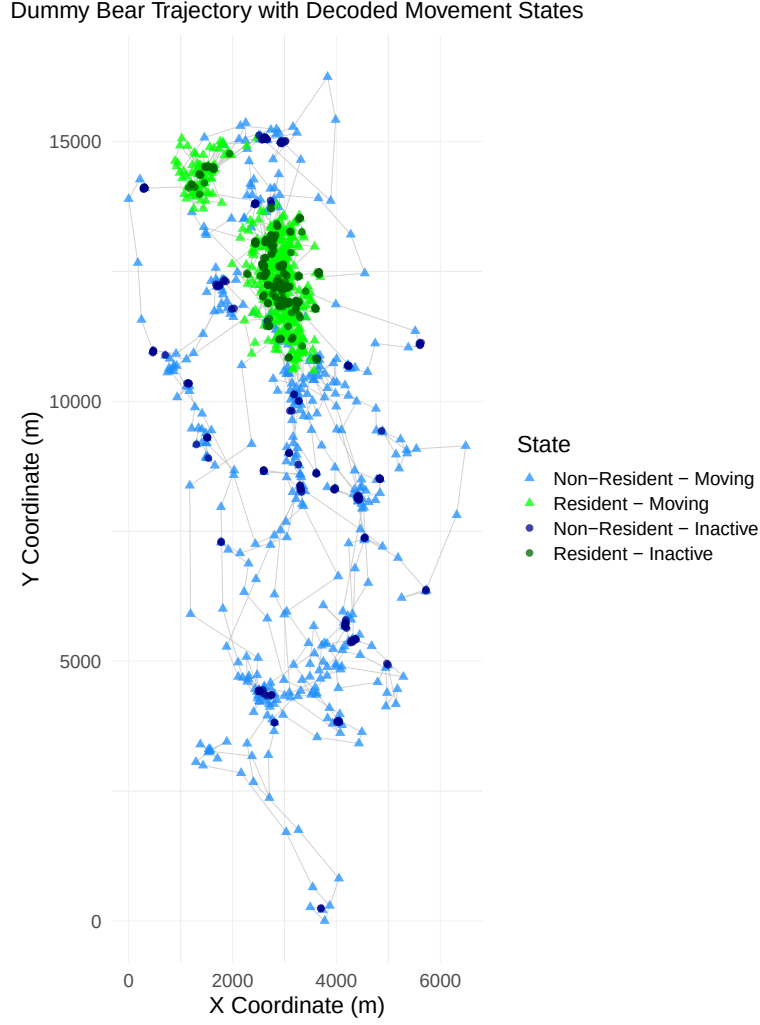


Figure 5: Bear trajectory with decoded movement states. Colors represent residency level (blue = Non-Resident, green = Fully Resident); shapes represent activity mode (circles = Inactive, triangles = Moving).

9 Home Ranges

We use the resulting resident states (decoded from the ACHMM model) to get precise home range estimations. We use kernel density estimation, and we compare the **home range** contours derived from:

- The unclassified bear trajectory, where we use the **90% level of the KDE** utilization distribution, estimated from **all GPS locations**.
- The classified bear trajectory, where we use the **99% level of the KDE** utilization distribution, estimated **only from resident locations**.

9.1 Home Range Estimation

```
# Create movement track object using amt package
trk <- make_track(data, .x = x, .y = y, .t = time, id = ID, state = state)

# Filter to resident locations only (States 3 and 4)
trk_resdata <- trk %>% filter(state %in% c(3, 4))

# Estimate KDE for resident locations
kde_resdata <- hr_kde(trk_resdata)

# Estimate KDE for all locations
kde_all <- hr_kde(trk)

# Extract isopleth contours at specified levels
kderes <- hr_isopleths(kde_resdata, levels = c(0.99)) # 99% for resident
kdeall <- hr_isopleths(kde_all, levels = c(0.9)) # 90% for all

# Prepare data for plotting
kde_combined <- bind_rows(
  kderes %>% mutate(layer = "kderes"),
  kdeall %>% mutate(layer = "kdeall")
)

kde_labels <- c(
  "kderes" = "99% KDE of Resident Locations",
  "kdeall" = "90% KDE of All Locations"
)
```

9.2 Home Range Visualization

```
# Plot trajectory with home range contours
ggplot(data, aes(x = x, y = y)) +
  geom_path(color = "grey70", size = 0.3, alpha = 0.6) +
  geom_point(
    data = data[order(as.numeric(data$state)), ],
    aes(x = x, y = y, color = state, shape = state),
    size = 1.5,
    alpha = 0.7
  ) +
  scale_color_manual(
```

```

    values = custom_colors_decoded,
    labels = state_labels_decoded,
    name = "Movement State"
  ) +
  scale_shape_manual(
    values = custom_shapes_decoded,
    labels = state_labels_decoded,
    name = "Movement State"
  ) +
  coord_equal() +
  geom_sf(
    data = kde_combined,
    aes(linetype = layer),
    inherit.aes = FALSE,
    fill = NA,
    linewidth = 0.7,
    color = "black"
  ) +
  scale_linetype_manual(
    name = "Home Range Contours",
    values = c("kderes" = "solid", "kdeall" = "dashed"),
    labels = kde_labels,
    guide = guide_legend(order = 2)
  ) +
  labs(
    title = "Home Range Estimations",
    x = "X Coordinate (m)",
    y = "Y Coordinate (m)"
  ) +
  theme_minimal(base_size = 16) +
  theme(
    plot.title = element_text(size = 16, hjust = 0.5),
    legend.title = element_text(),
    panel.grid.major = element_line(size = 0.2, color = "grey95")
  )

```

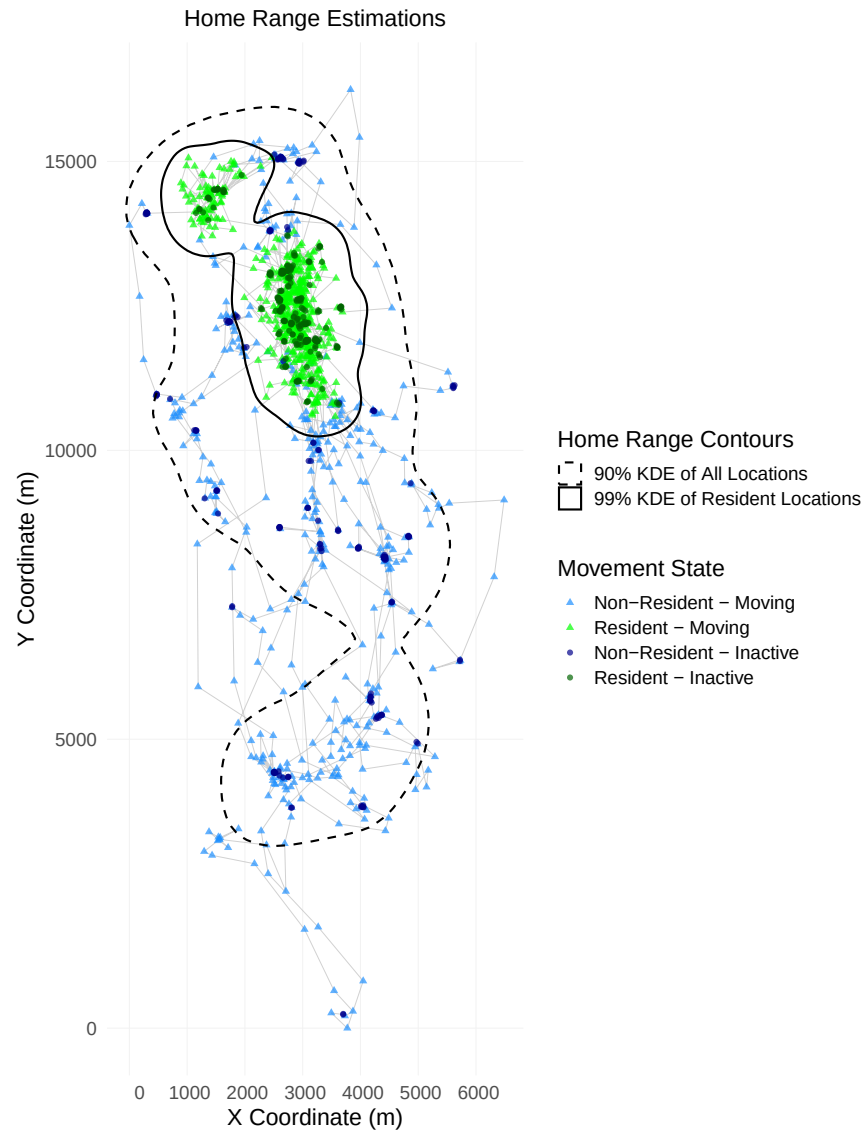


Figure 6: We observe that the 99% KDE of resident locations (States 3 and 4) successfully excludes non-resident locations (in blue) with low residence times (< 4 days); while the 90% KDE of all locations includes many of these non-resident locations inside the estimated home range.

10 Conclusion

This 4-state ACHMM framework:

- Allows the identification of activity modes (Inactive, Moving) within broader movement phases (Non-Resident, Resident) in GPS trajectories with regular sampling rates.
- Provides a movement informed approach to identifying home ranges.

11 Session Information

```
sessionInfo()
```

```
## R version 4.4.3 (2025-02-28)
## Platform: aarch64-apple-darwin20
## Running under: macOS 26.3
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib; LAPACK v
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: Asia/Bangkok
## tzcode source: internal
##
## attached base packages:
## [1] stats      graphics  grDevices  utils      datasets  methods   base
##
## other attached packages:
## [1] raster_3.6-32    sp_2.2-0        sf_1.0-22       amt_0.2.2.0
## [5] dplyr_1.1.4      lubridate_1.9.4  hmmTMB_1.0.2    ggplot2_4.0.0
## [9] TMB_1.9.17       mgcv_1.9-3      nlme_3.1-167    R6_2.6.1
## [13] momentuHMM_1.5.5
##
## loaded via a namespace (and not attached):
## [1] tidyselect_1.2.1  farver_2.1.2    loo_2.8.0
## [4] S7_0.2.0          fastmap_1.2.0   pracma_2.4.4
## [7] digest_0.6.38     timechange_0.3.0 lifecycle_1.0.4
## [10] StanHeaders_2.32.10 survival_3.8-3   terra_1.8-80
## [13] magrittr_2.0.4    compiler_4.4.3  CircStats_0.2-6
## [16] rlang_1.1.6       rngtools_1.5.2  tools_4.4.3
## [19] yaml_2.3.10       knitr_1.50      labeling_0.4.3
## [22] doRNG_1.8.6.2     pkgbuild_1.4.8  classInt_0.4-11
## [25] RColorBrewer_1.1-3 KernSmooth_2.23-26 withr_3.0.2
## [28] purrr_1.2.0       numDeriv_2016.8-1.1 grid_4.4.3
## [31] stats4_4.4.3      e1071_1.7-16    colorspace_2.1-1
## [34] inline_0.3.21     scales_1.4.0    iterators_1.0.14
## [37] MASS_7.3-64       cli_3.6.5       mvtnorm_1.3-3
## [40] rmarkdown_2.30    generics_0.1.4  RcppParallel_5.1.10
## [43] rstudioapi_0.17.1 proxy_0.4-27     DBI_1.2.3
```

```
## [46] rstan_2.32.7      stringr_1.6.0      tmbstan_1.0.91
## [49] splines_4.4.3      parallel_4.4.3      matrixStats_1.5.0
## [52] vctrs_0.6.5        boot_1.3-31         Matrix_1.7-2
## [55] foreach_1.5.2      optimx_2025-4.9     tidyr_1.3.1
## [58] units_1.0-0        crawl_2.3.0         glue_1.8.0
## [61] nloptr_2.2.1       codetools_0.2-20    stringi_1.8.7
## [64] gtable_0.3.6       QuickJSR_1.7.0      tibble_3.3.0
## [67] pillar_1.11.1      htmltools_0.5.8.1   Brodningnag_1.2-9
## [70] Rdpack_2.6.4       doParallel_1.0.17   evaluate_1.0.5
## [73] lattice_0.22-6     backports_1.5.0     rbibutils_2.3
## [76] class_7.3-23       Rcpp_1.1.0          checkmate_2.3.2
## [79] gridExtra_2.3      xfun_0.56           pkgconfig_2.0.3
```