Previous space use density parameter estimation

Scott Forrest

2024-03-04

This script estimates the most likely parameters for spatial (using Kernel Density Estimation - KDE) memory with a temporal decay (negative exponential) component. Firstly we use the `amt` package to estimate a KDE bandwidth (which is the standard deviation when using Gaussian kernels), using the 'reference' bandwidth. To get a population-level estimate for fitting a hierarchical model we use the mean between individuals. For the temporal decay component, we are trying to estimate a negative exponential rate parameter that reduces the influence of previous locations the further they are in the past. There is a function to estimate a temporal decay value for some given parameters, which can be optimised using maximum likelihood for each individual animal, and then a function to estimate a population-level temporal decay parameter. After estimating the temporal decay parameter(s), there is a function to estimate the previous space use density for all used and randomly sampled steps, using the estimated KDE bandwidth and temporal decay parameter(s), which is used in the step selection model fitting.

Load packages

```
options(scipen=999)

library(tidyverse)
packages <- c("amt", "terra", "tictoc", "matrixStats", "beepr", "ks", "viridis")
walk(packages, require, character.only = T)
```

Import buffalo data

These data include the random steps and covariates. The random steps are included in this dataset so that the spatiotemporal memory density can be estimated at every used and random step, but they are not used to estimate the bandwidth or temporal decay parameters.

```
buffalo_data_rand_steps <- read_csv(
  "outputs/buffalo_parametric_popn_covs_GvM_10rs_2024-02-18.csv")
```

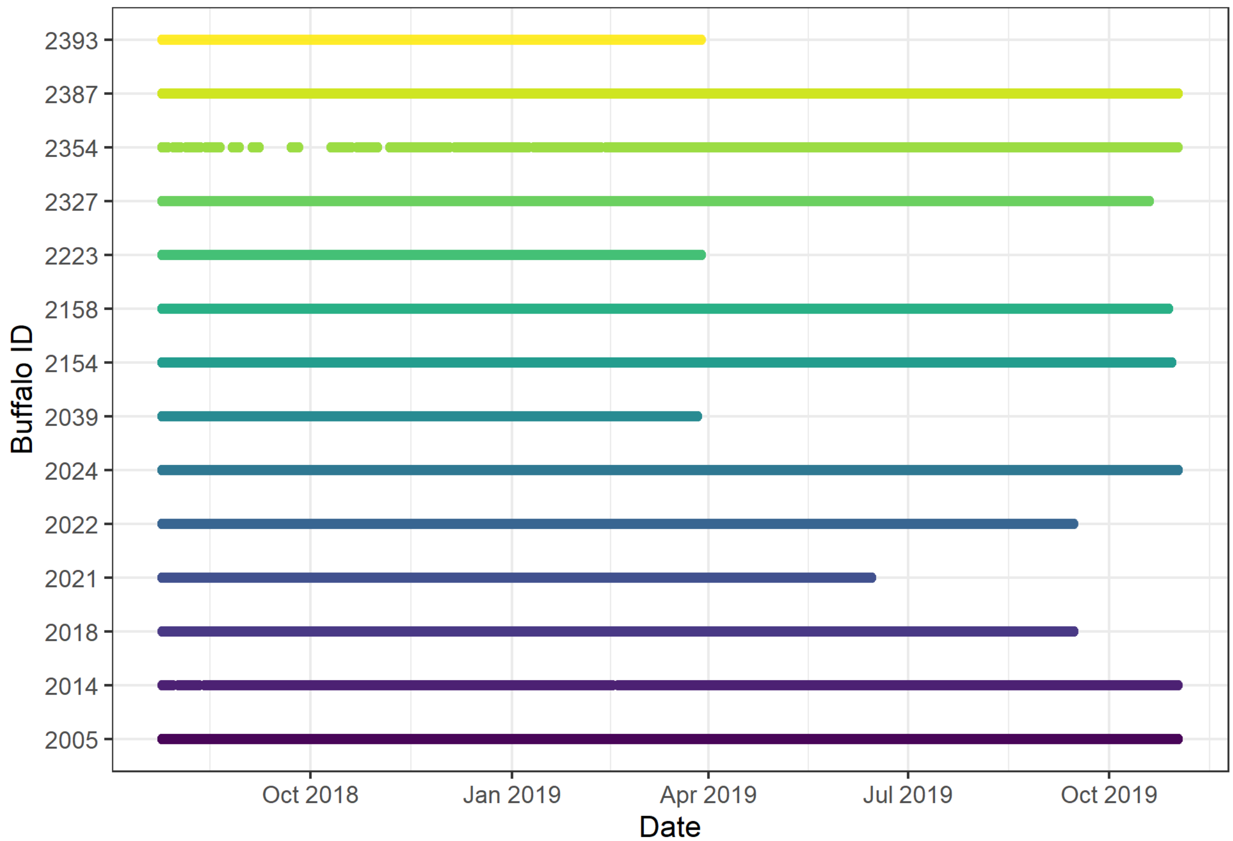
```
## Rows: 1165406 Columns: 22
## -- Column specification ------------------------------------------------------------------------
## Delimiter: ","
## dbl  (18): id, burst_, x1_, x2_, y1_, y2_, sl_, ta_, dt_, hour_t2, step_id_, y, ndvi_temporal, veg_h
## lgl   (1): case_
## dttm  (3): t1_, t2_, t2_rounded
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# to ensure that the time is in the right timezone
attr(buffalo_data_rand_steps$t1_, "tzone") <- "Australia/Queensland"
```

```r
attr(buffalo_data_rand_steps$t2_, "tzone") <- "Australia/Queensland"
attr(buffalo_data_rand_steps$t2_rounded, "tzone") <- "Australia/Queensland"
```

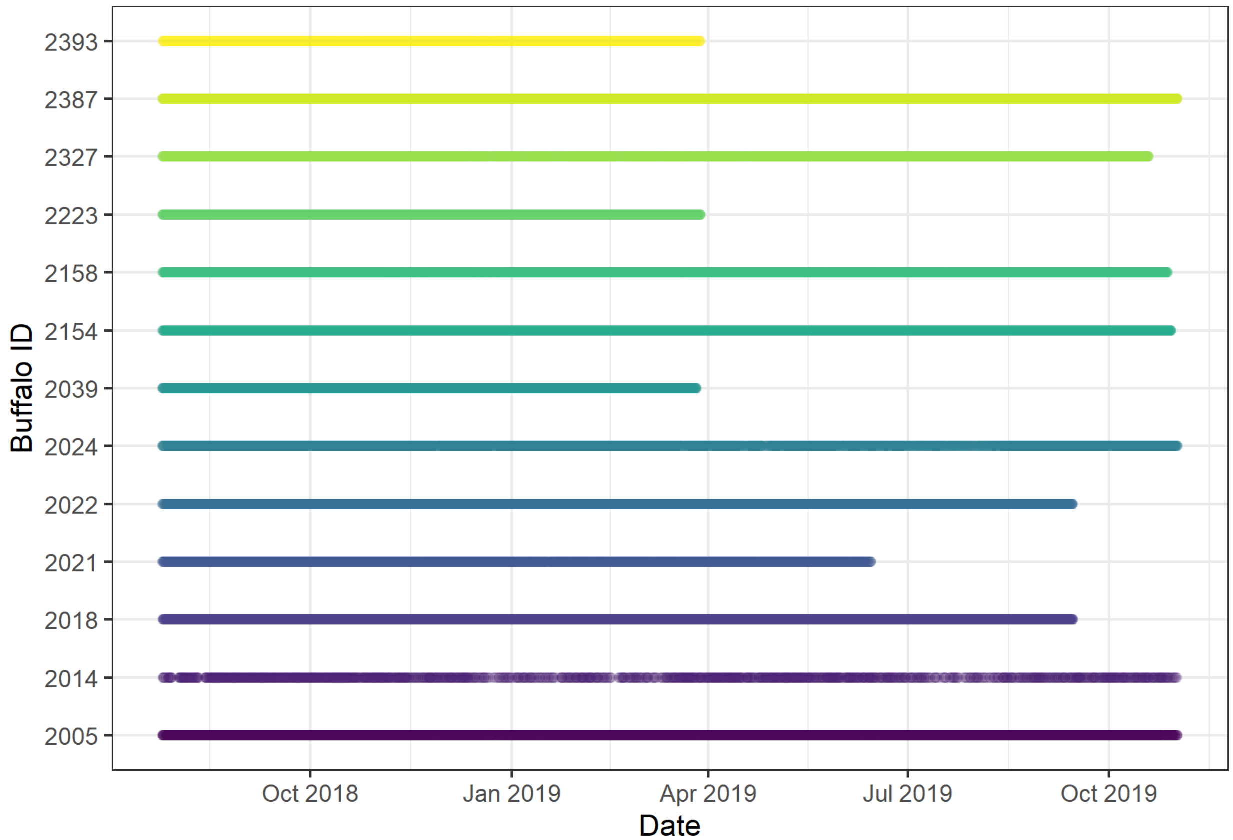
Check the GPS data through time to ensure that there are no large gaps in the data, and to identify individuals that have poor data quality.

```r
# check the timeline of GPS data across individuals
buffalo_data_rand_steps %>% ggplot(aes(x = t1_, y = factor(id),
                                       colour = factor(id))) +
  geom_point(alpha = 0.1) +
  scale_y_discrete("Buffalo ID") +
  scale_x_datetime("Date") +
  scale_colour_viridis_d() +
  theme_bw() +
  theme(legend.position = "none")
```




```r
# individuals to keep - remove 2354 due to poor data quality
# in the first couple months
buffalo_ids <- c(2005, 2014, 2018, 2021, 2022, 2024, 2039,
                 2154, 2158, 2223, 2327, 2387, 2393) # 2354,
# remove individuals not in the id vector
buffalo_data_rand_steps <- buffalo_data_rand_steps %>%
  filter(id %in% buffalo_ids)

buffalo_data_rand_steps %>% filter(y == 1) %>%
  ggplot(aes(x = t1_, y = factor(id), colour = factor(id))) +
```

```r
  geom_point(alpha = 0.05) +
  scale_y_discrete("Buffalo ID") +
  scale_x_datetime("Date") +
  scale_colour_viridis_d() +
  theme_bw() +
  theme(legend.position = "none")
```




```r
ggsave(paste0("outputs/plots/manuscript_figs/GPS_timeline_no2354_",
              Sys.Date(), ".png"),
       width=150, height=90, units="mm", dpi = 600)
```

Preparing data for KDE estimation

```r
# convert to track object to use the amt package for KDE estimation
buffalo_data_pres_track <- buffalo_data_rand_steps %>%
  filter(y == 1 & id %in% buffalo_ids) %>%
  mk_track(id = id, x1_, y1_, t1_, order_by_ts = T, all_cols = T, crs = 3112) %>%
  arrange(id)

head(buffalo_data_pres_track)
```

```
## # A tibble: 6 x 22
##        x_        y_  t_                          id burst_    x2_        y2_   sl_     ta_  t2_
## *   <dbl>     <dbl> <dttm>                    <dbl>  <dbl>  <dbl>      <dbl> <dbl>   <dbl> <dttm>
## 1 41968. -1435673. 2018-07-25 11:04:23  2005       1 41921. -1435656.  50.7  1.37    2018-07-25 12:04:2
```

```
## 2 41921. -1435656. 2018-07-25 12:04:39  2005      1 41778. -1435602. 152.  -0.0214 2018-07-25 13:04:1
## 3 41778. -1435602. 2018-07-25 13:04:17  2005      1 41840. -1435637.  70.7  2.99   2018-07-25 14:04:3
## 4 41840. -1435637. 2018-07-25 14:04:39  2005      1 41655. -1435606. 188.  -2.80   2018-07-25 15:04:2
## 5 41655. -1435606. 2018-07-25 15:04:27  2005      1 41618. -1435610.  37.1  0.285  2018-07-25 16:04:2
## 6 41618. -1435610. 2018-07-25 16:04:24  2005      1 41687. -1436127. 522.   1.58   2018-07-25 17:04:2
## # i 11 more variables: t2_rounded <dttm>, hour_t2 <dbl>, case_ <lgl>, step_id_ <dbl>, y <dbl>, ndvi_t
## #   veg_herby <dbl>, canopy_cover <dbl>, slope <dbl>, cos_ta_ <dbl>, log_sl_ <dbl>
```

Estimate the KDE kernel bandwidth

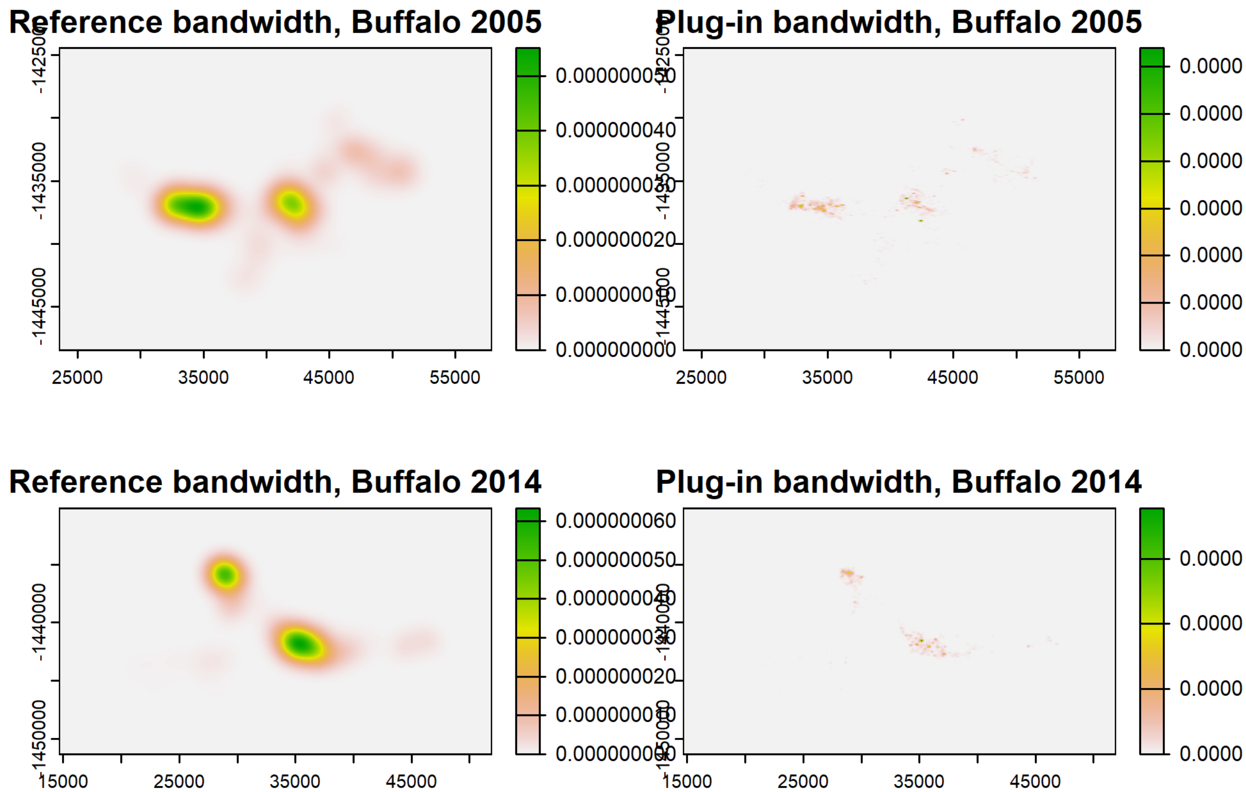
Initially we were estimating the kernel bandwidth (sd parameter) and the temporal decay component concurrently, based on the parameters that maximised the next step density. This is an interesting approach, and may be useful for inferring the 'strength' of memory, and how that differs between individuals, but as there is an additional inference process - the step selection model fitting, we thought it would be best to keep the procedure for the estimating the bandwidth simple. Here we assess the kernel bandwidth estimated by several methods, which is constrained to be the same in the x and the y direction, features of landscape can produce asymmetry, although we did not want to impose asymmetry when predicting in novel areas.

We are using KDE rather than a method that considers autocorrelation, such as AKDE, as when generating simulated trajectories, the density needs to be updated at every time step, and evaluated at each proposed step. Calculating densities using KDE with normal kernels is very fast, and is straightforward to include in the simulation model, as we can use the dnorm function with a vector of x (and y) locations, rather than estimating the AKDE (or similar) and incorporating as a spatial layer.

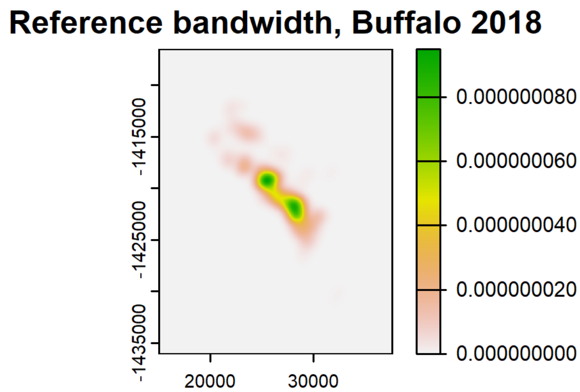
```
# change plotting to display four base plots at once
par(mfrow = c(2, 2))

buffer <- 5000
res <- 25

bandwidth_ref <- vector(mode = "list", length = length(buffalo_ids))
bandwidth_ref_vector <- c()
bandwidth_ref_hr <- vector(mode = "list", length = length(buffalo_ids))

bandwidth_pi <- vector(mode = "list", length = length(buffalo_ids))
bandwidth_pi_vector <- c()
bandwidth_pi_hr <- vector(mode = "list", length = length(buffalo_ids))

# bandwidth_lscv <- vector(mode = "list", length = length(buffalo_ids))
# bandwidth_lscv_vector <- c()
# bandwidth_lscv_hr <- vector(mode = "list", length = length(buffalo_ids))


tic(msg = "Total time for 13 individuals")

for(i in 1:length(buffalo_ids)) {

  # subset by buffalo id
  buffalo_data_id <- buffalo_data_pres_track %>%
    filter(y == 1 & id == buffalo_ids[i])

  # create template raster
  # create extent of the raster
  xmin <- min(buffalo_data_id$x2_) - buffer
```

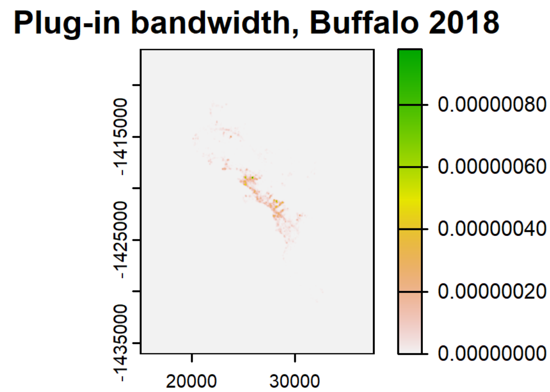
```r
  xmax <- max(buffalo_data_id$x2_) + buffer
  ymin <- min(buffalo_data_id$y2_) - buffer
  ymax <- max(buffalo_data_id$y2_) + buffer
  template_raster <- rast(xmin = xmin, xmax = xmax,
                          ymin = ymin, ymax = ymax,
                          res = res, crs = crs("epsg:3112"))


  # reference bandwidth
  tic(msg = "Reference bandwidth")
  bandwidth_ref[[i]] <- hr_kde_ref(buffalo_data_id)
  toc()
  print(bandwidth_ref[[i]])
  bandwidth_ref_vector[i] <- bandwidth_ref[[i]][1]


  bandwidth_ref_hr[[i]] <- hr_kde(buffalo_data_id,
                           h = bandwidth_ref[[i]],
                           trast = template_raster,
                           levels = c(0.5, 0.75, 0.95))
  plot(bandwidth_ref_hr[[i]]$ud, main = paste0("Reference bandwidth, Buffalo ",
                                                buffalo_ids[i]))


  # plug-in bandwidth
  tic(msg = "Plug-in bandwidth")
  bandwidth_pi[[i]] <- hr_kde_pi(buffalo_data_id)
  toc()
  print(bandwidth_pi[[i]])
  bandwidth_pi_vector[i] <- bandwidth_pi[[i]][1]

  bandwidth_pi_hr[[i]] <- hr_kde(buffalo_data_id,
                           h = bandwidth_pi[[i]],
                           trast = template_raster, levels = c(0.5, 0.75, 0.95))
  plot(bandwidth_pi_hr[[i]]$ud, main = paste0("Plug-in bandwidth, Buffalo ",
                                               buffalo_ids[i]))


  # least-squares cross validation bandwidth
  # estimate bandwidth using least-squares cross validation
  # tic(msg = "LSCV bandwidth")
  # bandwidth_lscv[[i]] <- hr_kde_lscv(buffalo_data_id, trast = template_raster,
  #                                    which_min = "local")
  # toc()
  # bandwidth_lscv_vector[i] <- bandwidth_lscv[[i]][1]
  #
  # bandwidth_lscv_hr[[i]] <- hr_kde(buffalo_data_id,
  #                           h = bandwidth_lscv[[i]],
  #                           trast = template_raster, levels = c(0.5, 0.75, 0.95))
  # plot(bandwidth_lscv_hr[[i]]$ud, main = paste0("LSCV bandwidth, Buffalo ",
  #                                               buffalo_ids[i]))

}
```

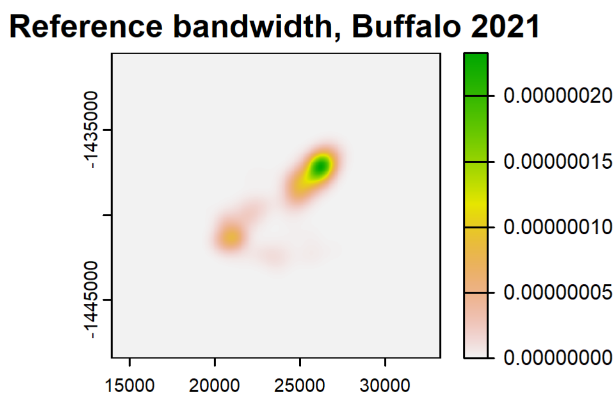
```
## Reference bandwidth: 0 sec elapsed
## [1] 845.9741 845.9741

## Plug-in bandwidth: 0.02 sec elapsed
## [1] 85.79459 25.29613

## Reference bandwidth: 0 sec elapsed
## [1] 870.951 870.951

## Plug-in bandwidth: 0.02 sec elapsed
## [1] 80.31529 57.90325
```

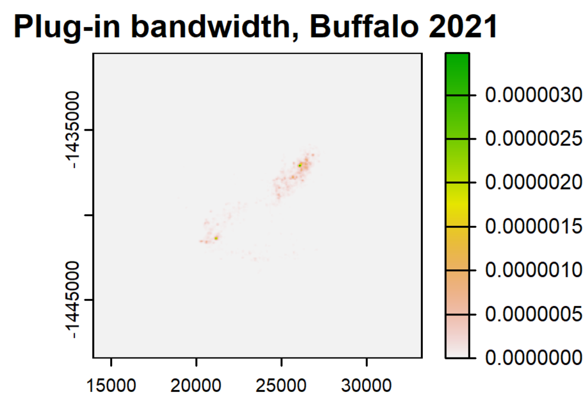



```
## Reference bandwidth: 0 sec elapsed
## [1] 514.6824 514.6824

## Plug-in bandwidth: 0 sec elapsed
## [1] 51.91688 54.90821

## Reference bandwidth: 0 sec elapsed
## [1] 451.5416 451.5416

## Plug-in bandwidth: 0 sec elapsed
## [1] 46.21448 43.85092
```







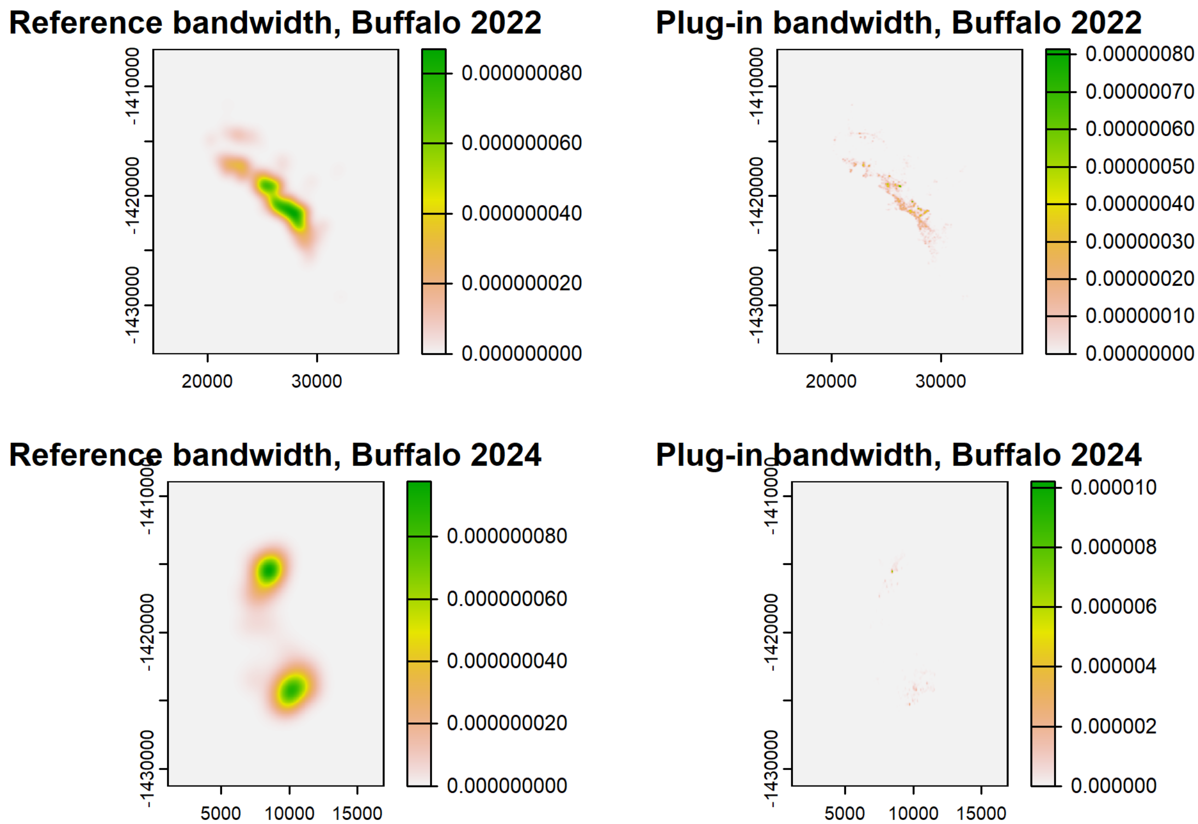



```
## Reference bandwidth: 0 sec elapsed
## [1] 490.8895 490.8895
```

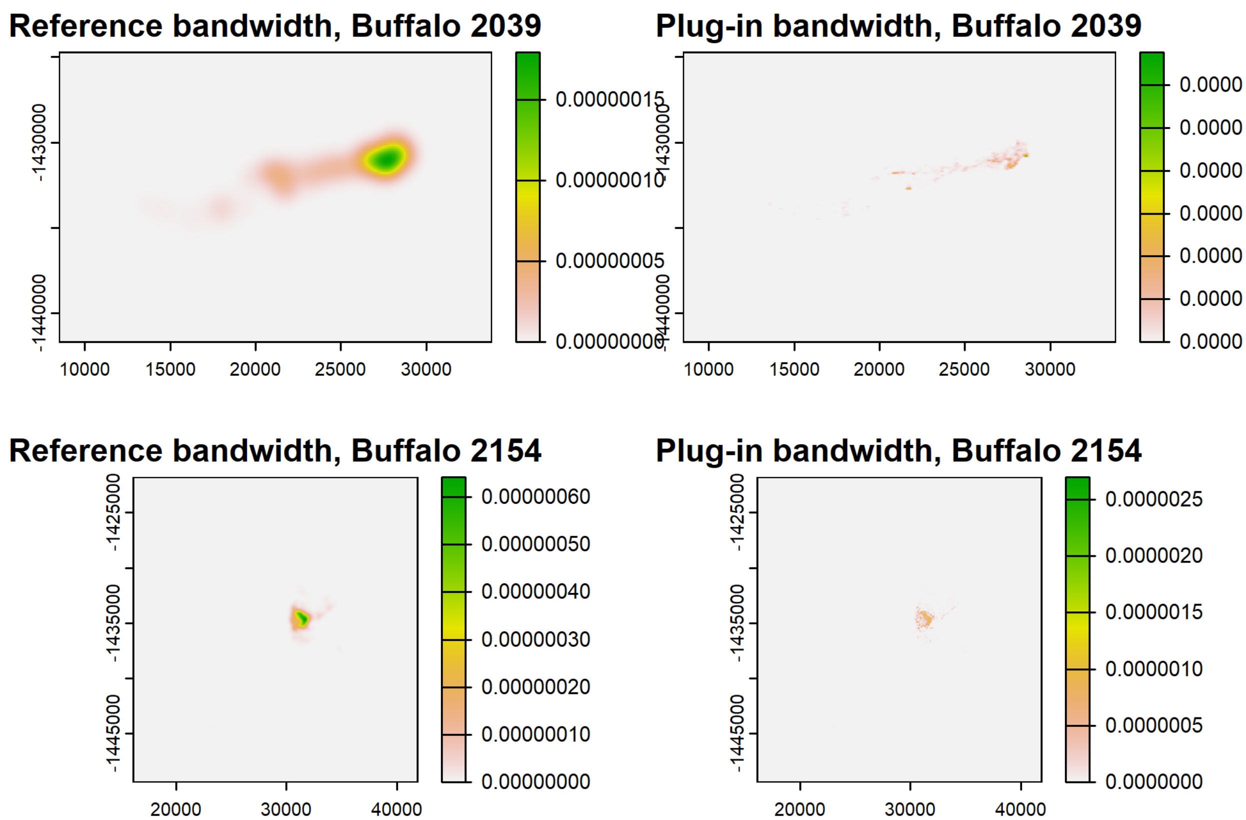
```
## Plug-in bandwidth: 0.01 sec elapsed
## [1] 58.24006 49.54056
```

```
## Reference bandwidth: 0 sec elapsed
## [1] 677.4343 677.4343
```

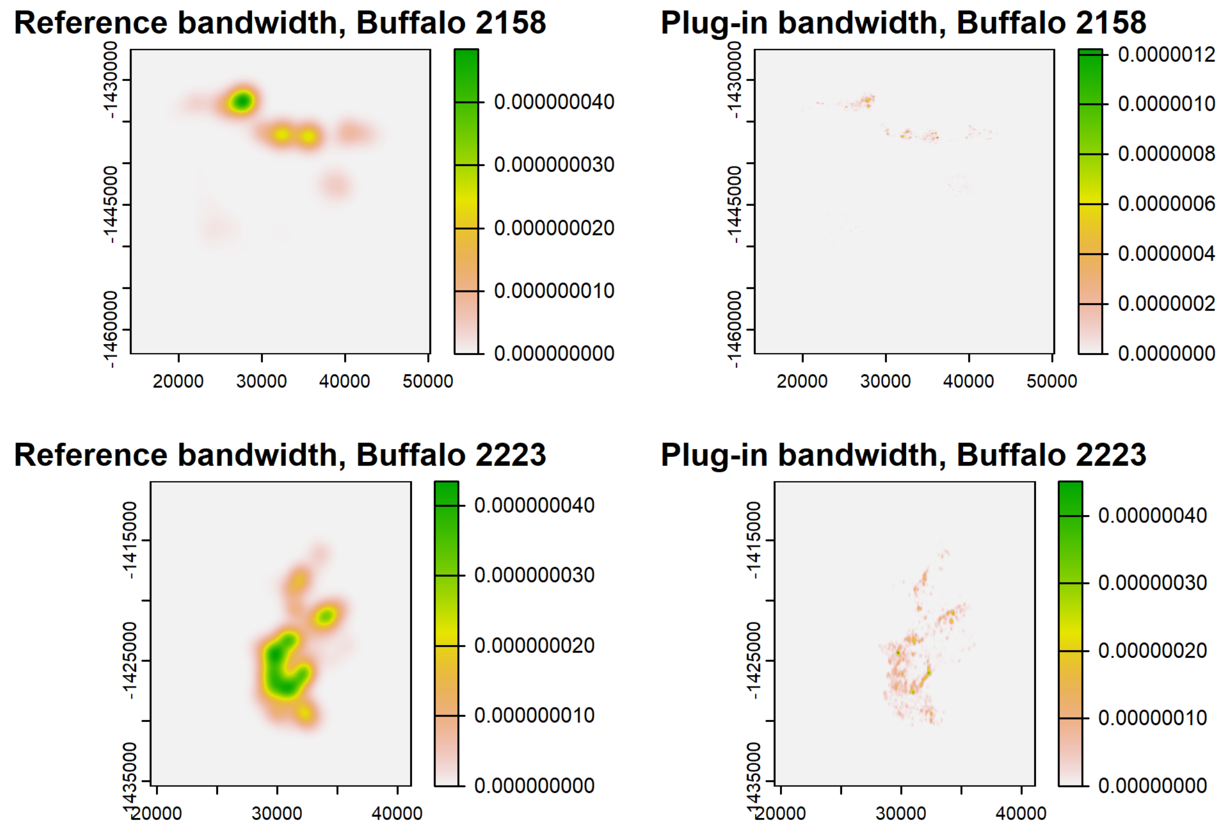
```
## Plug-in bandwidth: 0.02 sec elapsed
## [1] 25.01523 60.02277
```



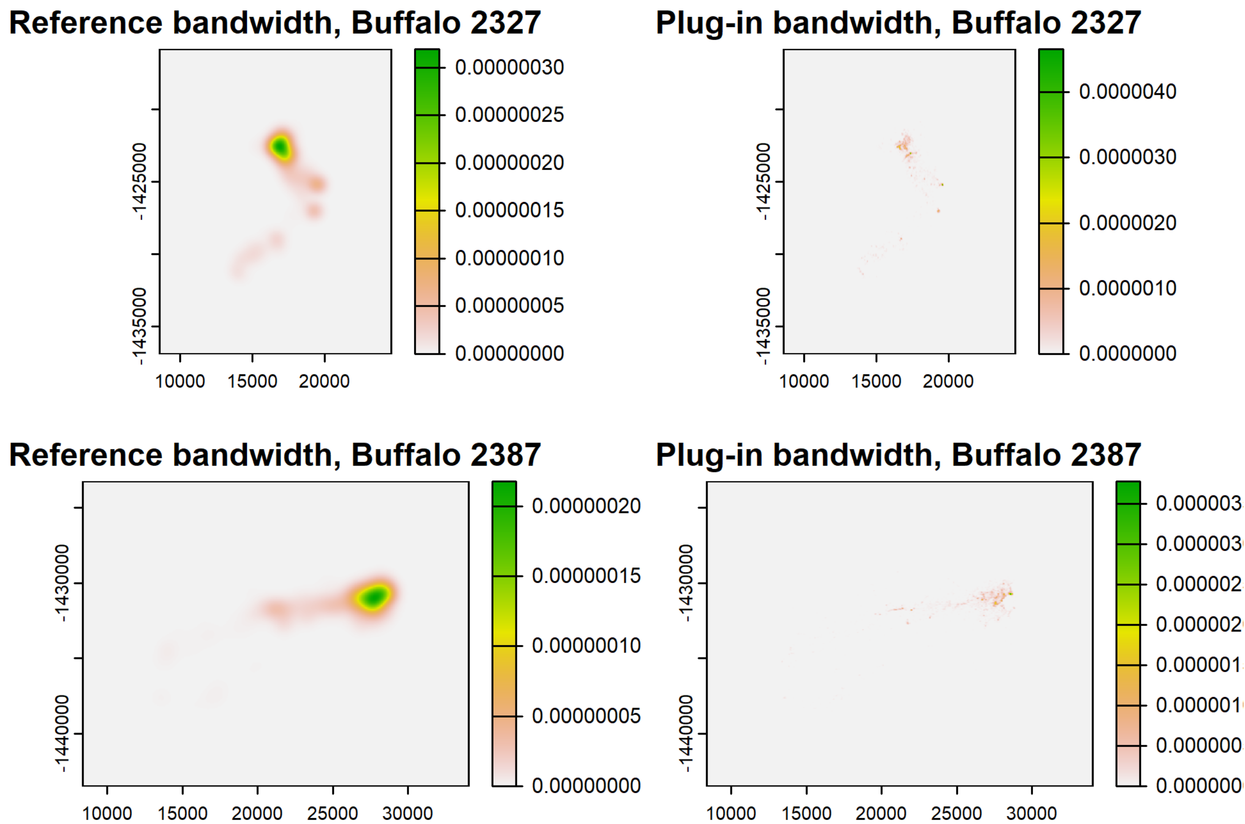

```
## Reference bandwidth: 0 sec elapsed
## [1] 547.5389 547.5389

## Plug-in bandwidth: 0.02 sec elapsed
## [1] 65.66955 27.37679

## Reference bandwidth: 0 sec elapsed
## [1] 172.9515 172.9515

## Plug-in bandwidth: 0 sec elapsed
## [1] 24.84565 25.78731
```




```
## Reference bandwidth: 0 sec elapsed
## [1] 957.0183 957.0183

## Plug-in bandwidth: 0.02 sec elapsed
## [1] 76.14371 48.52842

## Reference bandwidth: 0 sec elapsed
## [1] 601.7503 601.7503

## Plug-in bandwidth: 0 sec elapsed
## [1] 58.74228 98.44715
```



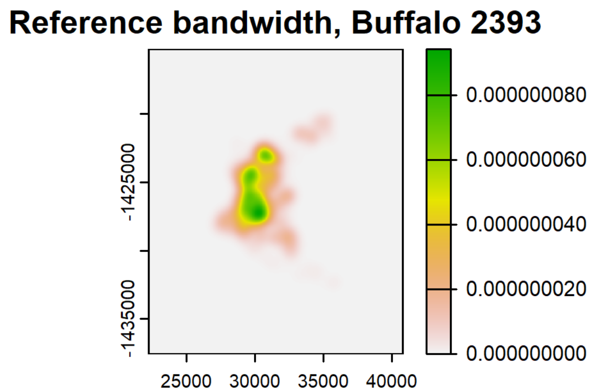

```
## Reference bandwidth: 0 sec elapsed
## [1] 406.984 406.984

## Plug-in bandwidth: 0.01 sec elapsed
## [1] 23.48089 38.85835

## Reference bandwidth: 0 sec elapsed
## [1] 473.602 473.602

## Plug-in bandwidth: 0.02 sec elapsed
## [1] 38.86717 20.80289
```

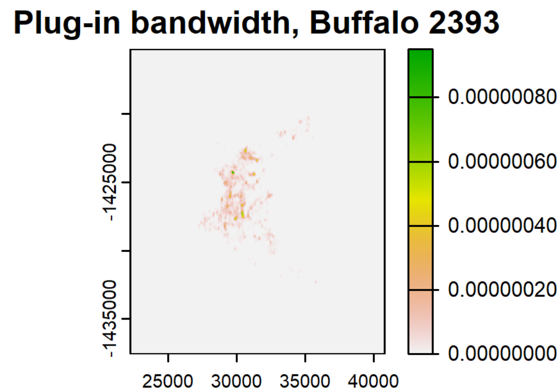



```
## Reference bandwidth: 0 sec elapsed
## [1] 420.2645 420.2645

## Plug-in bandwidth: 0 sec elapsed
## [1] 42.22782 73.21382
```

```
toc()
```

```
## Total time for 13 individuals: 60.75 sec elapsed
```

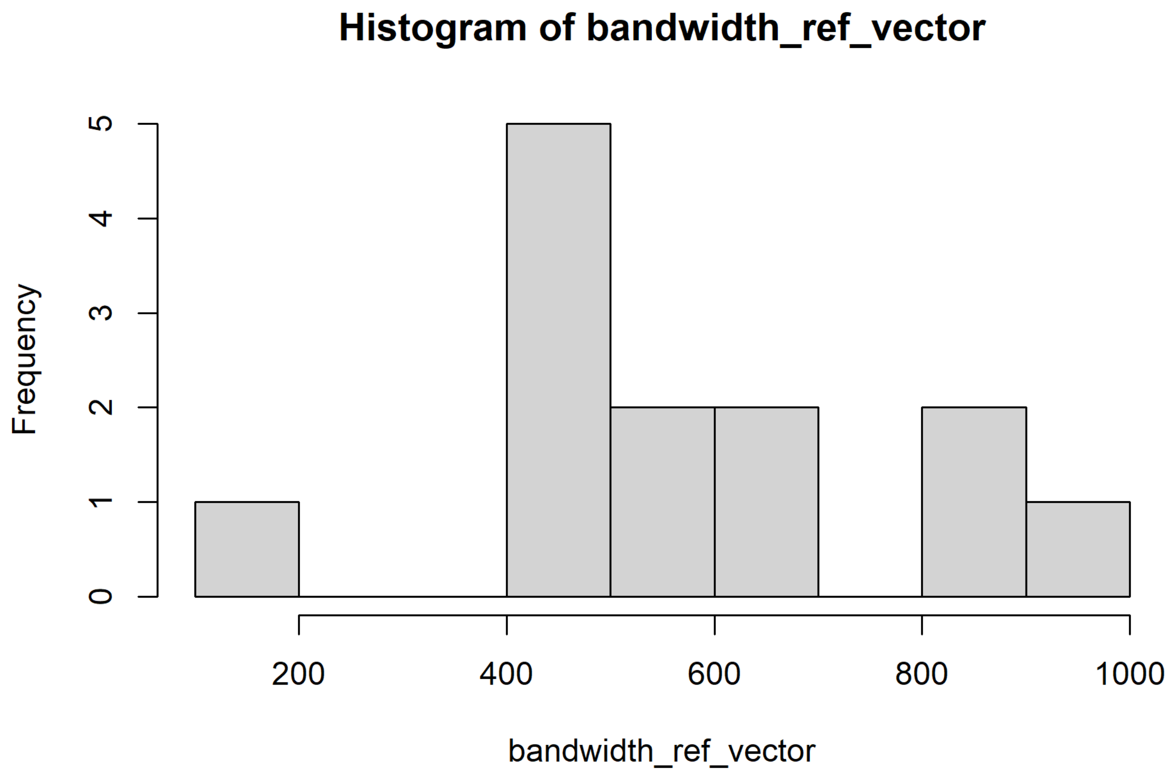




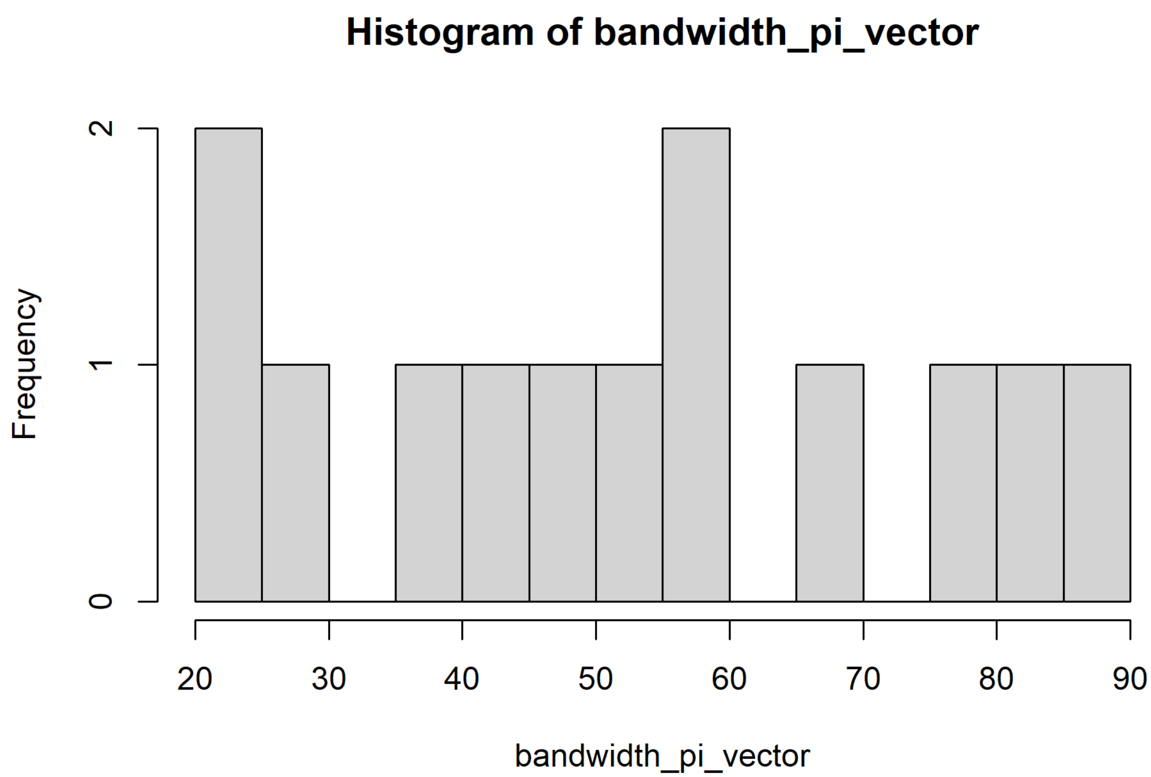
Use the mean bandwidth for population level analysis

It's clear that the bandwidths estimated by the different methods are very different. As we are predicting from these models, we opted for the reference bandwidth that undersmooths the space use, describing a broad-familiarity with an area, rather than the plug-in bandwidth that has many small discrete modes that conflate with the habitat in these areas. We did not run simulations with the plug-in bandwidth to test our assumptions however.

```
hist(bandwidth_ref_vector, breaks = 10)
```




```r
hist(bandwidth_pi_vector, breaks = 10)
```



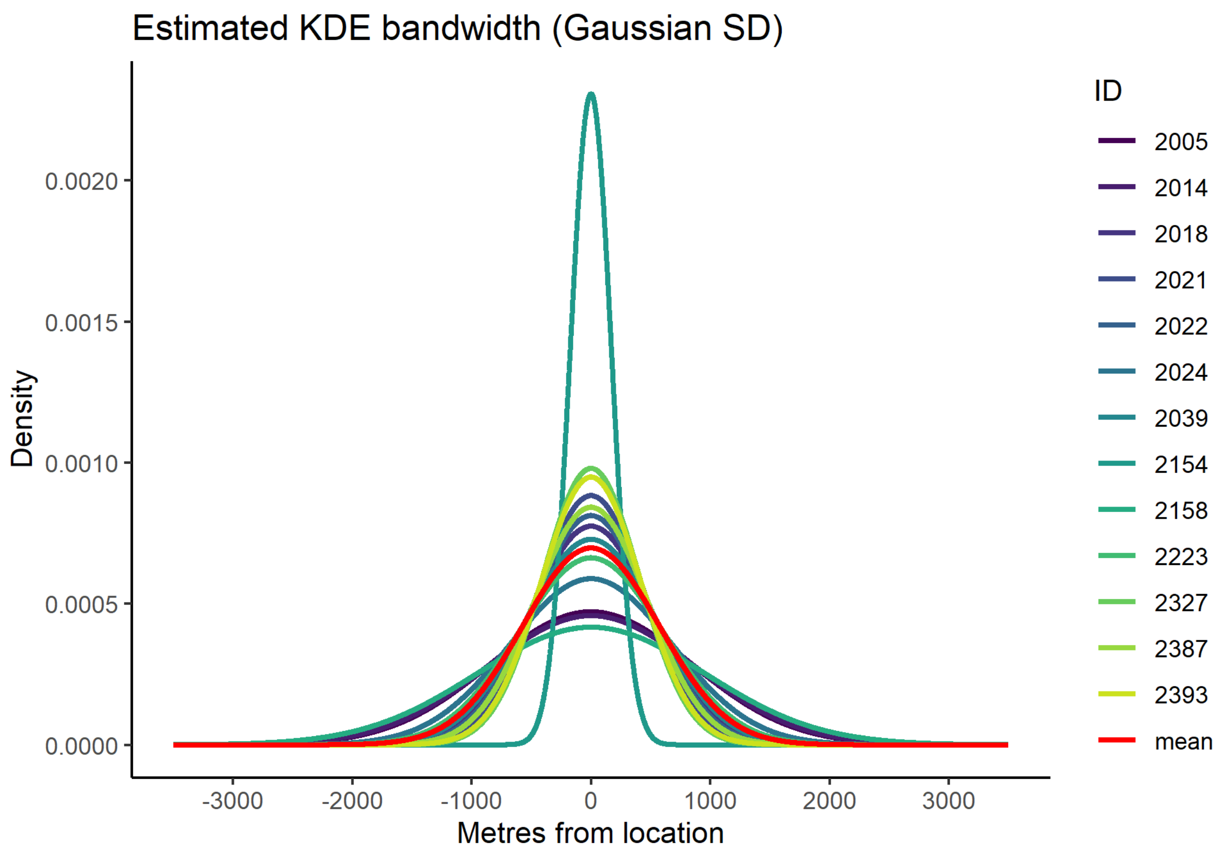

```
mean_kde_sd <- mean(bandwidth_ref_vector)
mean_kde_sd
```

```
## [1] 571.6602
```

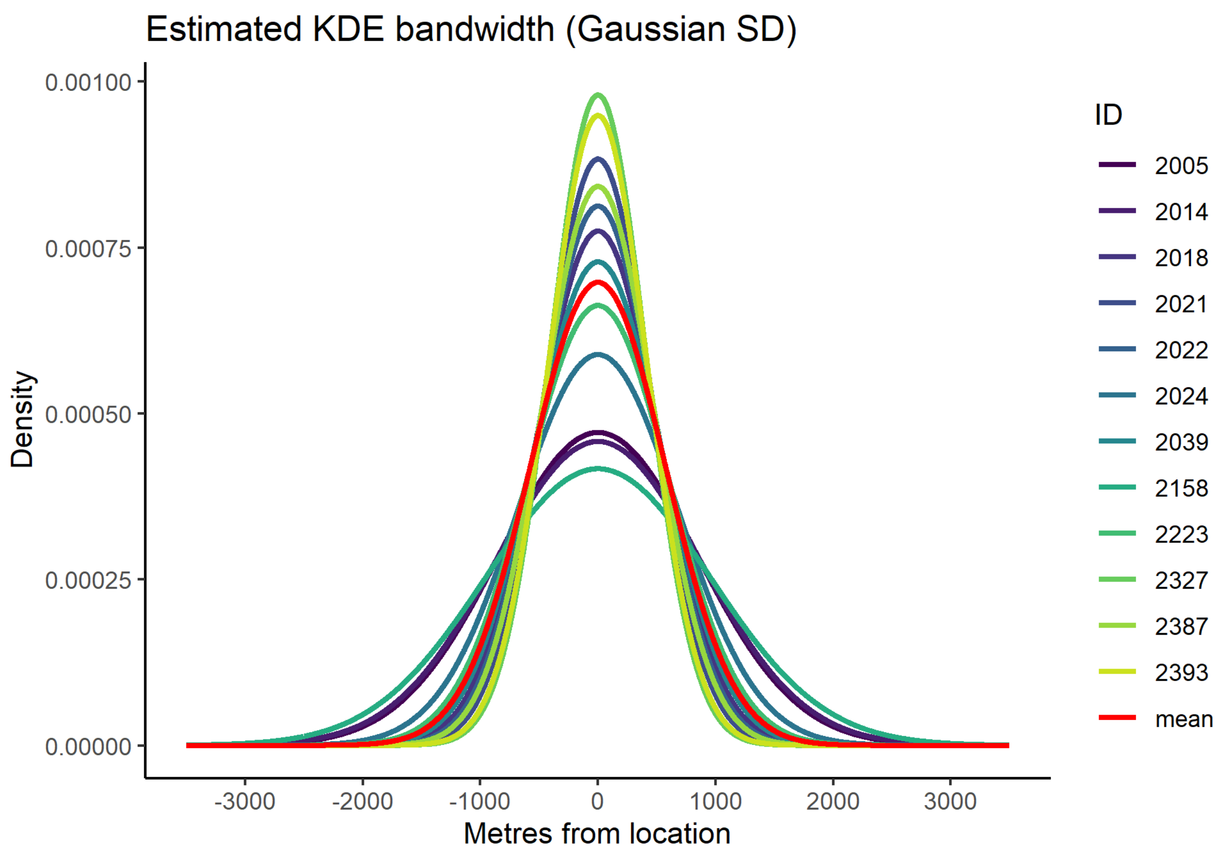
Plotting the estimated spatial sd parameters

```
range <- 3500 # for the x-axis - tune as needed

spatial_sds <- data.frame(-range:range,
                          sapply(c(bandwidth_ref_vector, mean_kde_sd),
                                 function(sd) dnorm(-range:range, mean = 0, sd = sd)))

colnames(spatial_sds) <- c("x", buffalo_ids, "mean")

spatial_sds_long <- pivot_longer(spatial_sds, cols = !1,
                                 names_to = "id", values_to = "value")

# Create color mapping
unique_groups <- unique(spatial_sds_long$id)
colors <- viridis(length(unique_groups))
names(colors) <- unique_groups
colors["mean"] <- "red"
```

```r
ggplot(spatial_sds_long) +
  geom_line(aes(x = -x, y = value, colour = id), size = 1) +
  scale_x_continuous("Metres from location", breaks = seq(-3000, 3000, 1000)) +
  scale_y_continuous("Density") +
  # scale_colour_viridis_d("ID") +
  scale_colour_manual("ID", values = colors) +
  ggtitle("Estimated KDE bandwidth (Gaussian SD)") +
  theme_classic() +
  theme(legend.position = "right")
```




```r
ggplot(spatial_sds_long %>% filter(!id == 2154)) +
  geom_line(aes(x = -x, y = value, colour = id), size = 1) +
  scale_x_continuous("Metres from location", breaks = seq(-3000, 3000, 1000)) +
  scale_y_continuous("Density") +
  # scale_colour_viridis_d("ID") +
  scale_colour_manual("ID", values = colors) +
  ggtitle("Estimated KDE bandwidth (Gaussian SD)") +
  theme_classic() +
  theme(legend.position = "right")
```




```r
# ggsave(paste0("outputs/plots/manuscript_figs/spatial_KDE_sd_byID_",
#               Sys.Date(), ".png"),
#        width=150, height=90, units="mm", dpi = 600)
```

Estimating the temporal decay component

To incorporate decaying memory into model fitting and predictions, we used a temporally decaying intensity of previous space use approach by combining kernel density estimation with weights that exponentially decay the further they are in the past, representing a gradual forgetting process. For our approach, the density f at the current location s and time t is defined by

$$f(s|t) = \frac{\sum_{i=1}^{n} \exp^{-\gamma(t-t_i)} K_h(x - x_i) K_h(y - y_i)}{\sum_{i=1}^{n} \exp^{-\gamma(t-t_i)}}.$$

where γ defines the strength of temporal decay, t_i is the time of previous locations, K_h is the kernel function $\sim \mathcal{N}(\mu = 0, \sigma)$, where the x distance is determined by the current location x and the previous locations x_i, which is the same in the y direction. For numerical stability, we used the log-sum-exp trick to sum over the densities relating to each previous location in the memory period. Similarly to Rheault2021-od, we excluded locations from the past 24 hours, as these locations reflect the autocorrelation in the movement process rather than a memory process.

Subsetting the memory period by the number of hours

```
density_space_time_hours_subset <- function(
    locations_x,
    locations_y,
    locations_time,
    spatial_sd,
    gamma_param,
    # the memory period should cover the
    #duration that the memory decays to nearly 0
    memory_period_hours, # in hours
    # number of hours to exclude
    memory_delay_hours
     ) {

  # create object for log_likelihood
  log_likelihood <- 0

  # start from the first location AFTER a duration of the memory period,
  # to subset the previous locations
  for(i in 1:(length(locations_x)-(i+memory_period_hours))) {

    # extract the current location and time
    location_current_x <- locations_x[i+memory_period_hours]
    location_current_y <- locations_y[i+memory_period_hours]
    location_current_time <- locations_time[i+memory_period_hours]

    # calculate the start and end times for the memory period
    #(from the start of the memory period until the memory delay begins)
    memory_start_time <- location_current_time - as.difftime(memory_period_hours,
                                                            units = "hours")
    memory_end_time <- location_current_time - as.difftime(memory_delay_hours,
                                                          units = "hours")

    # subset the x, y, and time vectors by the
    # locations BETWEEN the memory period and memory delay
    memory_period_x <- locations_x[locations_time >= memory_start_time &
                                     locations_time <= memory_end_time]

    memory_period_y <- locations_y[locations_time >= memory_start_time &
                                     locations_time <= memory_end_time]

    memory_period_time <- locations_time[locations_time >= memory_start_time &
                                           locations_time <= memory_end_time]

    num_locs <- length(memory_period_x)

    # spatial mixture density component
    # this subsets the locations from the start to the end (until the 24 hour delay)
    # of the memory period for x and y coords and time

    # difference between locations in the x direction
    diff_x <- location_current_x - memory_period_x
```

```r
    # difference between locations in the y direction
    diff_y <- location_current_y - memory_period_y
    # difference between locations in time
    diff_time <- as.numeric(difftime(location_current_time, memory_period_time,
                                     units = "hours"))

    # as all the parameters are on the log scale, then we can simply add them together
    # to get the probability density for a given SD parameter and exponential decay parameter
    # we use the normal density function, where the density is defined by the distance
    # (in the x or y direction, separately) and the SD parameter
    log_joint_density <- dnorm(diff_x, mean = 0, sd = spatial_sd, log = TRUE) +
      dnorm(diff_y, mean = 0, sd = spatial_sd, log = TRUE) +
      (-gamma_param*diff_time)

    # we subtract the sum of the log temporal decay component, which is equivalent
    #to dividing by the sum of the exponential components to normalise
    log_likelihood <- log_likelihood +
      logSumExp(log_joint_density) -
      log(num_locs) -
      logSumExp(-gamma_param*diff_time)

  }
  return(-log_likelihood)
}
```

This function outputs a value of the negative log-likelihood, which will be minimised during the optimisation.

Here we are using mean KDE bandwidth across individuals, although we could also use that individual's bandwidth by indexing through the `bandwidth_ref_vector` object.

```r
memory_period_hours <- 1000 # number of hours
memory_delay_hours <- 24 # number of hours to exclude from the most recent step

# buffalo id
i = 1

# picking a single buffalo from the list of ids, and selecting only the used points
buffalo_used <- buffalo_data_pres_track %>% filter(id == buffalo_ids[i] & y == 1)

tic(msg = "Single likelihood calculation")

# will output the negative log-likelihood for a single individual and gamma parameter
density_space_time_hours_subset(
  locations_x = buffalo_used$x_, # pull out a vector of x coordinates
  locations_y = buffalo_used$y_, # pull out a vector of y coordinates
  locations_time = buffalo_used$t_, # pull out a vector of times
  spatial_sd = mean_kde_sd, # mean bandwidth across individuals
  gamma_param = 0.01, # test gamma parameter
  memory_period_hours = memory_period_hours,
  memory_delay_hours = memory_delay_hours)
```

```
## [1] 205654.7
```

```r
toc()
```

```
## Single likelihood calculation: 56.46 sec elapsed
```

Estimating the temporal decay (Gamma) parameter for a single buffalo

```r
tic(msg = "ML optimisation for a single buffalo")

space_time_param_hours <- optim(
  0.01, # starting values for the gamma parameter
  density_space_time_hours_subset, # function
  locations_x = buffalo_used$x_, # single buffalo's used locations
  locations_y = buffalo_used$y_,
  locations_time = buffalo_used$t_,
  spatial_sd = mean_kde_sd, # mean bandwidth across individuals
  memory_period_hours = memory_period_hours,
  memory_delay_hours = memory_delay_hours,
  method = "L-BFGS-B", # this method allows for multiple parameters and box constraints
  lower = 0, upper = 1) # box constraints for the gamma parameter

toc()
```

```
## ML optimisation for a single buffalo: 1493.11 sec elapsed
```

```r
space_time_param_hours
```

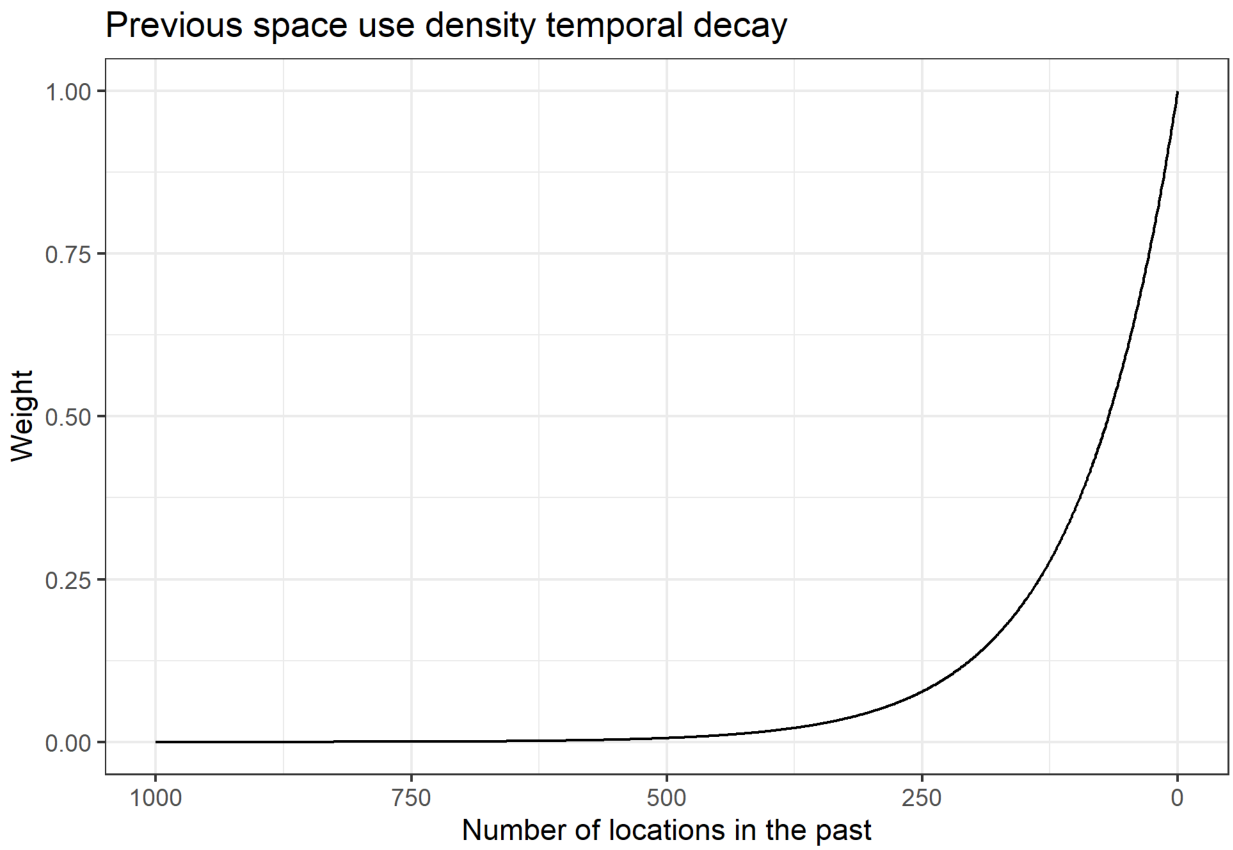
```
## $par
## [1] 0.01022755
##
## $value
## [1] 205654.2
##
## $counts
## function gradient
##        9        9
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

```r
beep(sound = 2)
```

Plotting the exponential decay component

```r
# time in the past
past_time <- 0:memory_period_hours
temporal_decay_weight <- exp(-space_time_param_hours$par[1]*(past_time))

ggplot() +
  geom_line(data = data.frame(past_time, temporal_decay_weight),
            aes(x = past_time, y = temporal_decay_weight)) +
  scale_x_reverse("Number of locations in the past") +
  labs(y = "Weight", title = "Previous space use density temporal decay") +
```

```
  theme_bw()
```



Looping over all individuals to get a temporal decay parameter for each individual, with each individual's bandwidth

This will take a while to run, so we are therefore loading the file when knitting the document. We also had memory issues when trying to run this function when knitting the document, which worked fine when running the code in the console.

```
# create an empty vector to store the estimated temporal decay parameters
exp_gamma_params_hours_subset <- c()

if(file.exists("outputs/temporal_decay_param_hours_list.rds")) {

  temporal_decay_param_hours_list <- readRDS("outputs/temporal_decay_param_hours_list.rds")
  temporal_decay_param_hours_list

  # create a vector of the estimated gamma parameters
  for(j in 1:length(buffalo_ids)) {
    exp_gamma_params_hours_subset[j] <- temporal_decay_param_hours_list[[j]]$par
  }

  exp_gamma_params_hours_subset
```

```r
} else {


  temporal_decay_param_hours_list <- vector(mode = "list", length = length(buffalo_ids))

  for(j in 1:length(buffalo_ids)) {

    tic("Each individual ML optimisation")
    # select the individual - ensure this is only USED points,
    # and not randomly sampled as well
    buffalo_used <- buffalo_data_pres_track %>% filter(id == buffalo_ids[j])

    temporal_decay_param_hours_list[[j]] <- optim(
      0.01, # starting values for the gamma parameter
      density_space_time_hours_subset, # function
      locations_x = buffalo_used$x_, # single buffalo's used locations
      locations_y = buffalo_used$y_,
      locations_time = buffalo_used$t_,
      spatial_sd = mean_kde_sd, # mean bandwidth across individuals
      memory_period_hours = memory_period_hours,
      memory_delay_hours = memory_delay_hours,
      method = "L-BFGS-B", # this method allows for multiple parameters and box constraints
      lower = 0, upper = 1) # box constraints for the gamma parameter

    print(temporal_decay_param_hours_list[[j]])
    exp_gamma_params_hours_subset[j] <- temporal_decay_param_hours_list[[j]]$par

    toc()

  }

  saveRDS(temporal_decay_param_hours_list,
          file = "outputs/temporal_decay_param_hours_list.rds")

  beep(sound = 2)

}
```

```
##  [1] 0.010227554 0.008684877 0.005923306 0.009841582 0.005915302 0.014338266 0.008119425 0.007304028
## [10] 0.005569090 0.012592407 0.007811763 0.004826748
```

Extracting parameters into data frame

```r
memory_params <- data.frame("id" = buffalo_ids,
                            "kde_sd" = bandwidth_ref_vector,
                            "temporal_decay" = exp_gamma_params_hours_subset)

write_csv(memory_params, paste0("outputs/memory_params_KDE_exp_decay_hour_subset_",
                                Sys.Date(), ".csv"))
```

For population-level estimation of the temporal decay parameter

Previously we have estimated the temporal decay (Gamma) parameter for a single buffalo. Now we will estimate the temporal decay parameter for all the individuals at once, using the `mean_kde_sd` as the spatial decay parameter.

To achieve this we just change the memory process function to have locations_x, locations_y and locations_time in lists that are iterated over in a loop. Here we are changing the function that subsetted by the number of locations, although changing the function that subsetted by the number of hours would be equivalent.

This is the function that we used to estimate the temporal decay parameter of the simulations in the paper.

```
density_space_time_ALL_hours_subset <- function(
    # this time we will pass in a list of dataframes (one for each buffalo),
    #and we will parse them out in the function
    data_list,
    spatial_sd,
    gamma_param,
    # the memory period (in locations) should cover the duration that the memory decays
    # to nearly 0
    memory_period_hours,
    # number of locations to exclude
    memory_delay_hours
    ) {

  # create object for log_likelihood
  log_likelihood <- 0

  n = length(data_list)

  for(j in 1:n) {

    # index j corresponds to individuals
    # index each dataset from the list, and then extract the relevant vector
    id_locations_x <- data_list[[j]]$x_
    id_locations_y <- data_list[[j]]$y_
    id_locations_time <- data_list[[j]]$t_


    # start from the first location AFTER a duration of the memory period,
    # to subset the previous locations
    for(i in 1:(length(id_locations_x)-memory_period_hours)) {

      # extract the current location and time
      location_current_x <- id_locations_x[i+memory_period_hours]
      location_current_y <- id_locations_y[i+memory_period_hours]
      location_current_time <- id_locations_time[i+memory_period_hours]

      # calculate the start and end times for the memory period
      #(from the start of the memory period until the memory delay begins)
      memory_start_time <- location_current_time - as.difftime(memory_period_hours,
                                                               units = "hours")

      memory_end_time <- location_current_time - as.difftime(memory_delay_hours,
                                                             units = "hours")

      # subset the x, y, and time vectors by the locations BETWEEN the memory period and memory delay
      memory_period_x <- id_locations_x[id_locations_time >= memory_start_time &
```

```
                                          id_locations_time <= memory_end_time]

      memory_period_y <- id_locations_y[id_locations_time >= memory_start_time &
                                          id_locations_time <= memory_end_time]

      memory_period_time <- id_locations_time[id_locations_time >= memory_start_time &
                                                id_locations_time <= memory_end_time]

      num_locs <- length(memory_period_x)

      # spatial mixture density component
      # this subsets the locations from the start to the end (until the 24 hour delay)
      # of the memory period for x and y coords and time

      # difference between locations in the x direction
      diff_x <- location_current_x - memory_period_x
      # difference between locations in the y direction
      diff_y <- location_current_y - memory_period_y
      # difference between locations in time
      diff_time <- as.numeric(difftime(location_current_time, memory_period_time,
                                       units = "hours"))

      # as all the parameters are on the log scale, then we can simply add them
      # together to get the probability density for a given SD parameter and
      # exponential decay parameter we use the normal density function,
      #where the density is defined by the distance
      # (in the x or y direction, separately) and the SD parameter
      log_joint_density <- dnorm(diff_x, mean = 0, sd = spatial_sd, log = TRUE) +
        dnorm(diff_y, mean = 0, sd = spatial_sd, log = TRUE) +
        (-gamma_param*diff_time)

      # we subtract the sum of the log temporal decay component,
      #which is equivalent to dividing by the sum of the exponential components
      # to normalise
      log_likelihood <- log_likelihood +
        logSumExp(log_joint_density) -
        log(num_locs) -
        logSumExp(-gamma_param*diff_time)

    }

  }

  return(-log_likelihood)

}
```

Running the optimisation

```
if(file.exists("outputs/optim_space_time_Gamma_param_ALLoptim_hour_subset.rds")) {

  space_time_ALLoptim <- readRDS("outputs/optim_space_time_Gamma_param_ALLoptim_hour_subset.rds")
```

```r
  space_time_ALLoptim


} else {


  # split the date into a list of dataframes, one for each individual
  buffalo_data_pres_list <- split(x = buffalo_data_pres_track,
                                  f = buffalo_data_pres_track$id)

  tic(msg = "Optimising over all individuals simulatenously")

  space_time_ALLoptim <- optim(
    # starting values for the gamma parameter
    0.01,
    # function
    density_space_time_ALL_hours_subset,
    # list of dataframes
    data_list = buffalo_data_pres_list,
    # value for the spatial sd (KDE bandwidth) parameter - using the mean
    spatial_sd = mean_kde_sd,
    # number of locations to include in the memory period
    memory_period_hours = 1000,
    # number of locations to exclude from the memory period, from the most recent step
    memory_delay_hours = 24,
    # this method allows for multiple parameters and box constraints
    method = "L-BFGS-B",
    # box constraints for the gamma parameter
    lower = 0, upper = 1)

  print(space_time_ALLoptim)
  toc()

  # should be just a single parameter for the gamma parameter
  saveRDS(space_time_ALLoptim,
          file = "outputs/optim_space_time_Gamma_param_ALLoptim_hour_subset.rds")

  beep(sound = 2)

}
```

```
## $par
## [1] 0.0083186
##
## $value
## [1] 1993479
##
## $counts
## function gradient
##        7        7
##
## $convergence
## [1] 0
##
```

```
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

Writing ALLoptim (optimisation of all individuals at once) memory parameters to csv

```r
# extracting the temporal decay parameter
exp_gamma_ALLoptim <- space_time_ALLoptim$par
exp_gamma_ALLoptim
```

```
## [1] 0.0083186
```

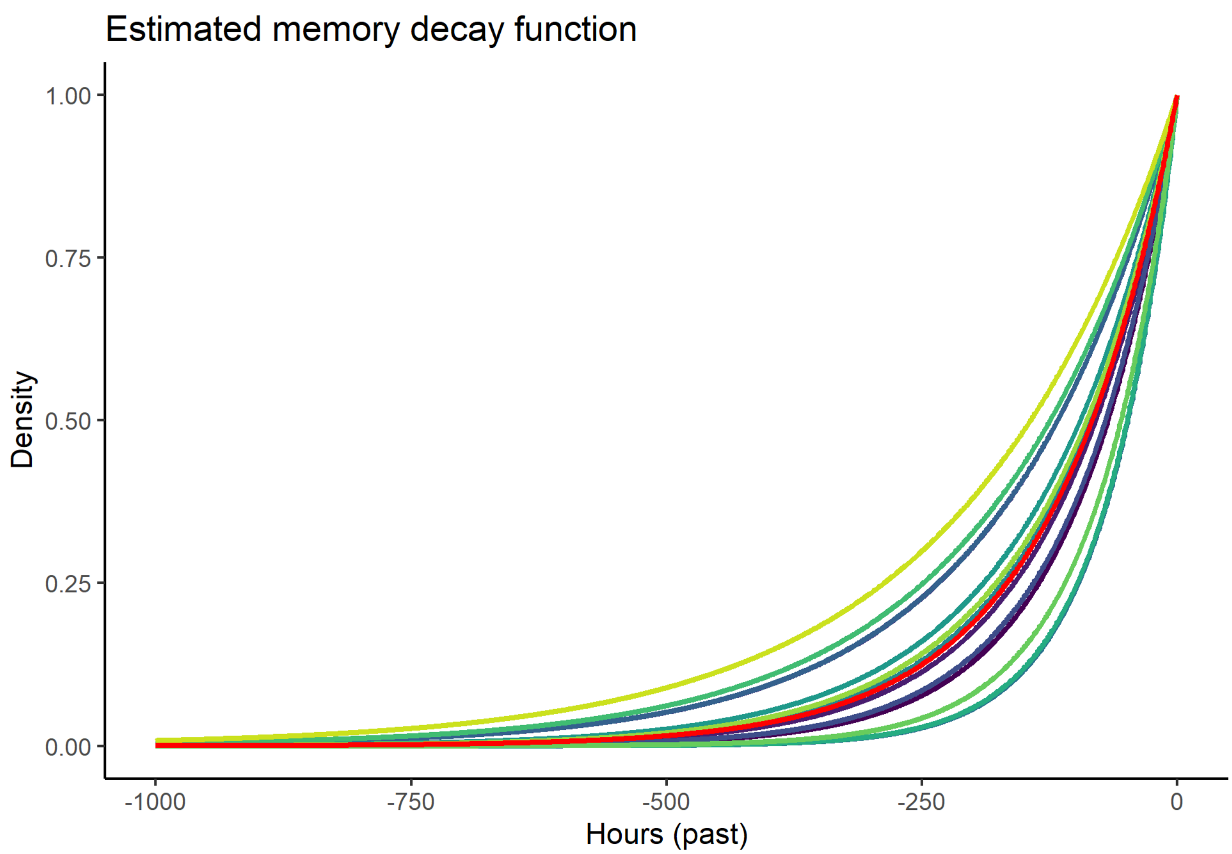
```r
# mean KDE parameters
mean_kde_sd
```

```
## [1] 571.6602
```

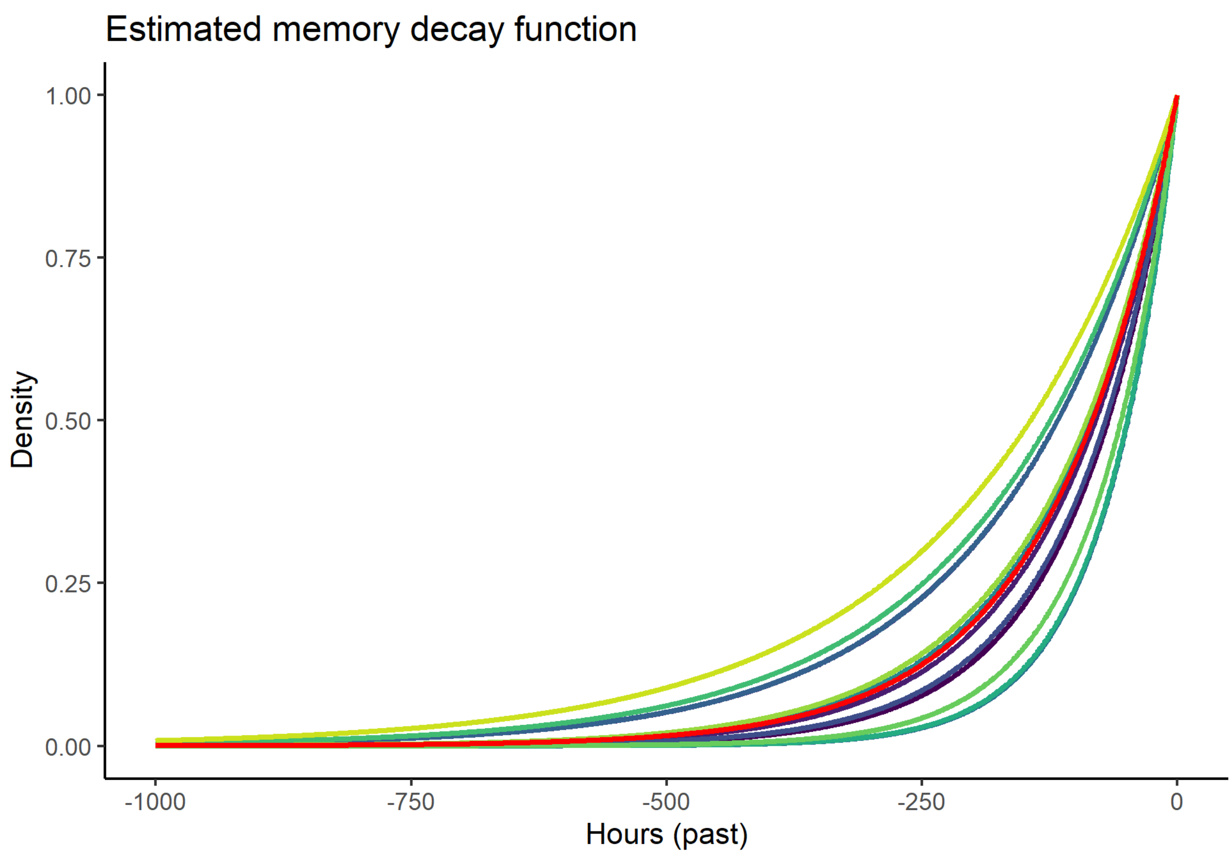
```r
# create dataframe and write to file - this dataframe will be read by the simulation scripts
memory_params_ALLoptim <- data.frame(exp_gamma_ALLoptim, mean_kde_sd)
write.csv(memory_params_ALLoptim,
          file = paste0("outputs/memory_params_ALLoptim_hour_subset_", Sys.Date(), ".csv"))
```

Plotting the optimised temporal decay parameter(s)

```r
memory_decay <- data.frame(0:memory_period_hours,
                           sapply(c(exp_gamma_params_hours_subset, exp_gamma_ALLoptim),
                                  function(gamma) exp(-gamma*(0:memory_period_hours))))
# add names to the dataframe
colnames(memory_decay) <- c("x", buffalo_ids, "mean")

# prepare for plotting with ggplot
memory_decay_long <- pivot_longer(memory_decay,
                                  cols = !1,
                                  names_to = "id",
                                  values_to = "value")

# Create color mapping
unique_groups <- unique(spatial_sds_long$id)
colors <- viridis(length(unique_groups))
names(colors) <- unique_groups
colors["mean"] <- "red"


ggplot(memory_decay_long) + #  %>% filter(name != "2154")
  geom_line(aes(x = -x, y = value, colour = id), size = 1) +
  scale_x_continuous("Hours (past)") +
  scale_y_continuous("Density") +
  # scale_colour_viridis_d("ID") +
  scale_colour_manual("ID", values = colors) +
  ggtitle("Estimated memory decay function") +
  theme_classic() +
  theme(legend.position = "none")
```




```r
ggplot(memory_decay_long%>% filter(id != "2154")) +
  geom_line(aes(x = -x, y = value, colour = id), size = 1) +
  scale_x_continuous("Hours (past)") +
  scale_y_continuous("Density") +
  # scale_colour_viridis_d("ID") +
  scale_colour_manual("ID", values = colors) +
  ggtitle("Estimated memory decay function") +
  theme_classic() +
  theme(legend.position = "none")
```



Adding previous space use density to the used and random steps

This function subsets the previous used locations within the memory period, and calculates the previous location density for all used and random steps based on the estimated memory parameters

```
spatial_temporal_density_function <- function(data_input,
                                              id_val,
                                              memory_period, # in hours
                                              memory_delay, # in hours
                                              spatial_sd,
                                              temporal_decay_gamma) {

  # subset by individual
  # all locations
  id_all_locations <- data_input %>% filter(id == id_val)
  # just the used locations to estimate the density (we want to estimate the
  # previous space use density only to used locations)
  id_used_locations <- data_input %>% filter(y == 1 & id == id_val)
  location_density <- c()

  for(i in 1:nrow(id_all_locations)){

    # current location
    location_x <- id_all_locations[i,]$x2_
    location_y <- id_all_locations[i,]$y2_
```

```r
    location_time <- id_all_locations[i,]$t2_

    # memory period start and end to subset with
    memory_start_time <- location_time - as.difftime(memory_period, units = "hours")
    memory_end_time <- location_time - as.difftime(memory_delay, units = "hours")

    # subset locations to use to estimate previous space use density
    memory_x <- id_used_locations[id_used_locations$t2_ >= memory_start_time &
                                id_used_locations$t2_ <= memory_end_time, ]$x1_

    memory_y <- id_used_locations[id_used_locations$t2_ >= memory_start_time &
                                id_used_locations$t2_ <= memory_end_time, ]$y1_

    memory_time <- id_used_locations[id_used_locations$t2_ >= memory_start_time &
                                   id_used_locations$t2_ <= memory_end_time, ]$t1_

    # difference in x, y and time to all points in the memory subset
    diff_x <- location_x - memory_x
    diff_y <- location_y - memory_y
    diff_time <- as.numeric(difftime(location_time, memory_time, units = "hours"))

    # calculate the density in relation to all the points in the memory subset
    log_joint_density <- dnorm(diff_x, mean = 0, sd = spatial_sd, log = TRUE) +
      dnorm(diff_y, mean = 0, sd = spatial_sd, log = TRUE) +
      (-temporal_decay_gamma*diff_time)

    # estimate the previous space use density for that location
    location_density[i] <- logSumExp(log_joint_density) -
      logSumExp(-temporal_decay_gamma*diff_time)

  }

  return(location_density)

}
```

Adding the previous space use density to the data using *individually* estimated memory parameters

For fitting individual-level models (not recommended when fitting population-level or hierarchical models)

```r
memory_period <- 1000 # in hours
memory_delay <- 24 # in hours

buffalo_id_memory_list <- vector(mode = "list", length = length(buffalo_ids))

for(i in 1:length(buffalo_ids)) {

  tic(msg = "Adding previous space use density to used and random steps")

   buffalo_id_memory_list[[i]] <- buffalo_data_rand_steps %>% filter(id == buffalo_ids[i]) %>%
     mutate(kde_memory_density_log = spatial_temporal_density_function(
```

```r
      ., id_val = buffalo_ids[i],
      memory_period = memory_period,
      memory_delay = memory_delay,
      spatial_sd = memory_params$kde_sd[i],
      temporal_decay_gamma = memory_params$temporal_decay[i]))

  toc()
}

buffalo_data_rand_steps_memory <- bind_rows(buffalo_id_memory_list)
write_csv(buffalo_data_rand_steps_memory,
          paste0("outputs/buffalo_popn_GvM_KDEmem_ID_hour_subset_10rs_", Sys.Date(), ".csv"))

beep(sound = 2)
```

Adding the previous space use density to the data using *population* estimated memory parameters

The only thing that changes here is that instead of indexing over the bandwidth and temporal decay parameters there is only one of each memory parameter

```r
buffalo_ALLoptim_memory_list <- vector(mode = "list", length = length(buffalo_ids))

for(i in 1:length(buffalo_ids)) {

  tic(msg = "Adding previous space use density to used and random steps")

  buffalo_ALLoptim_memory_list[[i]] <- buffalo_data_rand_steps %>% filter(id == buffalo_ids[i]) %>%
    mutate(kde_memory_density_log = spatial_temporal_density_function(
      ., id_val = buffalo_ids[i],
      memory_period = memory_period,
      memory_delay = memory_delay,
      spatial_sd = mean_kde_sd,
      temporal_decay_gamma = exp_gamma_ALLoptim))

  toc()
}

buffalo_data_rand_steps_memory_ALLoptim <- bind_rows(buffalo_ALLoptim_memory_list)
write_csv(buffalo_data_rand_steps_memory_ALLoptim,
          paste0("outputs/buffalo_popn_GvM_KDEmem_allOPTIM_hour_subset_10rs_", Sys.Date(), ".csv"))

beep(sound = 2)
```

Session info

```r
sessionInfo()
```

```
## R version 4.2.1 (2022-06-23 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19045)
##
## Matrix products: default
```

```
##
## locale:
## [1] LC_COLLATE=English_New Zealand.utf8  LC_CTYPE=English_New Zealand.utf8    LC_MONETARY=English_New
## [4] LC_NUMERIC=C                         LC_TIME=English_New Zealand.utf8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] cowplot_1.1.1       ggExtra_0.10.1      ggh4x_0.2.6         Rfast_2.0.7         RcppZiggurat_0.
##  [6] formatR_1.14        scales_1.2.1        glmmTMB_1.1.8       clogitL1_1.5        Rcpp_1.0.10
## [11] ecospat_3.5         TwoStepCLogit_1.2.5 survival_3.5-5      viridis_0.6.2       viridisLite_0.4
## [16] matrixStats_1.0.0   patchwork_1.1.2     ggpubr_0.6.0        adehabitatHR_0.4.21 adehabitatLT_0.
## [21] CircStats_0.2-6     boot_1.3-28.1       MASS_7.3-59         adehabitatMA_0.3.16 ade4_1.7-22
## [26] sp_1.6-0            ks_1.14.0           beepr_1.3           tictoc_1.2          terra_1.7-23
## [31] amt_0.2.1.0         lubridate_1.9.2     forcats_1.0.0       stringr_1.5.0       dplyr_1.1.2
## [36] purrr_1.0.1         readr_2.1.4         tidyr_1.3.0         tibble_3.2.1        ggplot2_3.4.2
## [41] tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##   [1] utf8_1.2.3            tidyselect_1.2.0      lme4_1.1-32           htmlwidgets_1.6.2
##   [5] grid_4.2.1            pROC_1.18.0           munsell_0.5.0         codetools_0.2-19
##   [9] ragg_1.2.5            units_0.8-1           miniUI_0.1.1.1        withr_2.5.0
##  [13] audio_0.1-10          colorspace_2.1-0      highr_0.10            knitr_1.42
##  [17] rstudioapi_0.14       ggsignif_0.6.4        Rdpack_2.4            labeling_0.4.2
##  [21] emmeans_1.8.5         TeachingDemos_2.12    bit64_4.0.5           farver_2.1.1
##  [25] coda_0.19-4           vctrs_0.6.2           generics_0.1.3        TH.data_1.1-2
##  [29] circular_0.4-95       xfun_0.39             timechange_0.2.0      randomForest_4.7-1.1
##  [33] R6_2.5.1              isoband_0.2.7         cachem_1.0.7          reshape_0.8.9
##  [37] promises_1.2.0.1      vroom_1.6.1           multcomp_1.4-23       nnet_7.3-18
##  [41] gtable_0.3.3          mda_0.5-3             sandwich_3.0-2        rlang_1.1.0
##  [45] systemfonts_1.0.4     splines_4.2.1         rstatix_0.7.2         TMB_1.9.10
##  [49] earth_5.3.2           broom_1.0.4           checkmate_2.1.0       biomod2_4.2-2
##  [53] yaml_2.3.7            reshape2_1.4.4        abind_1.4-5           backports_1.4.1
##  [57] httpuv_1.6.9          Hmisc_5.0-1           tools_4.2.1           nabor_0.5.0
##  [61] ellipsis_0.3.2        raster_3.6-20         jquerylib_0.1.4       RColorBrewer_1.1-3
##  [65] proxy_0.4-27          plyr_1.8.8            base64enc_0.1-3       classInt_0.4-9
##  [69] rpart_4.1.19          zoo_1.8-12            cluster_2.1.4         tinytex_0.48
##  [73] magrittr_2.0.3        data.table_1.14.8     mvtnorm_1.1-3         fitdistrplus_1.1-8
##  [77] mime_0.12             hms_1.1.3             evaluate_0.20         xtable_1.8-4
##  [81] mclust_6.0.0          gridExtra_2.3         compiler_4.2.1        KernSmooth_2.23-20
##  [85] crayon_1.5.2          minqa_1.2.5           htmltools_0.5.5       later_1.3.0
##  [89] mgcv_1.8-42           tzdb_0.3.0            Formula_1.2-5         DBI_1.1.3
##  [93] sf_1.0-12             Matrix_1.6-5          car_3.1-2             permute_0.9-7
##  [97] cli_3.6.1             rbibutils_2.2.13      parallel_4.2.1        pkgconfig_2.0.3
## [101] numDeriv_2016.8-1.1   foreign_0.8-84        foreach_1.5.2         bslib_0.4.2
## [105] estimability_1.4.1    plotmo_3.6.2          digest_0.6.31         pracma_2.4.2
## [109] vegan_2.6-4           rmarkdown_2.21        htmlTable_2.4.1       PresenceAbsence_1.1.11
## [113] shiny_1.7.4           gtools_3.9.4          nloptr_2.0.3          lifecycle_1.0.3
## [117] nlme_3.1-162          jsonlite_1.8.4        carData_3.0-5         fansi_1.0.4
## [121] pillar_1.9.0          lattice_0.21-8        fastmap_1.1.1         plotrix_3.8-2
## [125] glue_1.6.2            gbm_2.1.8.1           iterators_1.0.14      bit_4.0.5
## [129] class_7.3-21          stringi_1.7.12        sass_0.4.5            maxnet_0.1.4
## [133] textshaping_0.3.6     poibin_1.5            e1071_1.7-13          ape_5.7-1
```