Environmental Impact on Ungulate Behavior

Master's Thesis

Taylor Carter

Prof. Dr. Iain Couzin

Dr. Alex Jordan

Konstanz

July 2020

Max Planck Institute
of Animal Behavior

TABLE OF CONTENTS

I. ABSTRACT

Collective movement behavior for group-living animals residing in heterogeneous landscapes is a product of both environmental variables (like habitat structure) and social factors (like interactions between individuals). Furthermore, perceived risk in the environment from disturbances such as predation can influence where animals choose to go. In this study, I examine how a suite of environmental and social factors impact movement decision-making of individuals residing in herds of wild Grevy's zebra (*Equus grevyi*) and plains zebra (*Equus quagga*) at Mpala Research Centre in Kenya. I use movement data collected from unmanned aerial vehicles (UAVs) flying above the zebra herds to simultaneously track the movements of all herd members at once. Data on the habitat structure is produced using 3-dimension landscape renderings from the high resolution drone imagery. Using step selection functions and probability analyses, I analyze the importance of the following features of the social and ecological landscape in driving zebra movement behavior: 1) social density (number of group mates nearby within a certain area), 2) group center, 3) recently-used space (number of other zebras that have occupied a certain location recently), and 4) game trails. I further assess how movement decisions change before and after a "scare event," where a perceived disturbance in the environment visibly startles the zebra groups. Results broadly indicate that zebras tend to loosely group with other individuals in their vicinity, travel in areas that were recently used by other herd members up to a distance of 10 meters, and move along game trails (especially after scare events for Grevy's zebras). By exploiting recent advances in tracking technology with unmanned aerial vehicles, remote sensing of the environment, and analytical methods in quantifying animal movement, this study reveals ecological and social variables that influence the local movement decisions made by individual zebras living in wild herds.

II. INTRODUCTION

Collective animal behavior is ubiquitous, whether it be bird flocks, schooling fish, foraging ants, or herding ungulates. Studies of collective phenomena seek to understand how interactions between individuals produce large scale patterns of behavior (Sumpter 2010). A central question in the study of such collective behavior is how animals coordinate their movement, especially when living in a group or a herd that often moves as a unit (Vicsek & Zafeiris 2012). What biological mechanisms drive coordinated motion and collective movement behavior? The study of collective behavior tackles these questions head on, attempting to understand how both social and environmental influences shape biological processes at a group level.

Understanding the principles driving collective behavior can facilitate progress in a variety of fields, including anthropology, behavioral ecology, and conservation biology. Behavioral research conducted on non-human primates for instance, can shed light on human behavior, offering valuable perspectives about the evolution and fitness consequences of individual and social human actions. Animal behavior also helps us confront vexing conservation problems, such as how to save endangered species,

assess environmental quality, design nature preserves, and evaluate the importance of human-related threats to animal survival. Studying how animal collectives behave and interact with their environments can also inform the human development of things such as swarm robotics and search algorithm design.

However, characterizing the motivations or preferences driving animal collective movement remains difficult as researchers cannot directly communicate with their study organisms to ascertain what they want to do when. Ecologists and biologists use animal movement patterns as a metric to understand animal preferences for a variety of things, including habitat selection, migration, territoriality, foraging, mating, and responses to environmental changes. Understanding what factors influence movement can help us develop models that are capable of predicting behavior, even before the behavior happens. Not only could these predictive behavioral models be useful for understanding how animals might react in a variety of contexts, they could also be useful for predicting human behavior in fields such as behavioral analytics.

As animals move, they confront a continuous stream of choices about how to react to social and environmental stimuli. Studies of collective motion however tend to discount the potential role of environmental complexity. Experiments are either conducted in relatively simple laboratories where the environment is carefully controlled or in the wild, where often only social information is considered and the environment is neglected. However, in the field of movement ecology, the opposite is true where scientists tend to ignore potentially relevant social information, considering only how habitat influences movement decisions. Thus, in both the lab and the field, seldom are both social and environmental influences considered concurrently when assessing animal movement.

There are a variety of examples in laboratory settings where studies overlook the role of the environment in influencing collective movement behavior. For instance, a study by Buhl et al. 2006 designed an experiment with desert locusts (*Schistocerca gregaria*) with featureless arenas to investigate whether the transition of disordered movement to ordered movement is dependent on the density of locusts in the arena. While Buhl et al. confirm that the collective movement transition is a density dependent process, they do so without assessing the impact of the environment. By conducting experiments in featureless ring-shaped arenas, they discount any potential role the environment could have played in the transition from disordered to ordered movement. Similarly, Herbert-Read et al. in 2011 employed square, featureless arenas to study individual level interactions between mosquitofish (*Gambusia holbrooki*) and uncover the basic rules for collective motion. Although they nicely outline three rules governing social interactions, they discount any potentially impactful environmental data with their experimental setup. The same caveat is true for Katz et al. 2011. They use a white, featureless arena surrounded by a floor to ceiling white curtain to study individual level interactions in shoaling fish . Yet another example can be found in Rosenthal et al.'s 2015 study of collective evasion. For their experiment, they placed schooling fish (*Notemigonus crysoleucas*) into a featureless tank and studied the social cues that produce collective evasion. While their results suggest that collective evasion is largely dependent on nearest neighbor responses, these results do not consider how environmental features

might also influence such collective evasion behavior. One hallmark study that does examine a more complex environment in the lab in concert with data on collective movement behavior was conducted by Berdahl et al. 2013 with golden shiner fish (*Notemigonus crysoleucas*). Their results indicate that collective sensing occurs at the group level as individuals adjust their speed in response to environmental variables (notably, light) and social interactions. Ultimately, in order to have a comprehensive study of collective movement, both social and environmental information must be considered together.

Furthermore, even when animals are studied in their natural habitats, environmental data is often not accounted for. For instance, in a 2006 study, Biro et al. studied how homing pigeon (*Columba livia*) groups make navigational decisions when group members disagree on the direction of travel. They released pairs of birds with two different home destinations from the same location and observed where the birds decided to go despite their conflicting home destinations. If the conflict between the directional difference was small, the birds averaged their routes. However, if the conflict rose over a critical threshold, either the pair split or one of the birds became the leader and led the other bird along its preferred route. Although they conducted these experiments in a natural habitat, the influence of landscape features on the pigeons' movement behavior would provide additional information on how movement decisions are reached. Another study by Strandburg-Peshkin et al. 2015 investigated how consensus is achieved in groups of wild olive baboons (*Papio anubis*) when individuals disagree about where to go. They found that when a disagreement in direction arose, i.e. when one group of baboons went one way while another group of baboons went a different way, baboons only chose to follow one group or the other if the angle between the travel directions was large. Alternatively, if this angle was small, the baboons averaged the travel direction between the two groups and followed somewhere in between. This study highlighted that movement decisions are made on the group level, negating the previous notion that in times of indecision, high-ranking individuals have outsized influence on group behavior. Although these results are highly impactful, this study focuses on the social influences of group decision making and ignores any environmental factors. In a follow-up study to this 2015 *Science* paper, Strandburg-Peshkin et al. 2017 incorporate environmental and social variables into their modelling of how olive baboons decide where to move (discussed below). Another recent example of studying collective behavior in an environmental context assessed how groups of storks (*Ciconia ciconia*) collectively navigate air thermals (Nagy et al. 2017). These types of studies that incorporate aspects of both the environment and social interactions into models of animal movement behavior provide the basis for this thesis.

On the other hand, some studies in the field of movement ecology tend to focus only on environmental data and neglect the social information when studying behavior. For instance, Leblond et al. in their 2010 study on moose (*Alces alces*) use GPS tracking to uncover what environmental features affect moose movement. After a comprehensive analysis of vegetation, solar energy, and topography, they find that topography proved to be the most influential driver of moose movement. Nevertheless, since moose often associate with other individuals in the population, a broader analysis of how social

factors can affect moose movement would be beneficial. Similarly, in 2012 Northrup et al. conducted a study on how traffic flow impacts grizzly bear (*Ursus arctos*) behavior in Alberta, Canada. They found that grizzlies are less influenced by the number of roads in the area and rather the amount of traffic on those roads. However, what about the social impact on grizzly movement? Further analysis of how these animals adjust their movement in relation to social factors would provide a more holistic understanding of how animals make movement decisions.

In a groundbreaking study, Strandburg-Peshkin et al. 2017 investigated the interplay between social and habitat influences on collective movement at both an individual and group level. They studied the movement and spatial organization of olive baboons (*Papio anubis*) in their natural habitat at Mpala Research Centre in Laikipia, Kenya. In their study, they measure a variety of environmental and social features (environmental vegetation density, social density, sleep site direction, roads, recently used space, ever-used space, animal paths, visible neighbors, and slope) that could potentially influence collective movement. Overall, they found that both social and environmental factors influence baboon movement, illustrating the need for studies to consider these factors in concert. Specifically, baboons tended to consider where other baboons have previously been ("ever-used space"), as well as the direction of their sleep site to be the most important drivers of movement. This paper brings together cutting edge technology and analyses like GPS tracking, 3-dimensional landscape rendering, and step selection analysis to achieve a comprehensive understanding of the key drivers of baboon collective movement.

As the majority of animals aggregate in groups at some point in their lives, studies that concurrently consider both the social and environmental influences driving collective movement will advance our understanding of how individuals coordinate their actions and respond to the stimuli around them.

III. STUDY OVERVIEW

For my thesis, I analyze how several habitat and social factors shape individual decisions about where to go during the collective movement of two zebra species, the Grevy's zebra (*Equus grevyi*) and the plains zebra (*Equus quagga)*. Specifically, I assess if the desire to be at the center of a group (group center), stand closely with conspecifics (social density), follow conspecifics (recently used space), or walk on a game trail (game trails) have an affect on Grevy's and plains zebra movement. I analyze the importance of each of these features at both the genus level and the species level.

Analysis at the genus level enables a broad look at what drives zebra movement across species. As such, I aim to determine if zebra movement as whole is more influenced by either social inputs, environmental inputs, or a combination of both. As Grevy's and plains zebras differ considerably in their social dynamics (as discussed below), the analysis of zebra movement at the species level allows us to determine if the drivers of movement change for such socially different species.

Importantly, Grevy's and plains zebras differ in the stability and cohesiveness of their herds and the extent of social bonds among herd-mates. Overall, Grevy's tend to be much less social than plains. Grevy's do not form permanent herds or lasting social bonds between adult individuals. Associations between individual Grevy's tend to be ephemeral groupings of mares with their foals, with no hierarchical structure to these temporary groups. While the Grevy is largely apathetic to its conspecifics outside of mating events, plains zebras care deeply for each member of their harem. Plains zebras live in small harem units consisting of a male stallion and several mares with their foals. These harems can temporarily combine with other harems to form large herds, but members of the harem unit form lasting social bonds and stay close together. As a result of this contrasting social structure between the two zebra species, I expect that Grevy's will rely less on social information and more heavily on environmental information than the plains zebras during movement. Specifically, I predict that Grevy's will be less concerned about: where other individuals are headed (recently used space), being at the center of the group (group center), and being next to other individuals (social density). I also predict that as a consequence of a reduced focus on social information, Grevy's will be more concerned with finding avenues to get where they need to go (game trails). Conversely, I expect plains zebras to be much more concerned with incoming social information than environmental information during movement.

To add an additional dimension of how environmental factors such as risk can influence movement, I further assess movement drivers before and after a "scare event" for the two species. Zebras are a prey species for many top carnivores in their habitat (lions, leopards, cheetah, wild dogs, and hyenas) and thus experience moments of alarm when they are responding to what they believe is a possible threat (defined in this thesis as a "scare event"). To obtain a well-rounded view of how zebras make important movement decisions, it is critical to understand their movement in both a foraging as well as an anti-predatory state. I predict that after a scare event when the animals are startled, plains zebras will be more likely to move towards members of their herd (social density and group center), because maintaining social cohesion with their harem is an important part of their ecology. Conversely, I do not expect the less-social Grevy's zebras to prioritize social variables during their movement. Instead, I predict that because game trails may provide straightforward avenues of escape from a potential threat, Grevy's will be more likely to move towards game trails as opposed to towards conspecifics.

Step selection

Much like Strandburg-Peshkin et al. 2017, I use the step selection function to assess the importance of each environmental and social feature. Step selection is a tool that can be used to assess key drivers of animal movement. Movement is broken down into sets of consecutive locations or steps through a landscape, where each step is influenced by *n* number of elements called features. Features can be trees, roads, distance from sleep site, nearest neighbors, etc. Probabilistic models are fitted to predict the next location an animal will move to based on the set of features associated with

each step. Each step is paired with one or more randomly generated alternative steps with the same starting point. Alternative steps are drawn at random from a distribution of step lengths (distance traveled during each step) and turning angles (angle between consecutive step directions) and are used to characterize what is 'available' to the animal during its movement. From the resulting model fits, the relevance and strength of influence of each feature on animal movement can be inferred.

Traditionally, researchers have used step selection in an environmental context to either infer the habitat preferences of animals or understand how animals respond to different distributions of predators or competitors. For example, in 2008 Coulon et al. used step selection to infer the habitat preference of roe deer (*Capreolus capreolus*) in a rural region of South-Western France. Specifically, they investigated the landscape structures that roe deer prioritized most when selecting their habitats. They found that out of all the tested features (distance to buildings, distance to roads, and topographical position), the distance to roads proved to be the most influential feature for roe deer when selecting their habitat. Similarly, Hodson et al. 2010 used step selection to assess how gaps in Canadian boreal forests, an environmental feature, impacted snowshoe hare (*Lepus americanus*) habitat selection. They found that overall, hares increased their speed in areas of relatively low cover and tend to bias their movements away from gaps, only using gaps as foraging locations. Each of these studies implement step selection with environmental features to understand habitat selection behavior. Although informative, this work can be built on by incorporating the important influence of social information on animal movement to gain a more holistic understanding of what aspects of the social and physical environment animals prefer during movement.

Step selection has also been used to understand how animals respond to risk in their environment, such as different distributions of predators. In 2005 Fortin et al. used step selection to investigate the environmental features responsible for the observed trophic cascade among wolves (*Canis lupus*), elk (*Cervus canadensis*), and aspen trees (*Populus tremuloides*) in Yellowstone National Park, Wyoming. They discovered that elk movement was largely dependent on the spatial distribution of wolves across the park. When there were a small number of wolves in the area, elk primarily preferred to be near aspen stands, then open areas, and finally conifer forests. Conversely, when there were a large number of wolves, elk preferred to hide mostly in conifer forests. Basille et al. in 2015 used the same method to investigate how caribou (*Ragnifer tarandus*) respond to wolf distributions in the boreal forest of the Côte-Nord region in Québec, Canada. They found that similar to elk, caribou habitat selection changed in the presence of wolves. Female caribou tended to avoid open areas and deciduous forests and instead moved swiftly toward foraging areas when wolves were within 2.5 km. They found that caribou also avoided food-rich areas only when wolves were within 1 km of their location. Similarly, in 2014 Latombe et al. used step selection to uncover how wolves influence caribou and moose movement patterns in the Côte-Nord region of Québec, Canada. In general, they found that both caribou and moose increased their propensity to select safe land cover types. In each of these studies, step selection was used to investigate animal responses to predator distribution, overlooking potentially vital social information present within the elk, caribou, or moose populations.

Incorporating social information into such research will bolster our understanding of predator-prey dynamics.

In addition to studying how animals react to predators, step selection has been used to explore how animals adjust their movement in response to other competitors in their environment. For example, in 2013, Vanak et al. used step selection to study the coexistence of multiple carnivore species, lions (*Panthera leo*), leopards (*P. pardus*), cheetahs (*Acinonyx jubatus*), and African wild dogs (*Lycaon pictus*), in the Karongwe Game Reserve, South Africa. Similarly, in 2014 Potts et al. used step selection to study how birds in the Amazon react to their competitors, specifically what mechanisms drive territorial behavior. Traditionally, it was thought that territorial behavior was derived from birds simply knowing where competitor territories were and avoiding them. However, Potts et al. found that birds tended to retreat back inside their territory once they encountered a competitors' territory. Much like the studies listed above about predator-prey dynamics, these studies might be even more impactful if they incorporate both social and environmental features into their step selection function. Since social information is a prominent aspect of many animal groups, both social and environmental information should be considered when discussing how animals choose where to move.

Technological advances to generate animal movement data

Although models like step selection are useful for exploring animal movement behavior, they need sufficient data to be rigorously tested. In the past, acquiring enough data to properly utilize these models was difficult. The advent of affordable GPS tracking technologies like collars or GPS transmitting tags gave researchers the ability to acquire much larger datasets of animal positions, allowing researchers to study fine-scale movement decisions and social interactions (Nagy et al., 2010; King et al., 2012; Mann et al., 2014; Giuggioli et al., 2015; Strandburg-Peshkin et al., 2015). Simultaneously, a variety of tools to quantify aspects of the environment became more widely accessible to researchers in biology, including unmanned aerial vehicles (UAVs) as well as computational advances in image stitching and three-dimensional landscape reconstruction (Anderson and Gaston, 2013). Since their conception, these tools have undergone constant innovation. Specifically, they have been paired with powerful computer vision hardware and software, making it possible for scientists to track animals in the wild without using things like GPS radio collars (Dell et al., 2014). Not only can general positions of animals be tracked but their body posture can be tracked as well (Crall et al., 2015; Graving, 2017; Wild et al., 2018; Boenisch et al., 2018, Pérez-Escudero et al., 2014; Romero-Ferrero et al., 2019; Graving et al., 2019). Much like their predecessors, these technologies give biologists access to higher resolution behavioral and movement data and allow biologists to track all (or most) of the individuals in groups simultaneously.

Moving beyond GPS, new computer vision techniques allow us to extract meaningful information from videos of animals and landscapes filmed in the wild. Much of this success can be attributed to deep learning, a class of machine learning algorithms that learn complex statistical relationships from data (LeCun et al., 2015). There are a few

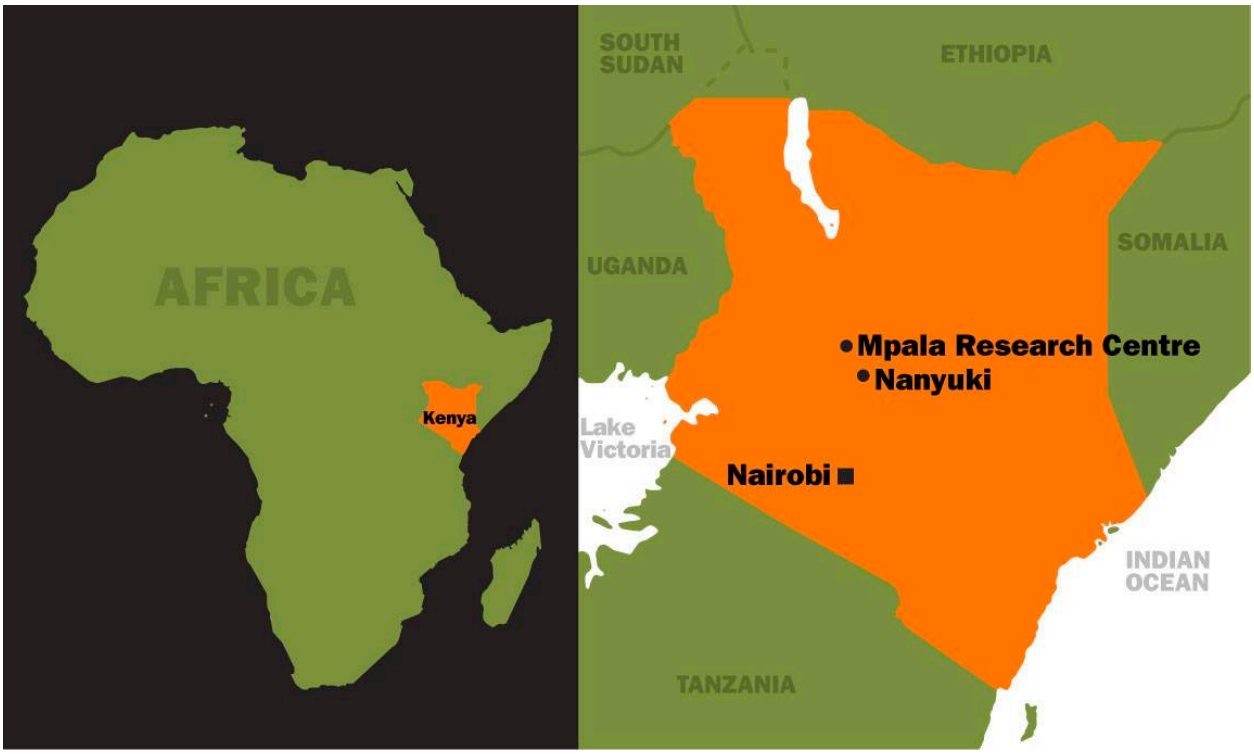
types of deep learning algorithms (e.g. deep neural networks, deep belief networks, recurrent neural networks, and convolutional neural networks), but the most important type to the field of computer vision are convolutional neural networks (ConvNets, see Appendix for explicit details on this method).

This project combines cutting edge technology and analytical tools like ConvNets, computer vision, and UAVs to conduct a comparative study of the movement of Grevy's and plains zebras in their natural habitat. Movement data was acquired using unmanned aerial vehicles (UAVs) and habitat data was acquired from 3-dimensional landscape renderings collected using high resolution drone imagery. To my knowledge, this is the only study that uses these technologies along with step selection to study how both habitat and social factors influence movement.

IV. METHODS

Study site

Data was collected at Mpala Research Centre, a 50,000-acre reserve and ranch in Laikipia County in central Kenya located 150 miles north of Nairobi. The facility is operated as a partnership involving Princeton University, the Smithsonian Tropical Research Institute, the Kenya Wildlife Service and the National Museums of Kenya. The property is maintained as a conservancy for native wildlife including large populations of African elephants, the endangered Grevy's zebra, the reticulated giraffe, wild dogs, other large mammals as well as multiple species of birds. The environment is semi-arid, with plants and wildlife highly adapted to long spans of dry weather and periodic rains.



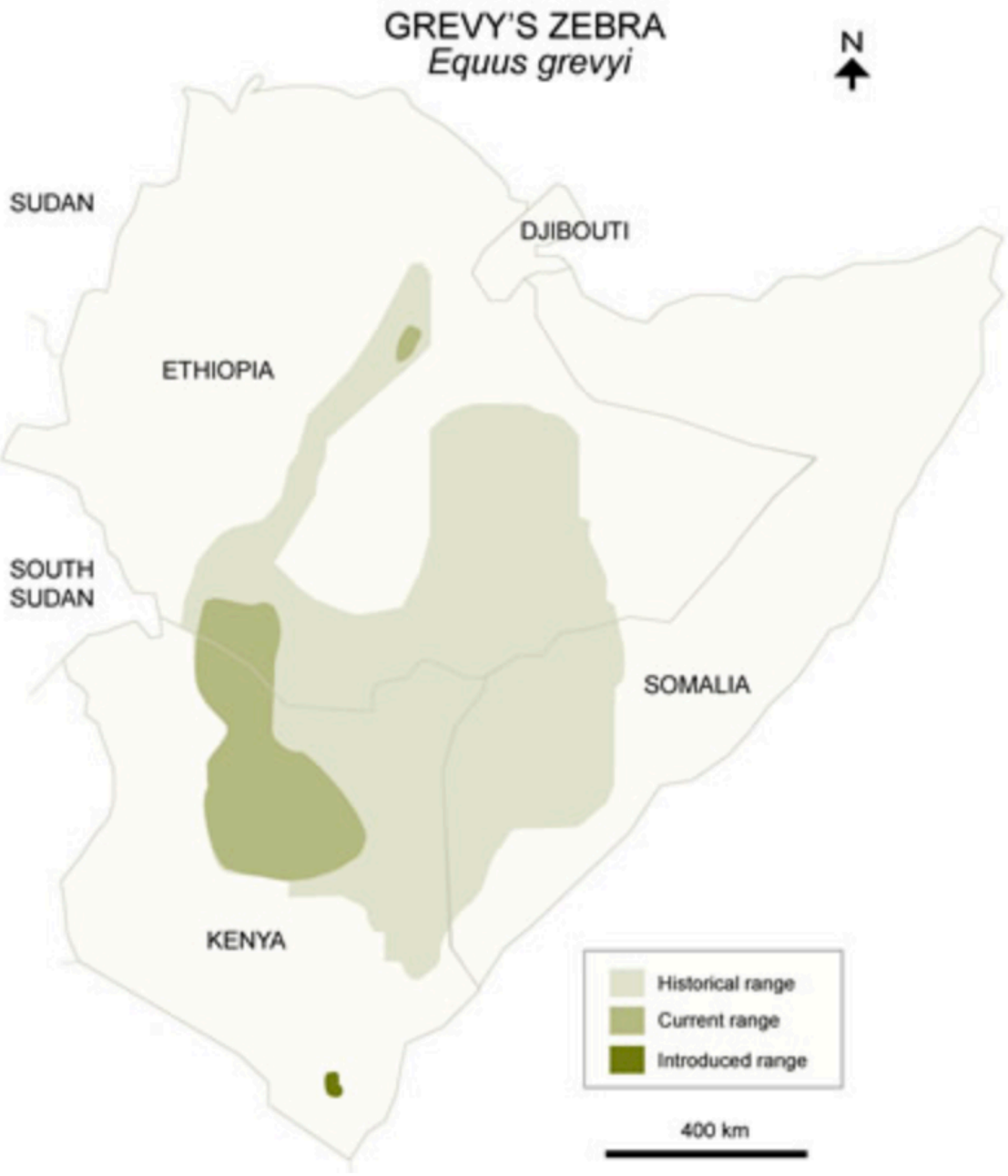
The Mpala Research Centre is in Laikipia County in central Kenya, roughly 150 miles north of Nairobi, Kenya's capital. (https://www.princeton.edu/news/2016/10/20/wild-science-nature-mpala-research-centre)

Study species

For this project we studied both the Grevy's zebra (*Equus grevyi*) as well as the plains zebra (*Equus quagga).*

Grevy's zebra

The Grevy's zebra (*Equus grevyi*), also known as the imperial zebra, is the largest living wild equid and the largest of all the zebra species. Although primarily dwelling in Ethiopia at the time, fossil records put ancestral forms of Grevy's in places as far reaching as China, Uzbekistan, South Africa and Egypt, with a nearly two million year old fossil being found in South Africa (Smithsonian). Today, the majority of the Grevy's population resides between the eastern side of Kenya's Rift Valley and the Tana River. The remaining individuals gather in the Alledeghi Plains of Ethiopia (IUCN). Grevy's are currently an endangered species with roughly only 2,500 of them remaning in the world (IUCN).



Distribution of Grevy's zebra populations in Eastern Africa (https://www.researchgate.net/publication/261066509_Genetic_diversity_of_the_Ethiopian_Grevy%27s_zebra_Equus_grevyi_populations_that_includes_a_unique_population_of_the_Alledeghi_Plain/figureslo=1&utm_source=google&utm_medium=organic)

Grevy's zebra typically live in arid/semi-arid grass/shrub-land landscapes. Grevy's feed mainly on grasses but they will also consume bark, fruit, and leaves. Extracting nutrients from these sources is difficult and thus requires a high intake volume and a specialized gut. Consequently, Grevy's typically spend up to 60 percent of their day just eating. In drier times when food is scarce, eating can occupy up to 80 percent of their time

(Smithsonian). The primary predator of the Grevy's zebra is the lion, however, other predators include: leopards, spotted hyenas, hunting dogs, and cheetahs (Lycaon pictus-Haltenorth and Diller, 1980).

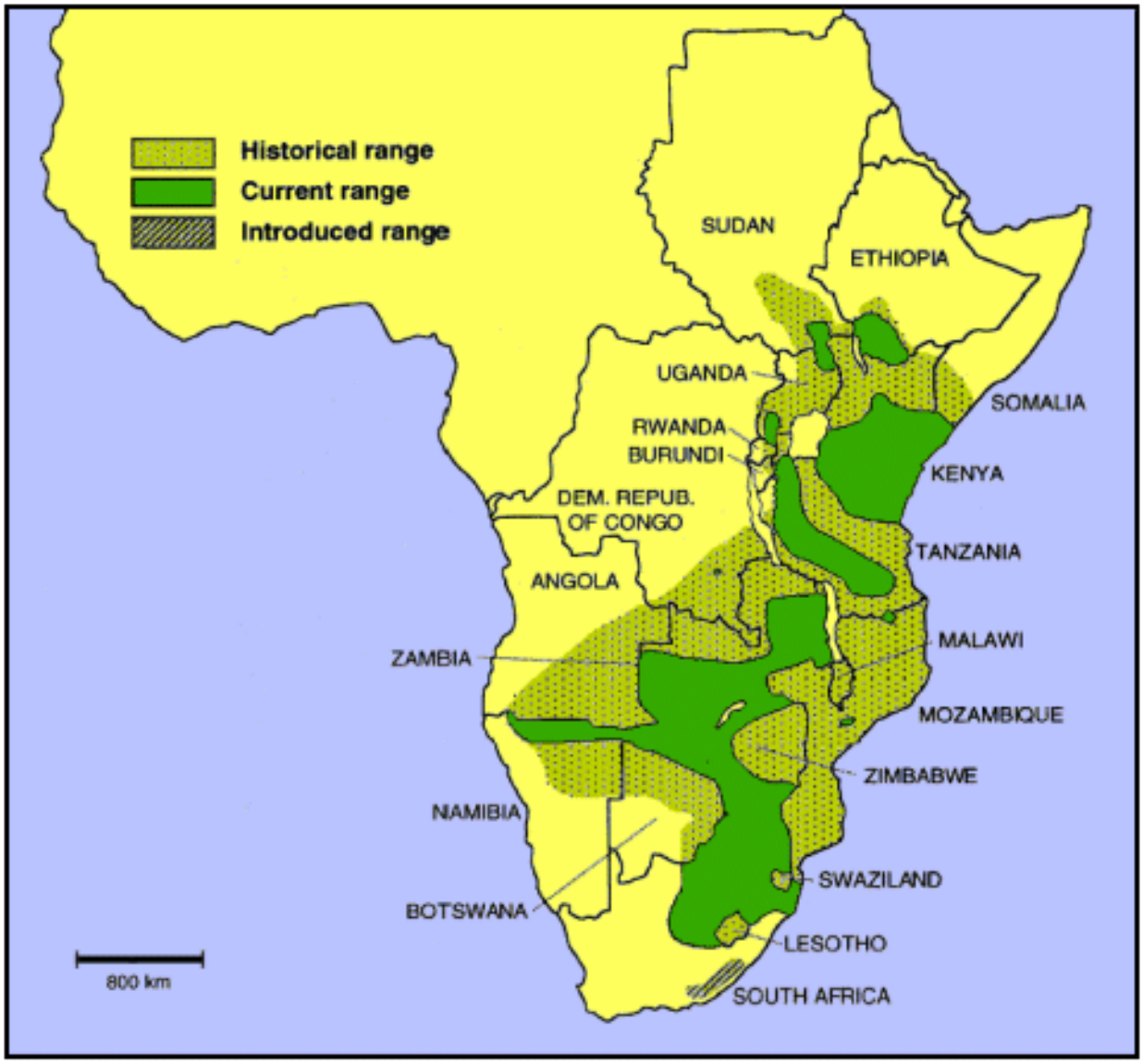
Behaviorally, Grevy's differ from all other zebras in that the social tendencies of Grevy's zebra is more akin to the ass (*Asinus spp*.) than any of its species counterparts. Much like the ass, Grevy's zebras do not form permanent social bonds, only temporary ones. The most common temporary bond being between mares and foals (Moelmann, 2002). Often multiple mare-foal units join together to form herds during foaling season. These aggregations usually include fewer than ten mares, only last until foals reach maturity, and do not have any dominance hierarchy. Once the group has disbanded, the mare sets out to find another mate.

Males either occupy their own territory or become part of a bachelor group. Bachelor groups consist of young free-ranging males that are not yet physically capable of securing their own territory. Like mare-foal herds, these groups have no dominance hierarchy and constantly change as more zebras enter or leave the herd (Kingdon, 1979). Often, this group will wander into areas occupied by resident male stallions in search of water or food. Males in these territories take no issue with the bachelor group and will even seek out their company at times, that is until a female in estrus is nearby. In this case, stallions will aggressively chase off all other males and try to mate with the female and keep her from leaving his area (Smithsonian). Once a male is old enough, he will establish his own territory.

Although Grevy's herds are typically small — about 6-20 individuals, there have been reports of large herds ranging anywhere from 200-800 individuals (Klingel, 1974b; Sidney, 1965). Grevy's can also form mixed herds with other species and typically do so with Bohm zebras (*E. quagga bohmi*) (Churcher, 1993). These herds usually contain roughly 5-14 Grevy's zebras with the majority of them being Bohm zebras (Keast, 1965). As is the case with the other groups, Grevy's tend not to stay in these herds for long periods of time.

Plains zebra

Whereas Grevy's zebra are restricted to the arid grassland landscapes of Kenya and Ethiopia, plains zebras (*E. quagga*) roam much of southern and eastern Africa. Their habitat is generally, but not exclusively, treeless grasslands and savanna woodlands, both tropical and temperate. They tend to avoid deserts, dense rainforest, and permanent wetlands and prefer Acacieae woodlands over Commiphora. Plains are also capable of living at higher elevations in areas like Mount Kenya, while Grevy's range in lower elevations (Estes R., 1991; Grub, P., 1981; Skinner J. D. et. al, 2006; IUCN). Current information on population size is limited but 2002 estimates put total numbers of plains at 660,000 (Hack et al. 2002).



Distribution of plains zebra across the African continent (http://www.equids.org/pzebra.php)

Physiologically, the plains zebra is mid-sized and thick bodied with relatively short legs. They are boldly striped in black and white, and no two individuals look exactly alike. Although they share the same color palette with their species counterparts, plains have a noticeably different striping pattern. While Grevy's have close-set narrow stripes that stay in relatively the same orientation throughout their body, plains have broader stripes that change from vertical to more horizontal as you look from their head to their hindquarters. Apart from these minor differences, Grevy's and plains are physiologically very similar. Both have adapted a set of specialized teeth and gut that allow them to extract nutrients from tough African grasses. They have the same eating habits, spending large parts of their day grazing. They also have similar speed, eyesight, and hearing, which they use to avoid their mutual predators: the lion, spotted hyena, wild dog, and cheetah.

Unlike Grevy's, the plains zebra is highly social and usually forms small family groups consisting of a single stallion, one or several mares, and their recent offspring. The adult membership of a harem is highly stable, typically remaining together for many years. A stallion forms a harem by abducting young mares from their families and mating with them to produce foals. In order to procure the mare, stallions must defeat her father as well as any other rival males seeking her courtship. Upon victory, the stallion will include the mare into his harem. If the harem already contains mares, it can be a delicate process to incorporate the mare into the group. Often new mares are met with hostility by preexisting mares and the stallion must shield them from the others aggression until it subsides. Once accepted, new mares join the female hierarchy within the harem.

The average family size varies considerably by region, but in general each family contains around 4-8 individuals. The biggest family group ever recorded was found in Ngorongoro and consisted of 16 members. As mentioned before, adult membership is highly stable, although rarely mares will change from one group to another. Typically, mares exclusively stay with one group, not communicating at all with members of other groups. Stallions on the other hand will contact other stallions in the vicinity and greet them in a set ceremony while foals play occasionally with foals of nearby families.

Turnover in a harem occurs with the departure and birth of foals. Young stallions leave their groups at the age of 1 to about 4 & 1/2 years. This process has no direct relation to their sexual maturity and there is no antagonism between them and their father or other members of the group. The timing depends on several factors: (a) the young stallion's mother has a new foal and thus the bonds between them are loosened. If the new foal dies, the original relationship is re-established. (b) There are no other male foals of about the same age in the groups and there is thus a lack of playmates for the fighting games typical of that age group. (c) There are stallion groups with such playmates nearby. If all these factors coincide, the young stallions leave their families at the age of a little over one year. Otherwise they may stay up to around 4 & 1/2 years.

After leaving their families the young stallions join bachelor groups. These groups are more variable in their composition, but personal bonds between the members are equally evident. Many of the groups consist of the same individuals for a few years. However, other stallions do often join them for short periods of time.

Data collection and tracking

This project is a collaboration with Dr. Blair Costelloe, Ben Koger, and multiple other members of the Max Planck Institute for Animal Behavior. Zebras were filmed in Laikipia County, Kenya. Videos were taken using off the shelf DJI phantom drones flying at a height of 80 meters. A relay of multiple drones were used to extend the length of the observations beyond the battery life of a single drone. Just before the first drone ran out of battery, the next drone was already hovering above the group filming. The drones followed the herds as they moved throughout the landscape in order to track all members of the herd simultaneously. A senseFly eBee fixed wing drone was used to

create a detailed (1 pixel is approximately a few centimeters) 3D map of the landscape the zebra groups were walking in.

Tracking was a two step process. First, a fine tuned model of faster rcnn with a resnet101 backbone was run on down sampled 1080x2048 video frames. This model was used to identify four classes: buffalo, gazelle, zebra, and all animals that were too close together to detect independently. Very close individuals were then cropped out of the full 2160x4096 resolution frame and fed into a retinanet model trained only to distinguish locations of close individuals. After classification, 3D positions of zebra were extracted. This was done using a 3D map of the underlying landscape. The first step in creating this map was to extract spatially overlapping video frames from each video taken by the senseFly eBee drone. The resulting map was then geo-referenced using the drones on-board GPS sensor. Geo-referencing made it possible to assign GPS locations to ground features in the map and have access to a camera matrix. The now geo-referenced map was then manually overlaid with a map generated using still video frames. The drone camera location and orientation was estimated for every frame of the observation video allowing us to project the location of every zebra from its position in the video frame to its location on the earth in terms of its latitude, longitude, elevation.

Spatial Scale

The UTM (Universal Transverse Mercator) coordinates of each individual are stored as 3D numpy arrays (individual id, number of frames, xy positions). The positions of each animal were processed using Python. The data was transformed from a temporal scale to a spatial one. Meaning instead of performing calculations on zebras in every frame, they were performed after the zebras had traveled a certain distance (step size). In this study, I used 5 meter and 10 meter step sizes. Spatial positions were extracted from the temporal data by drawing circles around each individual with 5 and 10 meter radiuses, recording only the positions that immediately fall outside the circle. A circle is then drawn around that point and the same process is repeated for the full length of the video. Since individuals move at different rates a time index was extracted for each spatial point pulled i.e. the frame number that the spatial point was pulled. When focusing on a specific individual, this will allow us to extract the locations of all the other individuals within the group at that particular individual's spatial scale. The spatial positions as well as the time index are used to calculate the features. From this point when the positions of the animal are mentioned it is inferred that these positions adhere to the spatial scale. See the appendix for details and code on the spatial distance function ("spatial_dis").

Alternative Steps

As mentioned previously, the step selection function pairs chosen locations by the individual with a number of randomly generated alternative steps. We decided to pair each step event with 5 alternative random steps. These random steps were calculated using the turning angle distributions and step sizes of each individual. The step sizes were fixed at either 5 or 10 meters. The turning angles were calculated by taking the

difference between consecutive step directions. That is, the difference between the vector pointing from the current position to the next step and the vector pointing from the previous step to the current location. Turning angles were calculated for each individual within an observation for every step event. To generate alternative steps for a particular individual, the following formulas were used

$$x = random\ step(cosine(turning\ angle) + current\ location$$
$$y = random\ step(sine(turning\ angle) + current\ location$$

The turning angles were randomly drawn from the distribution of turning angles calculated for each individual through the observation. 5 alternative steps were calculated for every step event for every individual for every observation. These real and random step pairs were used to build the step selection model. See the appendix for code details on random step generation (function titled: “random_steps”).

Group center

This feature is used to assess how important it is for zebras to position themselves at the center of the group. If they are concerned about putting themselves there they will constantly change their direction to move towards the group center. This was calculated by taking the difference between the vector pointing from an individual’s current location to its successive location and the vector pointing from the current location to the center of the group. The shortest angle between vectors was always taken. This was done on the assumption that the individual would take the path of shortest distance to head towards the group center. This data was paired with the difference between the vector pointing from the current location of the individual to each one of its alternative steps and the vector pointing from the current location to the center of the group. So for each step event there are six angles calculated. All angles fed into the step selection model were positive. All angles fed into the model are then between 0 and pi radians (180 degrees). All values associated with chosen locations were labeled with a 1 and the alternative steps a 0. They were also labeled with an individual id and step event id. See the appendix for details on the function (“dc_feature”).

Social Density

This feature assesses individuals propensity to want to be close with each other. Circles of radius sizes 1, 5, 10, 15, and 20 meters were drawn around an individual at every step event. The number of conspecifics that fell within that circle at that step event were then counted. The number of individuals within the circle were divided by the number of individuals present at that particular step event (excluding the focal individual). The result is the fraction of conspecifics within a given radius of a focal individual at each step event. This was performed for every individual and each other their step events. Circles with the same radius sizes were also drawn around all 5 of the focal individuals alternative steps at that step event. The number of conspecifics within each of these circles were calculated and again the fraction of individuals was taken. All 5 fractions were treated separately. The result is 6 values between 0 and 1 for each step event for

each individual. This was done across each observation. All values associated with chosen locations were labeled with a 1 and the alternative steps a 0. They were also labeled with an individual id and step event id. These values were fed into the model. See the appendix for details on the function ("sd_feature").

Recently used space

This feature assesses how important following other individuals is. It counts the number of conspecifics that have occupied a potential location within the last 2 minutes. A zebra is considered to have occupied the location if it was within 1, 5, 10, 15, or 20 meters. Circles were drawn around an individuals location at a particular point in time with 1, 5, 10, 15, or 20 meter radius sizes. The locations of all other individuals over the span of two minutes are checked to see if they fall within that particular location. To find the fraction of individuals that passed through this space, the number of individuals within the circle over the two minute time period were divided by the number of conspecifics that were present during the two minute period (not including the focal individual). Individuals were never counted more than once. This was done for each step for an individual. So at each spatial step I am looking back through two minutes of video. Thus, this feature incorporates a temporal and spatial element. The same process is used for each alternative step at an individuals step event. Circles with radiuses of 1, 5, 10, 15, or 20 were drawn around each of the 5 steps and the number of conspecifics to pass through these circles in a 2 min time period were counted (each individual again counted up to one time). The fraction of individuals that passed through the circle at every step event was then calculated. The result is 6 values between 0 and 1 for each step event for each individual. This was done across each observation. All values associated with chosen locations were labeled with a 1 and the alternative steps a 0. They were also labeled with an individual id and step event id. These values were fed into the model. See the appendix for details on the function ("ru_feature").

Game Trails

A major feature of the environment we work on is the many game trails weaving across the landscape. These trails are the result of many animals, potentially of many different species, moving over the same areas just as many humans walking across a field form a dirt path. Once these trails are established they provide ways of traveling through the landscape that are physically easier. Additionally, since many animals have similar needs—getting water for example—the fact that a trail has established in the first place may be a clue that there is something of value at the end of it. So in addition to easing movement, these trails could serve as a sort of long term shared memory of the landscape. Here we begin to investigate how game trails influence zebra movement.

Game trails a visible at the resolution of the landscape maps generated from the UAV imagery. While there is room for interpretation, many trails are obvious to a human observer. To quantify the trails in the landscape a human manually marked every trail they could see in each of four landscape maps. These trails were marked as one-dimensional vectors drawn along the midpoint of the paths. This means there is no

specific width defined for each trail. It would have been to time costly to specify the exact width. A single width is estimated for all trails. At every position on the map the trail is either fully present (1) or absent (0). While we investigated deep learning approaches which will allow us to generalize to all maps and define a gradient of the strength of a given trail, that remains future work. With the binary manual annotation, however, we can determine for each zebra step whether or not it on a game trail. Because the deep learning approach ended up being too ambitious for the time constraints of this thesis we are only looking at the effect of game trails on the four observations with human annotated maps.

Probability

Step selection modeling is useful for understanding how many social and environmental variables combine to influence an animals movement decisions. A drawback, however, is limited interpretability. Here, following Strandburg-Peshkin et al. 2017, we calculate more explicitly the probability of an animal stepping towards or away from the various social and environmental features described above. In this case, however, we only investigate one feature at a time. Every real step the animal takes is paired with a random step that begins at the same location as the real step but in a direction drawn from the animal's overall step distribution. The social or environmental feature is measured at each of these resulting positions. The difference between the feature value of the step taken minus the value at the step not taken is computed. A positive value corresponds to the feature being stronger at the end of the step taken while a negative difference means the measured feature was stronger where the animal didn't go. Assuming that the generated step was as reachable as where the animal actually went, the assumption is that the animal could have gone to either location but chose (maybe randomly) to go where it did instead of the alternative location.

For instance, imagine investigating social density: At the end of the real step there are three other zebras. At the end of the alternative step there is only one. The difference between these two locations is two. In another instance there are two zebras at the end of the real step and four at the end of the alternative step. The difference between these two locations is negative two. These two examples are a pair: in each case the zebra had the option of going to where there were two more zebras than the alternative. In one case it went towards the density, in another it didn't. In 50% of the cases where there was difference of two zebras the focal zebra went towards the higher density.

In general the probability is the proportion of times the focal individual went towards a given condition divided by the total number of chances it had to go towards that condition. See the appendix for details on the function used to calculate this ("calculate_probability").

Step selection function

When implementing this model there are four main elements to consider: the fix rate, whether to create alternative steps drawing from empirical distributions of individuals or entire populations, the number of alternative steps, and the features.

The fix rate is the frequency of sampling between successive locations and can be measured on both spatial and temporal scales (e.g. the GPS sampling rate for a tracked animal over time). Selecting the appropriate scale is dependent on the research question and the ecology of the target species. A smaller step size will capture more local fine scale movements decisions while larger movement scales capture more macro movements. It often makes sense to try a range of step sizes to see what drives movement decision at different scales. An improper fix rate can result in misleading interpretations about the key drivers of animal movement so it is crucial that the fix rate is selected carefully. To avoid making observations on the wrong scale, it is best to perform pilot studies initially with as high fix rates as possible followed by implementing models using successively lower fix rates (Thurfjell et al., 2014).

Equally as important to consider is whether to draw alternative steps from empirical distributions of individuals or entire populations. As mentioned before, alternative steps are used in an effort to characterize what is 'available' to the animal and are drawn at random from a distribution of step lengths and turning angles. The key difference between utilizing population level distributions as opposed to individual level distributions is the assumption that the characteristics you are looking at are the same across all the animals.

When choosing the number of alternative steps to generate it is important to consider computational time (i.e. how long it will take to run the step selection analyses). If sample size is relatively large, then a large number of alternative steps can make the size of the database excessive, resulting in higher computational time. Because most datasets generated using GPS fall into this category, where there are often thousands of locations recorded per individual, it may be better computational time wise to use a low number of alternative steps.

The last component of step selection functions to consider is the features. These are the social and environmental factors suspected to have an influence on animal movement. A thorough understanding of the ecology of the species as well as the study goals are necessary to evaluate which attributes should be considered. How you quantify these features is important, as some methods may better reflect the perception of the animal than others.

The equation for the step selection function is as follows:

$$w(x) = e^{\beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p} \quad (1)$$

Here $w(x)$ refers to the animals preference for a given location p when asked about characteristics about that location $x_p = x_{1p}, x_{2p}, x_{np}$. The values from group center,

social density, recently used space, and game trails were used as the x values in this function. Conditional logistic regression is used to ascertain the β values that correspond to each input vector of x values.

The probability of choosing a given location out of all potential locations is given by:

$$P(S_{i,j} = 1 \mid \sum_{j=1}^{n} S_{i,j} = 1) = \frac{w(x_{i,j})}{w_{(i,1)} + w_{(i,2)} + \ldots + w_{(i,j)}} = \frac{e^{\beta x_{i,j}}}{e^{\beta x_{i,1}} + e^{\beta x_{i,2}} + \ldots + e^{\beta x_{i,j}}}$$

(2)

where the denominator is a normalization factor such that the probabilities over all potential locations within a given set equals 1. In the code included in the appendix of this thesis, this object is called the “conditional_logistic_regression”.

V. RESULTS

Step selection model output

A) Social density

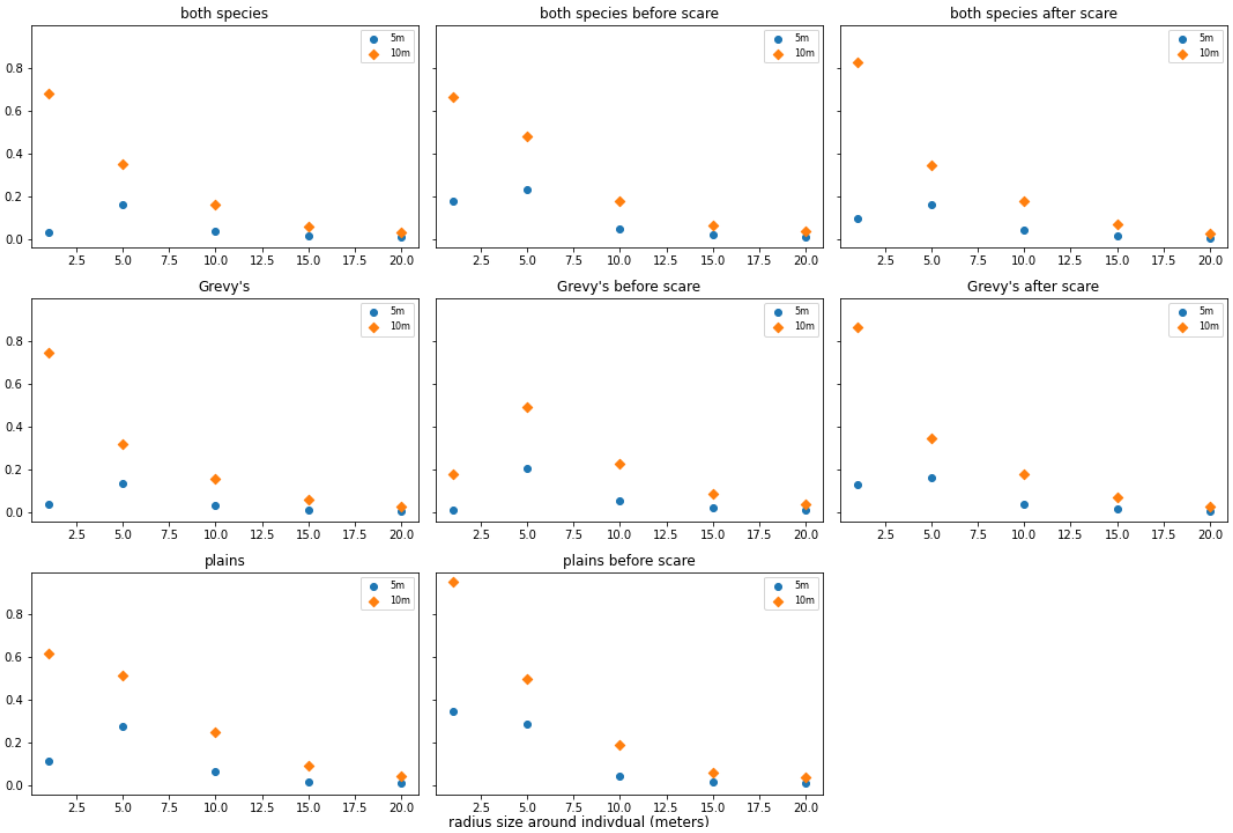


Figure 1. The plots above show the step selection function run with only the social density feature data. The y-axis for each of the plots corresponds to the beta value of the step selection function. This corresponds to the likelihood the that feature is having an affect on movement. The x-axis for each plot corresponds to the radius sizes of the circles draw around focal individuals. Blue points represent the output for a step size of 5 meters, the orange diamonds represent the output for a step size of 10 meters.

For most analyses at both the genus and species level, at the 5 meter step size, beta values mostly stay close to 0 for the 1, 10, 15, and 20 meter circle radius sizes with exceptions being for 'both species before scare' and 'plains before scare' plots. Here beta values for the 1 meter radius sizes are a bit higher, corresponding to ~0.2 and and ~0.35 respectively. There is also a slight exception for the 'both species after scare', 'Grevy's after scare', and 'plains' plots at the 1 meter radius size. Here beta values are ~0.1. At the 5 meter radius sizes for each plot, there is a noticeable spike in the beta values. These values range from ~0.1 to ~0.3.

At the 10 meter step size, all the plots generally follow the same trend, starting with beta values between ~0.6 to ~0.9 for the 1 meter radius size, ~0.3 to ~0.5 for the 5 meter

radius size, ~0.2 for the 10 meter radius size, ~0.1 for the 15 meter radius size, and 0 for the 20 meter radius size. The exception is at the 1 meter radius size for the ‘Grevy’s before scare’ plot where the beta value is ~0.2 instead of between ~0.6 and ~0.9.

B) Recently Used Space

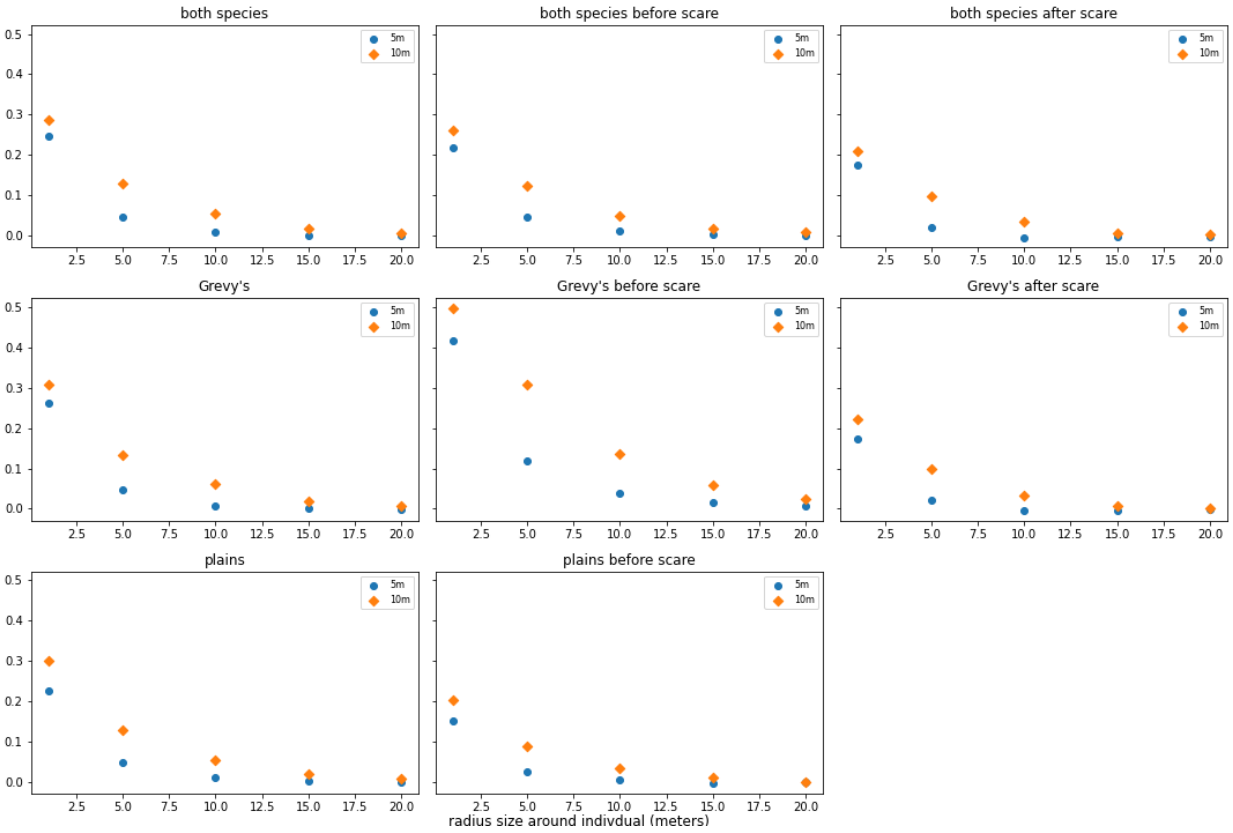


Figure 2. The plots above show the step selection function run with only the recently used space feature data. The y-axis for each of the plots corresponds to the beta value of the step selection function. This corresponds to the likelihood that the feature is having an affect on movement. The x-axis for each plot corresponds to the radius sizes of the circles draw around focal individuals. Blue points represent the output for a step size of 5 meters, the orange diamonds represent the output for a step size of 10 meters.

All plots generally follow the same downward trend in beta values for both step sizes. For the 5 meter step size beta values range from ~0.15 to ~0.25 for the 1 meter radius size, ~0.01 to ~0.05 for the 5 meter radius size, and 0 to ~0.01 for the 10, 15 and 20 radius sizes. There is an exception for the 'Grevy's before scare' plot at the 1, 5, and 10 meter radius sizes where beta values are nearly double. For the 10 meter step size beta values range from ~0.2 to 0.3 at the 1 meter radius size, ~0.1 for the 5 meter radius size, ~0.05 for the 10 meter radius size, and between ~0.01 for the 15 and 20 meter radius sizes. Again the exception lies within the 'Grevy's before scare' plot with all values being nearly double of what they were in the other plots at the 1, 5, 10, and 15 meter step size.

C) Group Center

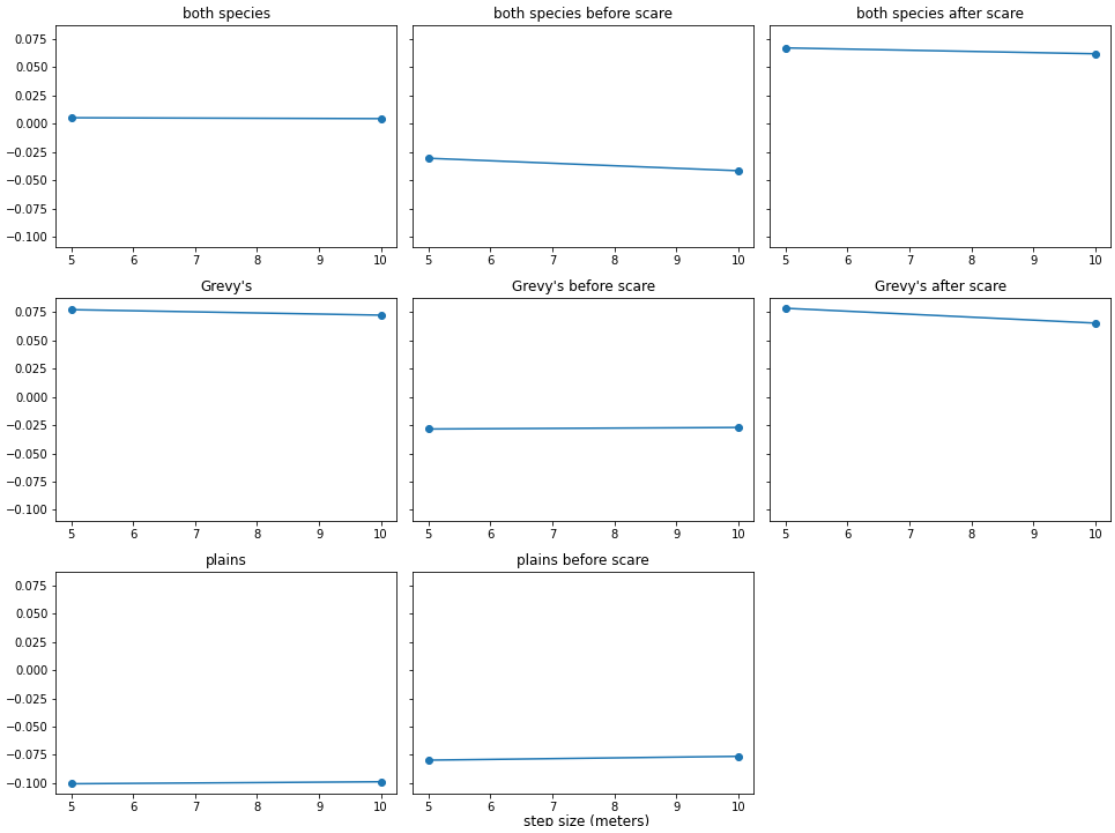


Figure 3. The plots above show the step selection function run with only the group center feature data. The y-axis for each of the plots corresponds to the beta value of the step selection function. This corresponds to the likelihood that the feature is having an affect on movement. The x-axis for each plot corresponds to each of the step sizes tested.

For all the plots the beta values between the 5 and 10 meter step sizes are very similar. In the 'both species' and 'Grevy's before scare' plots the beta values between the two step sizes are exactly the same. 'Both species' beta values are 0 for both the 5 and 10 meter step sizes and ~ -0.25 for 'Grevy's before scare'. As for the 'both species before scare' plot beta values, the 10 meter step size has a value of ~ 0.05 and the 5 meter step size ~ -0.025. 'Both species after scare', 'Grevy's', and 'Grevy's after scare' plots have a beta value of ~ 0.075 at the 5 meter step size and a slightly lower beta value (~0.070) at the 10 meter step size. The 'plains' beta values at the 5 and 10 meter step sizes are ~ -0.1 and ~ -0.075 for the 'plains before scare' plot.

Probability

A) Social Density

i) Both Species

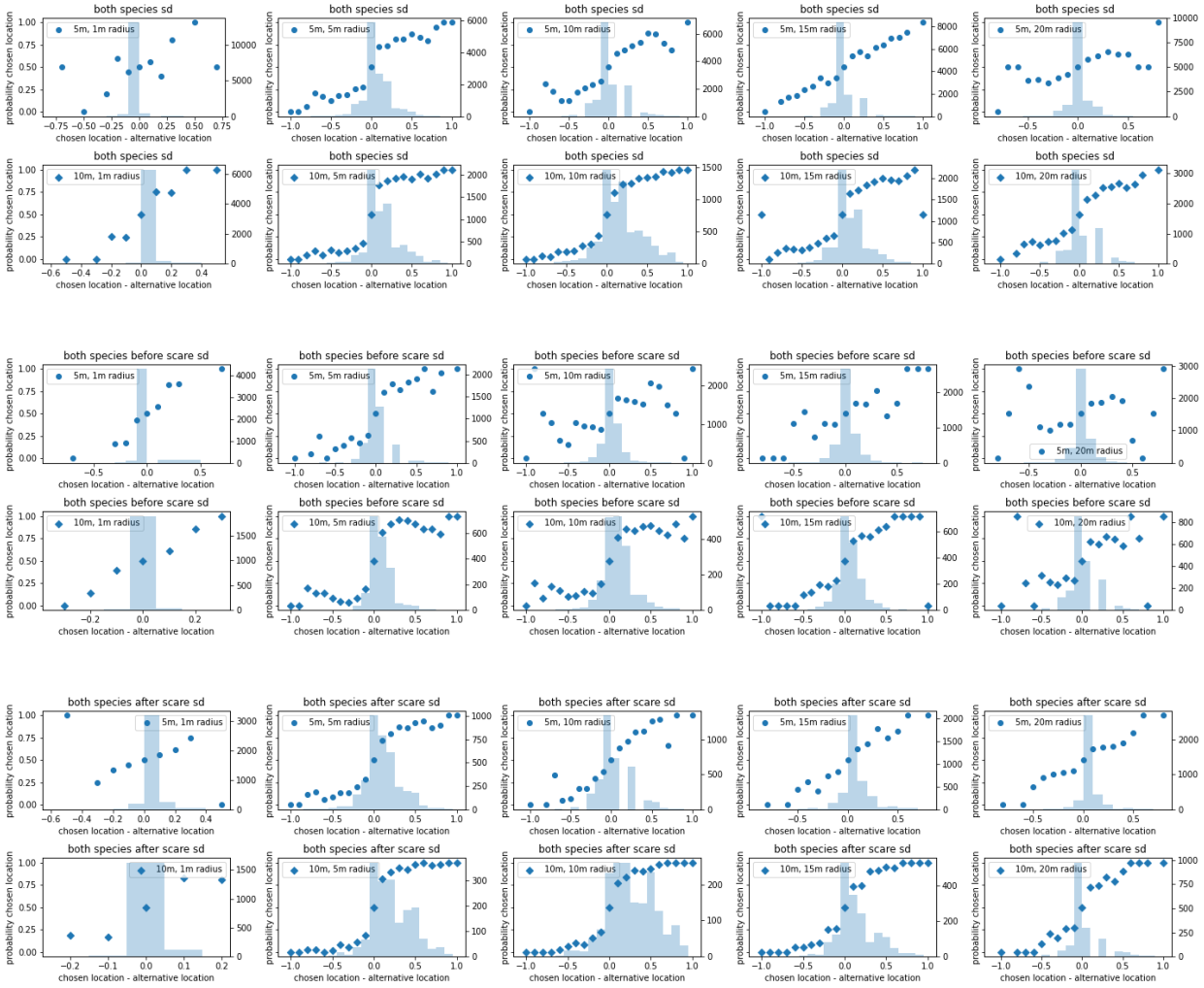
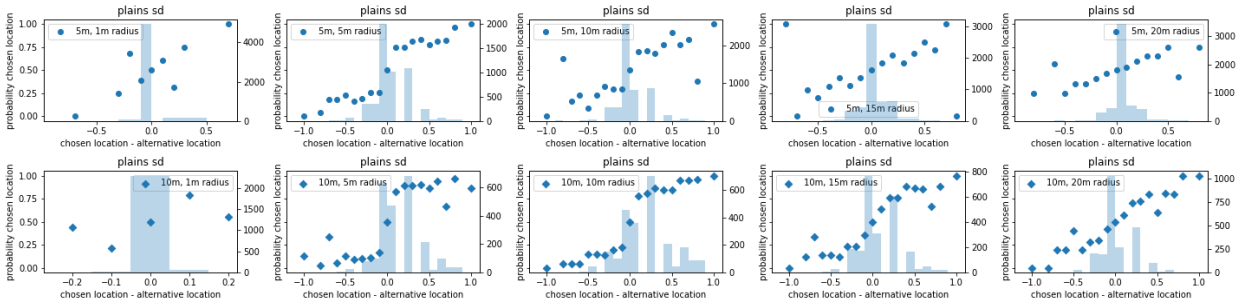


Figure 4. The scatter plot represents the probability that an individual will select a chosen or true location given the choice between a chosen location and one randomly selected alternative (not observed in the field) location. This choice was made by looking at the difference between the fraction of individuals that are within a 1, 5, 10, 15, and 20 meter radius of the chosen location and alternative location. These values are shown on the x-axis of each plot. The underlying histogram shows the number of number of counts for each difference event, thus there is a bin for every scatter point. The left y-axis on each plot is the probability of choosing the chosen location, the right y-axis corresponds to the number of counts for each difference event. For each resolution (both species, both species before scare, and both species after scare) there are 10 figures, 5 of which display the results with a 5 meter step size at 1, 5, 10, 15, and 20 meter radius sizes. These plots are designated by circle scatter points. The other 5 plots display the 5 radius sizes using a 10 meter step size.

The general trend in the data across all plots is sigmoidal, where the chosen location selected by a zebra is likely to be the one that a higher fraction of conspecifics are occupying. The probability increases as the difference in the fraction of individuals that occupy either the chosen or alternative location increases. Another trend can be seen as you look across the 1, 5, 10, 15, and 20 meter radius sizes. Generally, at the 1 meter radius size there are fewer probability events and the data is less sigmoidal. As you move to the 5 and 10 meter radius sizes, there are more probability events, less difference event occurrences, and the data are highly sigmoidal. At the 15 and 20 meter radius sizes, the data starts to become less ordered and less sigmoidal, with probabilities in both positive and negative difference events heading towards 0.5. Also, the distribution of difference event occurrences is restricted to a smaller range at the 20 meter radius sizes compared to the other radius sizes. As for the comparison across step sizes. There is a general trend for the probabilities to be more sigmoidal in the 10 meter step size and a decrease in the number of difference event occurrences. When comparing across no scare event consider, before a scare event, and after a scare event you see a that at the 5 meter step size the plots for no scare event considered and the after the scare event are similar. The same trend does not appear in the 5 meter step size plots for before a scare event, where the figures are less sigmoidal and contain fewer probability events and difference event occurrences.

ii) Plains



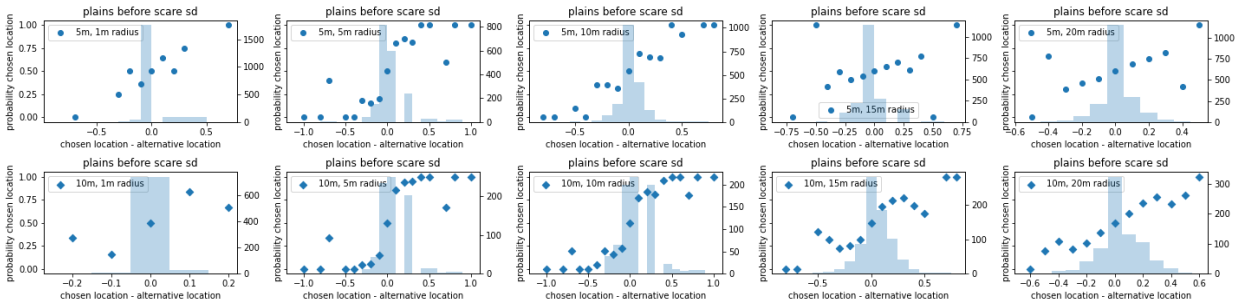


Figure 5. The scatter plot represents the probability that an individual will select a chosen or true location given the choice between a chosen location and one randomly selected alternative (not observed in the field) location. This choice was made by looking at the difference between the fraction of individuals that are within a 1, 5, 10, 15, and 20 meter radius of the chosen location and alternative location. These values are shown on the x-axis of each plot. The underlying histogram shows the number of counts for each difference event, thus there is a bin for every scatter point. The left y-axis on each plot is the probability of choosing the chosen location, the right y-axis corresponds to the number of counts for each difference event. For each resolution (plains and plains before scare) there are 10 figures, 5 of which display the results with a 5 meter step size at 1, 5, 10, 15, and 20 meter radius sizes. These plots are designated by circle scatter points. The other 5 plots display the 5 radius sizes using a 10 meter step size.

Similar to the results on the genus level all plots follow a general sigmoidal trend, where the likelihood of a zebra selecting a chosen location increases as the fraction of conspecifics within the chosen location increases. The after scare event analysis is missing due to the lack of data for this event. As you look across the 1, 5, 10, 15 and 20 meter radius sizes, the trend in probability starts out being less sigmoidal, then becomes more sigmoidal and finally in the 15 and 20 meter radius sizes less sigmoidal again. There is also a change in the distribution range for the difference events. The range starts out small for the 1 meter radius size, increases at the 5, 10, and sometimes 15 meter radius sizes, then shrinks again for the 20 meter radius size. When comparing across step sizes the probability plots tend to me more ordered and sigmoidal at the 10 meter step size. The number of difference event occurrences decrease when moving from the 5 to 10 meter step size with the exception being for the ‘plains sd’ 5 and 10 meter plots. Here the number of difference event occurrences stays the same. When comparing across ‘plains’ and ‘plains before scare’ the probabilities follow the same sigmoidal trend and fall between 0 and 1. The total number of difference event occurrences decreases from ‘plains’ to ‘plains before scare’. The range of difference events at both step sizes before the scare event are smaller then the ones for ‘plains’

iii) Grevy's

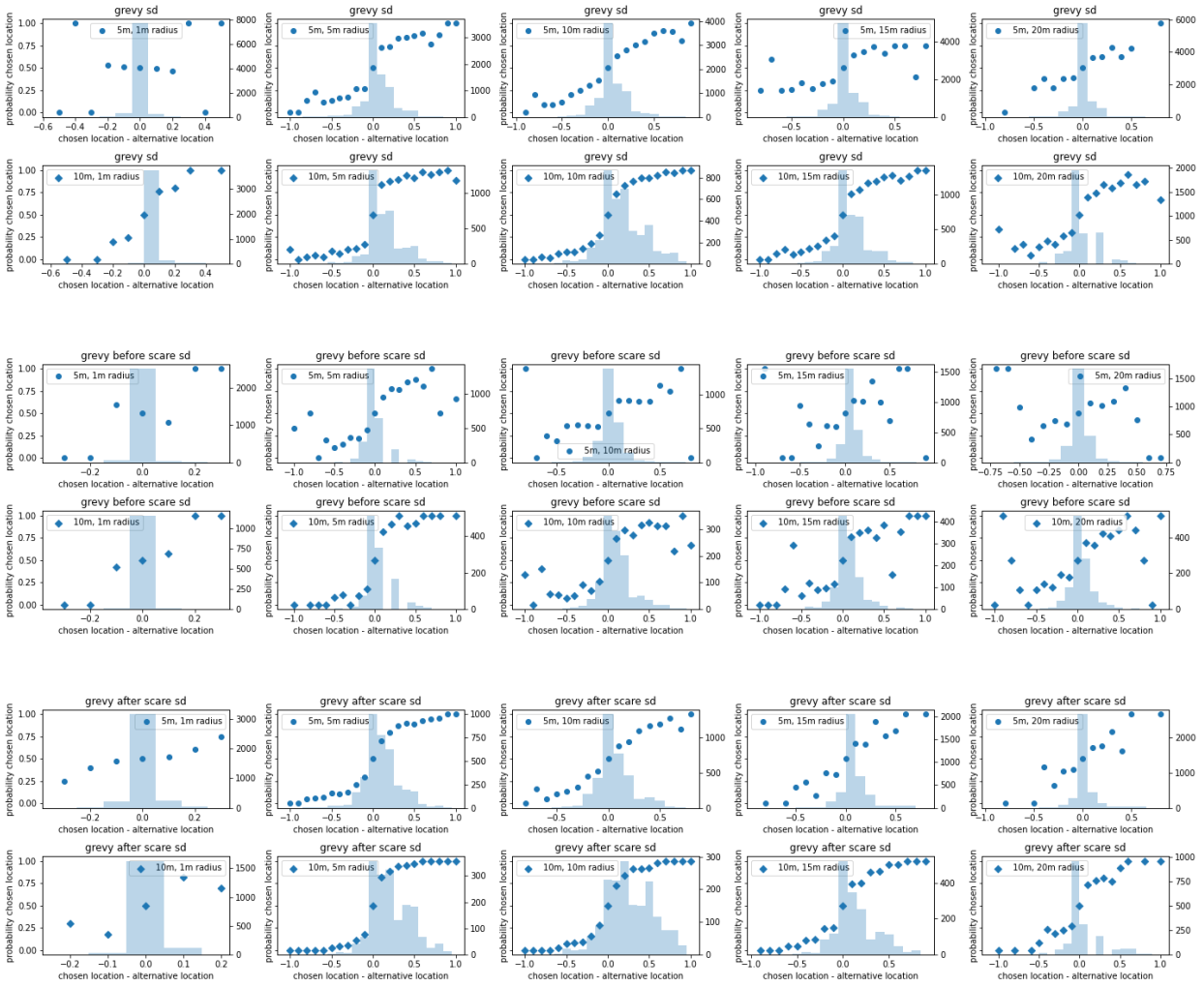


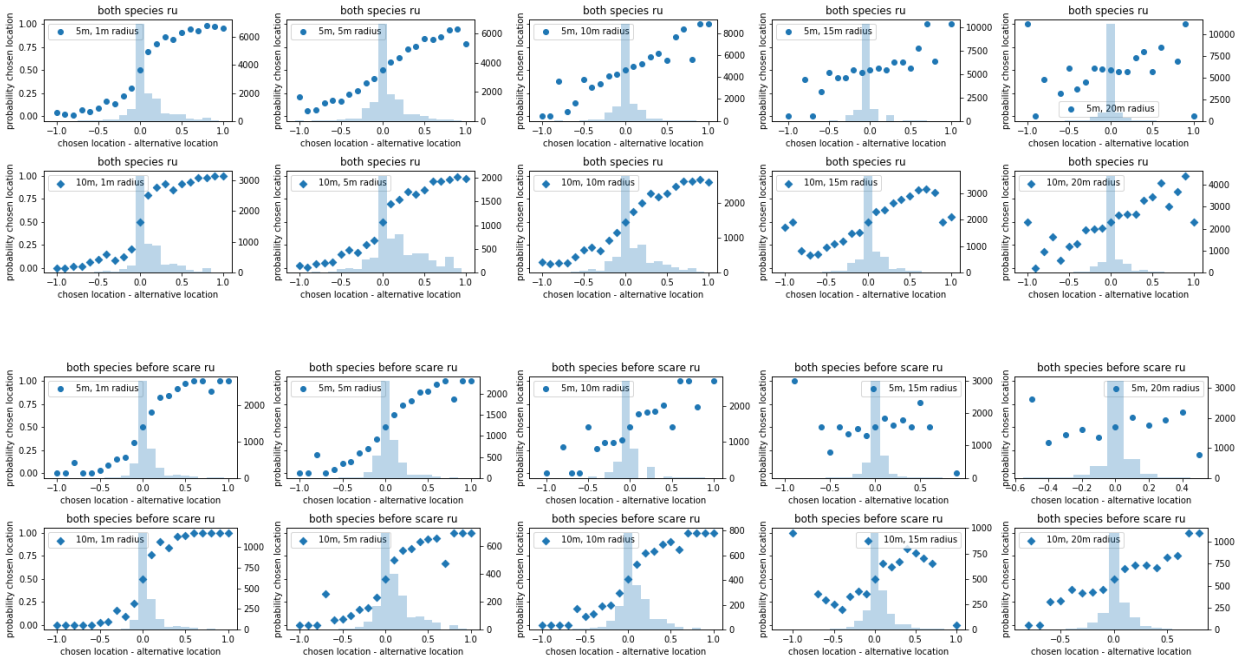
Figure 6. The scatter plot represents the probability that an individual will select a chosen or true location given the choice between a chosen location and one randomly selected alternative (not observed in the field) location. This choice was made by looking at the difference between the fraction of individuals that are within a 1, 5, 10, 15, and 20 meter radius of the chosen location and alternative location. These values are shown on the x-axis of each plot. The underlying histogram shows the number of counts for each difference event, thus there is a bin for every scatter point. The left y-axis on each plot is the probability of choosing the chosen location, the right y-axis corresponds to the number of counts for each difference event. For each resolution (Grevy's, Grevy's before scare, and Grevy's after scare) there are 10 figures, 5 of which display the results with a 5 meter step size at 1, 5, 10, 15, and 20 meter radius sizes. These plots are designated by circle scatter points. The other 5 plots display the 5 radius sizes using a 10 meter step size.

Comparing across radius sizes for all plots the shape of the probability values generally starts out loosely sigmoidal at the 1 meter radius size, becomes very sigmoidal at the 5 meter radius size, and starts to loose this shape in the 10, 15 and 20 meter radius sizes. However, there are a few exceptions to this trend, specifically at the 'Grevy's' and 'Grevy's before scare' plots. At the 'Grevy's' plot, the probability curve values are linear

at the 0.5 whereas there is a negative relationship between probability values and difference events in the ‘Grevy’s before scare’ plot. For the histogram showing the distribution of difference event occurrences, there are a higher number of total difference event occurrences at the 1 and 20 meter radius sizes and a similar number of total difference event occurrences for the 5, 10, and 15 meter radius sizes. The range in difference events however is are larger for radius sizes 5, 10, and 15 and larger for radius sizes 1 and 20. Comparing across step sizes for all plots the shape of the probability curve is very slightly more sigmoidal at the 10 meter step sizes. The total number of difference event occurrences decreases from the 5 to 10 meter step size for all plots with the exception being for ‘Grevy’s after scare’ where there is an increase in the total number of difference event occurrences. The range for the difference events is the same across step sizes for each radius size. When comparing across ‘Grevy’s’ ‘Grevy’s before scare’, and ‘Grevy’s after scare’, trends in the probability plots hold through step sizes and radius sizes. This is also true for the trend in the range of difference events. However, there is a change in the total number of difference event occurrences. ‘Grevy’s’ have more total difference event occurrences than ‘Grevy’s before scare’, which have more than ‘Grevy’s after scare’.

B) Recently Used Space

i) Both Species



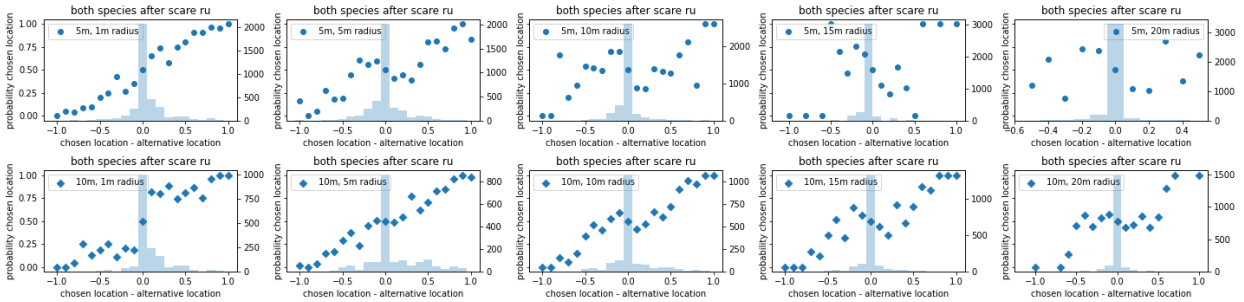


Figure 7. The scatter plot represents the probability that an individual will select a chosen or true location given the choice between a chosen location and one randomly selected alternative (not observed in the field) location. This choice was made by looking at the difference between the fraction of individuals that have occupied the chosen location and alternative location over a the last 2 minutes. These values are shown on the x-axis of each plot. The underlying histogram shows the number of counts for each difference event, thus there is a bin for every scatter point. The left y-axis on each plot is the probability of choosing the chosen location, the right y-axis corresponds to the number of counts for each difference event. For each resolution (both species, both species before scare, and both species after scare) there are 10 figures, 5 of which display the results with a 5 meter step size at 1, 5, 10, 15, and 20 meter radius sizes. These plots are designated by circle scatter points. The other 5 plots display the 5 radius sizes using a 10 meter step size.

Comparing across radius sizes for all plots the shape of the probability data starts out very sigmoidal at the 1 meter radius size and becomes less and less so for the 5, 10, and 15 meter radius sizes. Finally at the 20 meter radius size all the data hovers closely around 0.5 probability. This trend is more prominent at the 5 meter step size rather than the 10 meter step size. However, for ‘both species after scare’ the data trend shifts from sigmoidal to slightly sinodial with the probability starting around 0 at highly negative difference event values, then increases to around 0.5 probability before a difference event of 0, it then drops slightly again to a probability value of 0.4 only to steadily increase as the difference event value approaches 1. This trend can be seen for the 5 meter radius plot at the 5 meter step size and the 10 and 15 meter radius size plots for the 10 meter step size. At the 5 meter step size and 20 meter radius size, the curve is highly sinodial. Conversely, at the 5 meter step size and 15 meter radius size the core of the data shows high probabilities at negative difference events and low probabilities for positive difference events. When looking at the total number of difference event occurrences across radius sizes for all plots you will notice higher total occurrences at the 1 and 20 meter radius sizes than for radius sizes 5-15. Across step sizes, probability curves tend to be more ordered at the 10 meter radius size but have less total number of difference event occurrences. At the 5 and 10 meter step sizes the range of difference events across radius sizes is the same. As we discussed above when comparing across ‘both species’, ‘both species before scare’, and ‘both species after scare’, the plots for ‘both species’ and ‘both species before scare’ followed the same trends whereas ‘both species after scare’ exhibited different behavior at the 5 meter step size for the 10, 15 and 20 meter radius sizes. As for the total number of difference event occurrences, the before and after scare plots are the same across radius and step sizes, and are both less than ‘both species’ plots.

ii) Plains

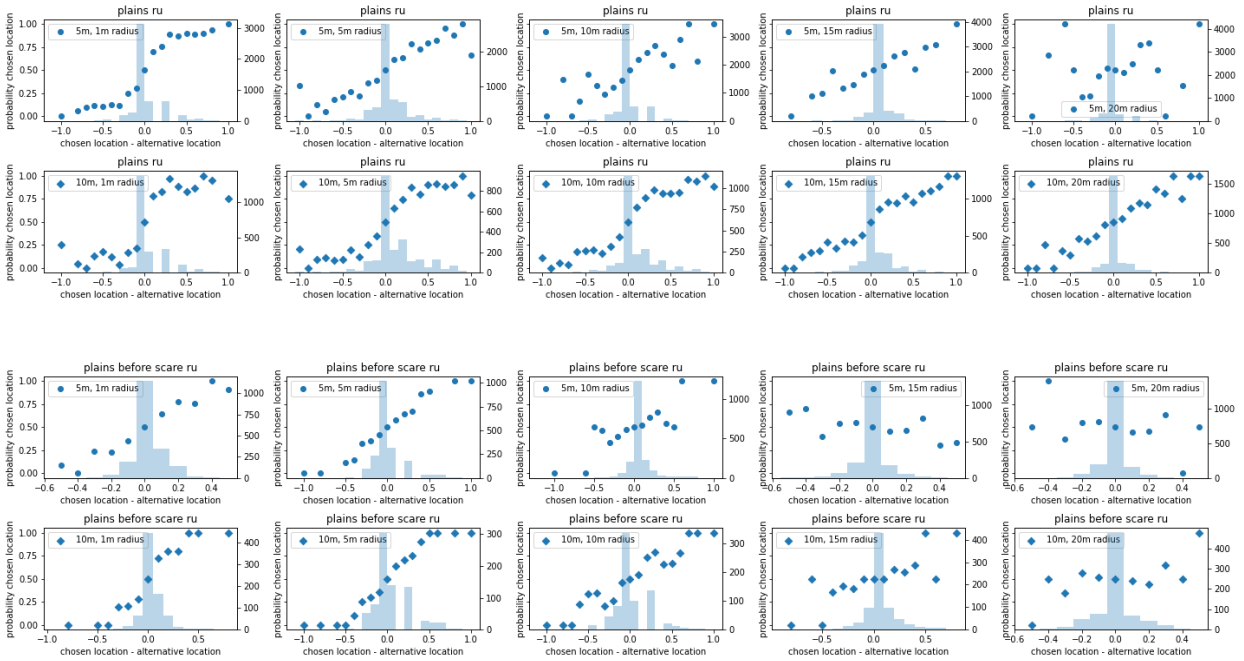


Figure 8. The scatter plot represents the probability that an individual will select a chosen or true location given the choice between a chosen location and one randomly selected alternative (not observed in the field) location. This choice was made by looking at the difference between the fraction of individuals that have occupied the chosen location and alternative location over a the last 2 minutes. These values are shown on the x-axis of each plot. The underlying histogram shows the number of counts for each difference event, thus there is a bin for every scatter point. The left y-axis on each plot is the probability of choosing the chosen location, the right y-axis corresponds to the number of counts for each difference event. For each resolution (plains and plains before scare) there are 10 figures, 5 of which display the results with a 5 meter step size at 1, 5, 10, 15, and 20 meter radius sizes. These plots are designated by circle scatter points. The other 5 plots display the 5 radius sizes using a 10 meter step size

The shape of the data for all the probability plots across radius sizes is generally linear throughout with slight sigmoidal tendencies at the edges. The exception is at the ‘plains’ 5 and 10 meter step size and 1 meter radius size where the data is highly sigmoidal. The number of probability points as well as the total number of difference event occurrences stays consistent across radius sizes. There is a slight difference in the range of difference events across radius sizes. As the radius size increases, the range of different events becomes slightly smaller. Across step sizes the shape and order of the probability plots stays consistent. However there is a noticeable decrease in the total number of difference event occurrences from the 5 to 10 meter step sizes. There is no major differences in the range of difference events. When comparing between ‘plains’ and ‘plains before scare’ trends hold consistent across step sizes and radius sizes with the exception of ‘plains before scare’ 5 meter step size, 10 meter radius size. The slope of the the probability curve in this plot is much less significant than any other. The reduction in slope carries through for the 15 and 20 meter radius sizes. When comparing histograms between ‘plains’ and ‘plains before scare’ you will notice a

decrease in the total number of difference event occurrences as well as a slight decrease in the range of difference events for ‘plains before scare’. There is no ‘plains after scare’ due to a lack of data.

iii) Grevy’s

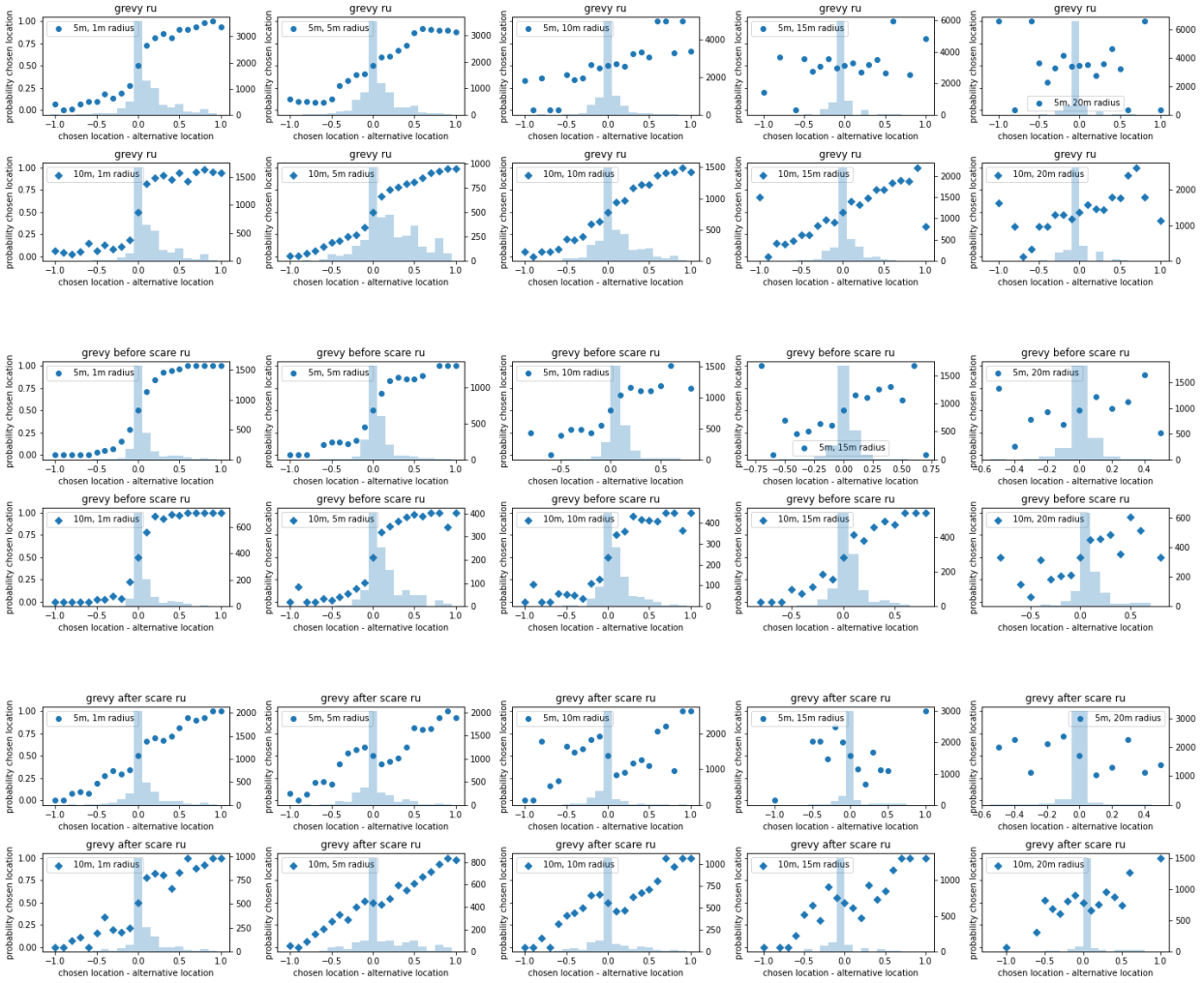


Figure 9. The scatter plot represents the probability that an individual will select a chosen or true location given the choice between a chosen location and one randomly selected alternative (not observed in the field) location. This choice was made by looking at the difference between the fraction of individuals that have occupied the chosen location and alternative location over a the last 2 minutes. These values are shown on the x-axis of each plot. The underlying histogram shows the number of counts for each difference event, thus there is a bin for every scatter point. The left y-axis on each plot is the probability of choosing the chosen location, the right y-axis corresponds to the number of counts for each difference event. For each resolution (Grevy’s, Grevy’s before scare, and Grevy’s after scare) there are 10 figures, 5 of which display the results with a 5 meter step size at 1, 5, 10, 15, and 20 meter radius sizes. These plots are designated by circle scatter points. The other 5 plots display the 5 radius sizes using a 10 meter step size.

When comparing ‘grevy’, ‘grevy before scare’, and ‘grevy after scare’ you will notice trends across radius sizes to be similar for the ‘grevy’ and ‘grevy before scare’. The probability data starts out sigmoidal and as the radius size around the individual gets larger, more linear with either a slope of zero (5 meter step size case) or roughly 1 (10 meter step size case). As for the plots in ‘grevy after scare’ there is almost a sinodial loop in the middle the data. This is most evident at the 5 meter step size, 5 meter radius size plot as well as the 10 meter step size, 10 meter radius plot. This artifact becomes more pronounced at the 5 meter step size, 10 and 20 meter radius size plots as well as the 10 meter step size, 15 and 20 meter radius plots. The data for the 5 meter step size, 15 meter radius size however is somewhat linear with a negative slope. If you look at the number of difference event occurrences across ‘grevy’, ‘grevy before scare’, and ‘grevy after scare’ you will notice an overall decrease in the total number of difference events.

C) Group center

i) Both species

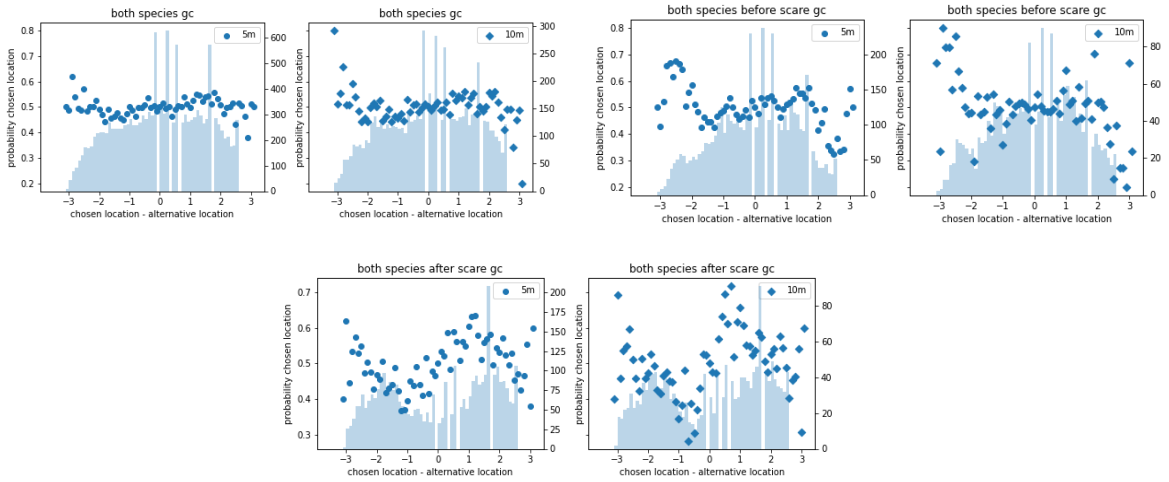


Figure 9. The scatter plot represents the probability that an individual will select a chosen or true location given the choice between a chosen location and one randomly selected alternative (not observed in the field) location. This choice was made by looking at the difference between two differences. The first difference is between the vector pointing from the animals current step to its next chosen location and the vector pointing from the current step and the center of the group. This value is termed chosen location. The second difference is between the vector pointing from the animals current step to an alternative location and the vector pointing from the current step to the group center. This value is term alternative location. The difference is then taken between these two difference values. These values are shown on the x-axis of each plot. The underlying histogram shows the number of counts for each difference event, thus there is a bin for every scatter point. The left y-axis on each plot is the probability of choosing the chosen location, the right y-axis corresponds to the number of counts for each difference event. For each resolution (both species, both species before scare, and both species after scare) there are 2 figures, one which displays the results corresponding to a 5 meter step size (circles) and one for a 10 meter step size (diamonds).

The general shape of the probability data for ‘both species’ and ‘both species before scare’ at each step size is very similar, linear in the middle with a slope of approximately zero and positive and negative tails. The tails at the 10 meter step size are more pronounced than for the 5 meter step size plots. Conversely, the shape of the probability data for ‘both species after scare’ is sinodial at the 5 and 10 meter step size. Again the shape of the curve is more pronounced for the 10 meter step size plot. When comparing between step sizes you will see a general decrease in the total number of difference event occurrences from the 5 to 10 meter step size. There is also a noticeable difference in the distribution of difference event occurrences, with less occurrence events at a difference event of zero for ‘both species after scare’ 5 and 10 meter step size plots. The range in difference events for all plots is between -pi and pi radians however the total number of difference event occurrences for ‘both species’ are higher than ‘both species before scare’ and ‘both species after scare’.

ii) Plains

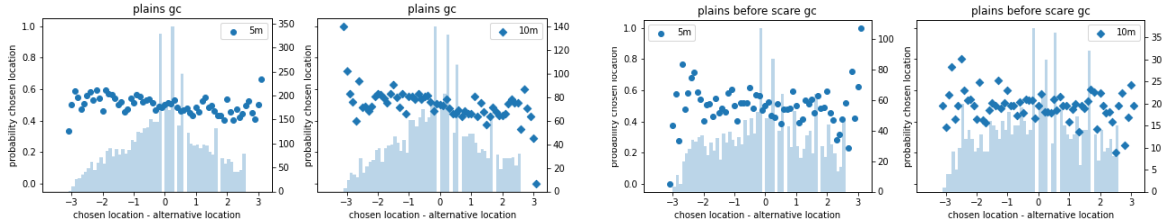


Figure 10. The scatter plot represents the probability that an individual will select a chosen or true location given the choice between a chosen location and one randomly selected alternative (not observed in the field) location. This choice was made by looking at the difference between two differences. The first difference is between the vector pointing from the animals current step to its next chosen location and the vector pointing from the current step and the center of the group. This value is termed chosen location. The second difference is between the vector pointing from the animals current step to an alternative location and the vector pointing from the current step to the group center. This value is term alternative location. The difference is then taken between these two difference values. These values are shown on the x-axis of each plot. The underlying histogram shows the number of counts for each difference event, thus there is a bin for every scatter point. The left y-axis on each plot is the probability of choosing the chosen location, the right y-axis corresponds to the number of counts for each difference event. For each resolution (plains and plains before scare) there are 2 figures, one which displays the results corresponding to a 5 meter step size (circles) and one for a 10 meter step size (diamonds).

The overall shape of the probability data for all plots is linear with a slope of zero. For ‘plains before scare’ the data is not as tightly compact as it is in ‘plains’. The total number of difference event occurrences across step sizes decreases from the 5 meter to the 10 meter step size. They also decrease between ‘plains’ and ‘plains before scare’. The distributions of difference event occurrences look similar across step sizes. However there is a slightly wider distribution in ‘plains before scare’ plots. There are is no ‘plains after scare’ due to a lack of data

iii) Grevy's

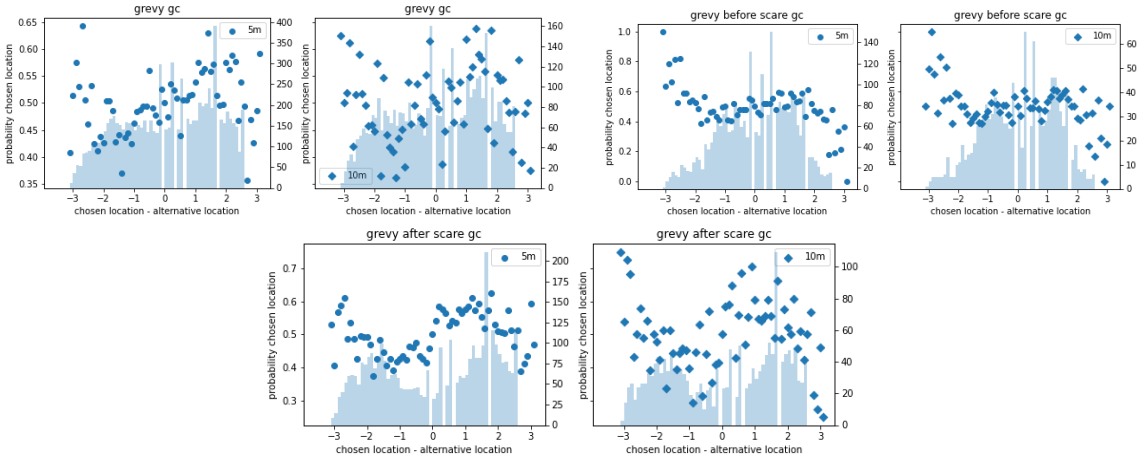


Figure 11. The scatter plot represents the probability that an individual will select a chosen or true location given the choice between a chosen location and one randomly selected alternative (not observed in the field) location. This choice was made by looking at the difference between two differences. The first difference is between the vector pointing from the animals current step to its next chosen location and the vector pointing from the current step and the center of the group. This value is termed chosen location. The second difference is between the vector pointing from the animals current step to an alternative location and the vector pointing from the current step to the group center. This value is term alternative location. The difference is then taken between these two difference values. These values are shown on the x-axis of each plot. The underlying histogram shows the number of counts for each difference event, thus there is a bin for every scatter point. The left y-axis on each plot is the probability of choosing the chosen location, the right y-axis corresponds to the number of counts for each difference event. For each resolution (Grevy's, Grevy's before scare, Grevy's after scare) there are 2 figures, one which displays the results corresponding to a 5 meter step size (circles) and one for a 10 meter step size (diamonds).

If comparing across 'grevy', 'grevy before scare', and 'grevy after scare' you will see differently shaped plots. For 'grevy' 5 and 10 meter step size plots there is no noticeable trend in the data, points are randomly scattered. The plots in 'grevy before scare' are linear in the middle with a slope of around zero and have positive and negative tails. Finally, the plots in 'grevy after scare' have sinodial like curves across step sizes. The amount of difference event occurrences are similar in 'grevy before scare' and 'grevy after scare', however there are more difference event occurrences in 'grevy'. When comparing total numbers of difference events across step sizes, you see a decrease in the amount of difference event occurrences from the 5 to the 10 meter step size. The range of difference events holds consistent throughout plots.

D) Game trails

i) Grevy's

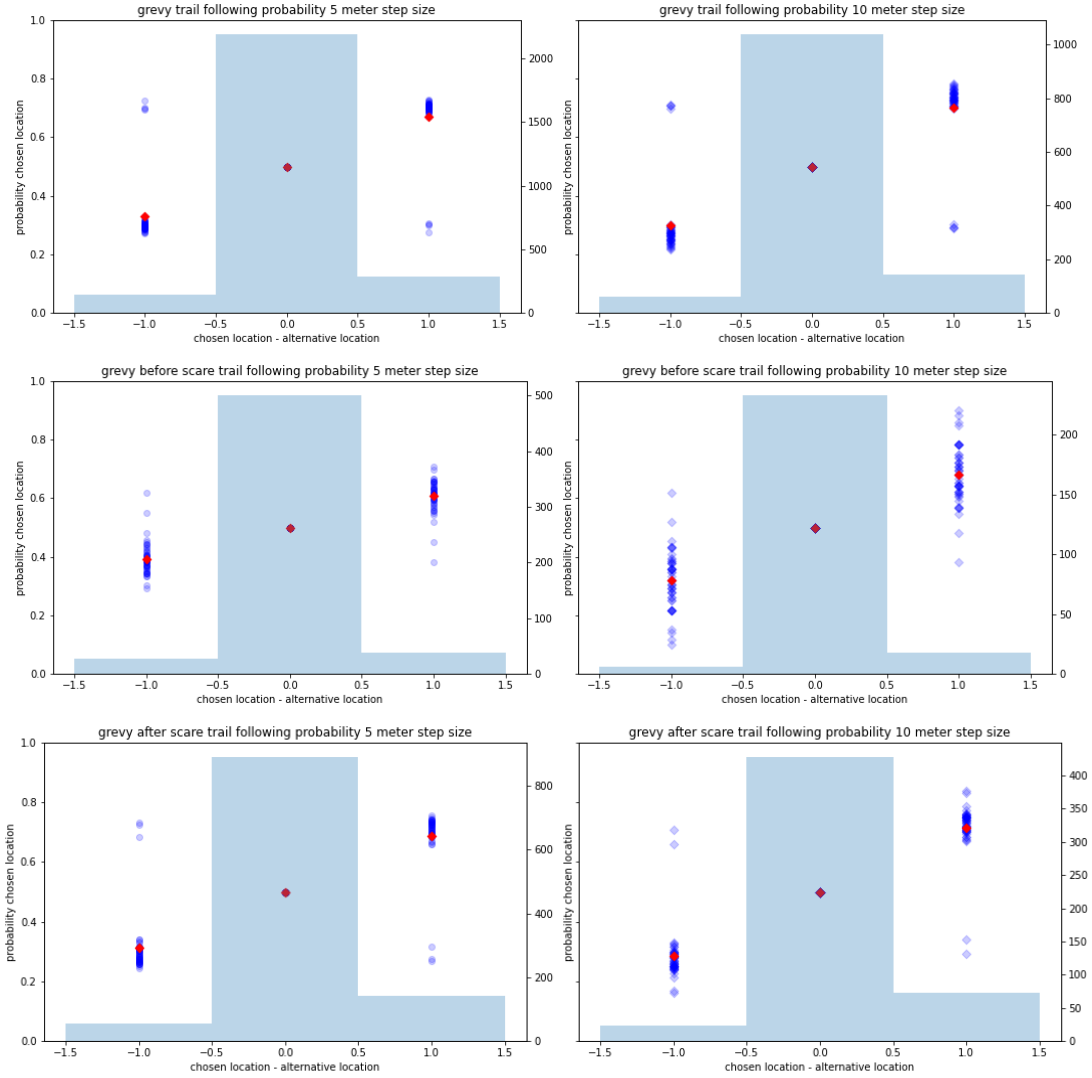


Figure 12. The scatter plot represents the probability that an individual will select a chosen or true location given the choice between a chosen location and one randomly selected alternative (not observed in the field) location. This choice was made by looking at difference between whether the number of times a chosen location was on a trail and the number of times an alternative location was on a trail. These values are shown on the x-axis of each plot. The underlying histogram shows the number of counts for each difference event, thus there is a bin for every scatter point. The left y-axis on each plot is the probability of choosing the chosen location, the right y-axis corresponds to the number of counts for each difference event. For each resolution (Grevy's, Grevy's before scare, Grevy's after scare) there are 2 figures, one which displays the results corresponding to a 5 meter step size (circles) and one for a 10 meter step size (diamonds). To measure consistency probability was calculated 50 times which is represented by the multitude of blue points for each plot. The red points are the average probability of the the 50 runs. Only the Grevy's are measured because there was no data for the plains

If looking at the average of the 50 runs (red points) you can see the same linear trend in each one of the plots. If comparing across step sizes within each event (no scare, before scare, and after scare) you see that the probabilities tend to stay the same. When comparing between 'grevy before scare' and 'grevy after scare' you will notice a that the blue points at both the 5 and 10 meter step size are more closer to each other in 'grevy after scare'. You will also notice an increase in the difference event occurrences at the difference event of one for 'grevy after scare'.

VI. DISCUSSION

This study examines how both habitat and social factors influence movement decisions in two sympatric species of zebra, Grevy's and plains. With data collected from unmanned aerial vehicle imagery of the zebras moving around their wild environment at Mpala Research Centre in Kenya, I utilize the step selection function to test how zebras base their movement on the following features: 1) social density (number of conspecifics around them), 2) group center (desire to be near the center of the group), 3) recently used space (areas where other group members have moved across recently, likely indicative of "following" behavior), and 4) game trails (paths that animals use to commute in the habitat). I further assess how moments of perceived risk influence movement behavior by examining movement decisions before and after scare events where the zebra groups are startled. Below I discuss the main findings for each of the four ecological features analyzed, considering the results from both the step selection model and the probability that an individual will select the chosen (or "true") location given a choice between the chosen location and a randomly selected alternate location.

Social Density

Model
When positional data is sampled less frequently (10 meter step size) step selection outputs show that both species tend to want to be close to each other and seek out individuals within 1 to 10 meters from themselves. Once individuals get farther than 15 meters away from a focal individual however neither species is motivated to be close to them. This holds true for before and after scare events at the 'both species' resolution. After scare event model outputs show that both species tend to want to be even closer to individuals that were very close to them already. This is evident in the jump in beta value at the 1 meter radius size around the individual. Grevy's on the other hand, are less concerned about being closer to very nearby individuals and rather in the general vicinity of a group of individuals before a scare event. This is evident for the 1 and 5 meter radius sizes. However, this changes as soon as the Grevy enters an alarmed state. Now they p refer to be close to nearby individuals. Plains at this sampling rate desire to be close to individuals within a range of 1 to 10 meters. This holds true before scare events. At a higher sampling frequency (5 meter step size) Grevy's seem less concerned about being close with other animals. On the contrary, plains are interested

in being close with other individuals that are within 1 to 5 meters, especially before scare events.

Probability
At the both species resolution the probability plots generally corroborate the outputs generated from the step selection function. Again, when using a lower sampling rate, there is a clear trend for an individual to choose a location where there is a higher fraction of other individuals. Especially when the individuals in this location are not packed together but loosely aggregated. This is evident from the 5 and 10 meter radius size plots across all resolutions. One thing to observe when comparing the model and probability output however, is that although the model suggests that zebras want to be near other individuals that are very close by (1 meter radius size) the corresponding probability plots show that there are very few events supporting this finding. Instead, majority of the data supports the notion that zebras are indifferent when it comes to associating with zebras that are extremely close to them. This artifact is again evident in the Grevy's probability dataset, where at 1 meter radius sizes there is more data supporting the notion that Grevy's are uninterested in being close to other conspecifics. However, model outputs for Grevy's after a scare event are validated by the the probability outputs. Both the model and the probability analysis suggest that after a scare event Grevy's tend to want to be loosely near other individuals (5 and 10 meters away). As for the plains zebras, the probability output shows for both sampling rates (5 and 10 meter step sizes) that plains want to be near loosely grouped individuals. This is concurrent with the step selection output. Again there is insufficient data supporting the model finding that zebras want to be very closely aggregated together. At a higher sampling frequency (5 meter step size) again the probability data supports the model output. Showing that both Grevy's and plains do not want to be right up next to each other but loosely aggregated as is evident from the 5 meter radius size plots

Recently used space

Model
At both sampling rates (5 and 10 meter step size), zebras tend to follow individuals that are up to 10 meters ahead of them. Beyond 10 meters, zebras do not move towards recently used locations of other individuals. This trend broadly holds true across the genus level. However, Grevy's zebras after a scare event tend to care less about where other individuals have recently been. In other words, Grevy's zebras are less likely to follow where their conspecifics were moving after a scare event (which could be indicative of potential risk). At higher sampling rates (5 meter step size), zebras overall are less likely to follow other individuals when compared to lower sampling rates (10 meter step size).

Probability
At both the 5 and 10 meter step sizes, the probability analysis matches the step selection model output. There are sufficient values for each probability point to support the step selection model results.

Group center

Model
The model output for each sampling rate (5 and 10 meter step sizes) does not differ substantially across analyses. For both zebra species, the analyses indicate that they don't tend to move towards the center of their groups. Before a scare event however, the zebra species appear to actively move away from the center of the group. On the other hand, after a scare event, the zebras change their movement preferences to actively move towards the group center. This seems to suggest that zebras prefer to move towards the center of the group after perceiving a potential risk (scare event). On the species level, Grevy's in general move towards the group center, while plains zebras move away from the group center in general and before a scare event. Before scare events, Grevy's somewhat move away from the center of the group, which contrasts with their behavior after a scare event where they are more likely to move toward the group center. These results are interesting because they are opposite to the predictions, where plains zebras were expected to move towards the group center and Grevy's zebras were expected to move away from the group center due to their difference in social dynamics.

Probability
In general, the probability data supports the output from the model. At each sampling rate both species are largely apathetic about moving towards the center of the group unless there is a scare event, then zebras slightly tend towards their group's center. For the plains zebras, the probability plot suggests that plains are not concerned with moving towards or away from the group center. This contrasts with the model results. For the Grevy's zebras, the probabilities suggest that before a scare event Grevy's are uninterested in moving towards the group center. After a scare event, Grevy's tend to want to head towards the center of the group ever so slightly.

Game trails

Probability
As there was not sufficient data for the plains zebras for this feature, the results for Grevy's suggest that this species overall tends to chose locations where there are game trails. At both sampling rates (5 and 10 step sizes), the data becomes more ordered after a scare event, indicating that Grevy's use game trails more after a scare event.

VII. CONCLUSION

By exploiting recent advances in tracking technology with unmanned aerial vehicles, remote sensing of the environment, and analytical methods, this study reveals ecological and social variables that influence the local movement decisions made by individual zebras living in wild herds. Taken together, this study exemplifies the need to consider both habitat and social variables to gain a greater understanding of how animals choose where to go when living in complex environments.

The results indicate that when sampling at a lower rate (10 meter step size), both zebra species tend to loosely group with other individuals in their vicinity. Interestingly, Grevy's zebra were also particularly likely to aggregate with others after a scare event, contrary to my predictions that Grevy's zebras would not base their movements on social factors due to the absence of strong social connectedness in this species. These results clearly indicate that social grouping behavior for Grevy's zebra remains important during movement decisions and especially during movement decisions after perceived risk. At the genus level, zebras preferred to travel in areas that were recently used by other herd members up to a distance of 10 meters. This pattern held true for both species overall and before and after a scare event with one notable exception: Grevy's appear to not use areas recently used by others after scare events. Results from the influence of moving towards the group center indicate no strong preference for either species, although Grevy's zebra appear to slightly move towards the group center. These results are interesting in light of the social behavior of Grevy's zebra, as I predicted they would not consider social variables as strongly when making movement decisions. Because this effect is very small, further analysis may be required to adequately determine how the group center impacts zebra movement behavior. The results also show that Grevy's zebra were likely to choose locations in the habitat where there were game trails, especially after a scare event. The use of game trails imply that Grevy's zebra prioritize traveling along paths in the habitat. These paths may be provide greater connectivity between resources and be easier to move along with less vegetation, facilitating preferred use. By using these game trails even more after a perceived threat (scare event), Grevy's zebras may also be selecting straightforward escape routes to move away from threats. Incorporating game trail use together with social variables (like recently used space and social density) may provide a promising direction of future research.

Ultimately, it appears that both plains and Grevy's zebra base their movement behavior on social and ecological variables concurrently. By examining how habitat and social variables shape individual decisions about where to move, we can eventually predict emergent group structure during collective animal movements. As the majority of animals aggregate in groups at some point in their lives, studies that simultaneously consider both the social and environmental influences driving collective movement will advance our understanding of how individuals coordinate their actions and respond to the stimuli around them.

VII. REFERENCES

Alarcón-Nieto, G., Graving, J. M., Klarevas-Irby, J. A., Maldonado-Chaparro, A. A., Mueller, I., & Farine, D. R. (2018). An automated barcode tracking system for behavioural studies in birds. *Methods in Ecology and Evolution*, *9*(6), 1536-1547.

Berdahl A, Torney CJ, Ioannou CC, Faria JJ, Couzin ID. Emergent sensing of complex environments by mobile animal groups. Science. 2013;339(6119):574-576. doi:10.1126/science.1225883

D Biro, DJT Sumpter, J Meade, T Guilford. 'From compromise to leadership in pigeon homing' *Current biology* (2006)

Boenisch, F., Rosemann, B., Wild, B., Dormagen, D., Wario, F., & Landgraf, T. (2018). Tracking all members of a honey bee colony over their lifetime using learned models of correspondence. *Frontiers in Robotics and AI*, *5*, 35.

Boyce MS, McDonald LL. Relating populations to habitats using resource selection functions. Trends Ecol Evol. 1999;14(7):268-272. doi:10.1016/s0169-5347(99)01593-1

Buhl J, Sumpter DJ, Couzin ID, et al. From disorder to order in marching locusts. Science. 2006;312(5778):1402-1406. doi:10.1126/science.1125142

Coulon, A., Morellet, N., Goulard, M. et al. Inferring the effects of landscape structure on roe deer (Capreolus capreolus) movements using a step selection function. Landscape Ecol 23, 603–614 (2008). *https://doi.org/10.1007/s10980-008-9220-0*

Couzin ID, Krause J, Franks NR, Levin SA. Effective leadership and decision-making in animal groups on the move. Nature. 2005;433(7025):513-516. doi:10.1038/nature03236

Crall, J. D., Gravish, N., Mountcastle, A. M., & Combes, S. A. (2015). BEEtag: a low-cost, image-based tracking system for the study of animal behavior and locomotion. *PloS one*, *10*(9), e0136487.

Dell, A. I., Bender, J. A., Branson, K., Couzin, I. D., de Polavieja, G. G., Noldus, L. P., ... & Brose, U. (2014). Automated image-based tracking and its application in ecology. *Trends in ecology & evolution*, *29*(7), 417-428.

Fortin, D., Beyer, H.L., Boyce, M.S., Smith, D.W., Duchesne, T. and Mao, J.S. (2005), Wolves influence elk movements: behavior shapes a trophic cascade in Yellowstone national park. Ecology, 86: 1320-1330. doi:10.1890/04-0953

Graving, J. M., Chae, D., Naik, H., Li, L., Koger, B., Costelloe, B. R., & Couzin, I. D. (2019). DeepPoseKit, a software toolkit for fast and robust animal pose estimation using deep learning. *Elife*, *8*, e47994.

Hodson, J., Fortin D., Bélanger, L., Fine-scale disturbances shape space-use patterns of a boreal forest herbivore, Journal of Mammalogy, Volume 91, Issue 3, 16 June 2010, Pages 607–619, *https://doi.org/10.1644/09-MAMM-A-289.1*

Katz Y, Tunstrøm K, Ioannou CC, Huepe C, Couzin ID. Inferring the structure and dynamics of interactions in schooling fish. Proc Natl Acad Sci U S A. 2011;108(46):18720-18725. doi:10.1073/pnas.1107583108

King AJ, Douglas CM, Huchard E, Isaac NJ, Cowlishaw G. Dominance and affiliation mediate despotism in a social primate. Curr Biol. 2008;18(23):1833-1838. doi:10.1016/j.cub.2008.10.048

Latombe G, Fortin D, Parrott L. Spatio-temporal dynamics in the response of woodland caribou and moose to the passage of grey wolf. J Anim Ecol. 2014;83(1):185-198. doi:10.1111/1365-2656.12108

Leblond, M., Dussault, C., & Ouellet, J. P. (2010). What drives fine-scale movements of large herbivores? A case study using moose. *Ecography*, *33*(6), 1102-1112.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436-444.

Mann, R. P., Armstrong, C., Meade, J., Freeman, R., Biro, D., & Guilford, T. (2014). Landscape complexity influences route-memory formation in navigating pigeons. *Biology letters*, *10*(1), 20130885.

Mathis, A., Mamidanna, P., Cury, K. M., Abe, T., Murthy, V. N., Mathis, M. W., & Bethge, M. (2018). DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. *Nature neuroscience*, *21*(9), 1281-1289.

Nagy, M., Couzin, I. D., Fiedler, W., Wikelski, M., & Flack, A. (2018). Synchronization, coordination and collective sensing during thermalling flight of freely migrating white storks. Philosophical Transactions of the Royal Society B: Biological Sciences, 373(1746), 20170011.

Northrup, J. M., Pitt, J., Muhly, T. B., Stenhouse, G. B., Musiani, M., & Boyce, M. S. (2012). Vehicle traffic shapes grizzly bear behaviour on a multiple-use landscape. *Journal of Applied Ecology*, *49*(5), 1159-1167.

Pereira, T. D., Aldarondo, D. E., Willmore, L., Kislin, M., Wang, S. S., Murthy, M., & Shaevitz, J. W. (2019). Fast animal pose estimation using deep neural networks. Nature methods, 16(1), 117–125. https://doi.org/10.1038/s41592-018-0234-5

Pérez-Escudero, A., Vicente-Page, J., Hinz, R. C., Arganda, S., & de Polavieja, G. G. (2014). idTracker: tracking individuals in a group by automatic identification of unmarked animals. Nature methods, 11(7), 743–748. https://doi.org/10.1038/nmeth.2994

Potts, J. R., Mokross, K., & Lewis, M. A. (2014). A unifying framework for quantifying the nature of animal interactions. *Journal of The Royal Society Interface*, *11*(96), 20140333.

Romero-Ferrero, F., Bergomi, M. G., Hinz, R. C., Heras, F., & de Polavieja, G. G. (2019). idtracker.ai: tracking all individuals in small or large collectives of unmarked animals. Nature methods, 16(2), 179–182. https://doi.org/10.1038/s41592-018-0295-5

Rosenthal, S. B., Twomey, C. R., Hartnett, A. T., Wu, H. S., & Couzin, I. D. (2015). Revealing the hidden networks of interaction in mobile animal groups allows prediction of complex behavioral contagion. *Proceedings of the National Academy of Sciences*, *112*(15), 4690-4695.

Strandburg-Peshkin, A., Farine, D. R., Crofoot, M. C., & Couzin, I. D. (2017). Habitat and social factors shape individual decisions and emergent group structure during baboon collective movement. *Elife*, *6*, e19505.

Sumpter, D J T. “The principles of collective animal behaviour.” *Philosophical transactions of the Royal Society of London. Series B, Biological sciences* vol. 361,1465 (2006): 5-22. doi:10.1098/rstb.2005.1733

Thurfjell, H., Ciuti, S., & Boyce, M. S. (2014). Applications of step-selection functions in ecology and conservation. *Movement ecology*, *2*(1), 4.

Vicsek, T., & Zafeiris, A. (2012). Collective motion. Physics reports, 517(3-4), 71-140.

APPENDIX

Convolutional Neural Networks

In an abstract sense, ConvNets recognize and classify objects within an image using some combination of four basic operations - convolution, non linearity, pooling, and classification. In convolution, features are extracted from an image by taking the dot product between a filter - a 3D matrix of weight values with a specified size - and each part of an image - a 3D matrix of pixel values - to produce another 3D matrix of a different shape which is often referred to as a feature map. The primary purpose of convolution is to find features in the image, put them into a feature map, and still preserve the spatial relationship between pixels. Changing a filter's weights produces different feature maps for the same input image. In practice, ConvNets *learn* the values for filters that best help them recognize a specific object in an image by optimizing a cost function that compares that output of the training model to known ground truth values. To extract complex patterns in the images, a convolutional neural network needs to introduce an element of non linearity in the form of a non-linear equation or activation function. The activation function gives the ConvNet the capability to learn non-linear relationships. A neural network without an activation function is essentially just a linear regression model.

There are a few activation functions researchers commonly use - the sigmoid function, the tanh function, and the ReLU function - but the ReLU function is typically the most used. For an image, the values in the feature map are fed through the activation function. The resulting feature map is the same size, spatial representation, and if using ReLU, all negative values from the map are set to zero. Pixel values that give the best representation of the feature are then extracted from the feature map and placed into a smaller matrix. This process is called pooling. Pooling reduces the dimensionality of each feature map, making it more manageable computationally. Most importantly however, it allows the network to recognize images containing variations of the same object. Once a condensed version of the feature map has been created, the image is ready to be classified. To do this, values from the condensed feature map are put through a fully connected layer.

A fully connected layer is a traditional artificial neural network that uses a softmax activation function in the output layer. The term "fully connected" implies that every neuron in the previous layer is connected to every neuron on the next layer. The purpose of the fully connected layer is to create non-linear combinations of the condensed feature maps to classify the input image. Often this is better than using the non-combined features extracted from from convolution and pooling layers. The output of the CNN is a list of n probabilities where n is the number of classes (categories) an image can be classified into. The sum of these probabilities is always one. To check how close the output probability matches the true label (an integer assigned to an image by the programmer) of the image a cost function is used. If the cost is high (meaning the predicted value is very different from the true value), the network updates the values in the filter to extract better features for classifying the image. Mathematically this is

achieved by calculating the gradient of the cost function. Filters are continually updated based on the calculated gradient until the difference between the predicted value and the true value of the image is as small as it can be. This process is repeated using a large amount of images - training data - until the statistical relationships between a set of input data (i.e. images) and the desired output (i.e. classification). After adequate training, a model can be used to make predictions on previously-unseen data from the same dataset - inputs that were not part of the training set, which is known as inference. In other words, these models are able to generalize sometimes human-level expertise at scale after having been trained on only a relatively small number of examples.

Although the task of implementing deep-learning models in software and training them on new data is complex and requires expert knowledge, once the underlying model and training routine are implemented, a high accuracy model for a novel context can be built with minimal modification - often just by changing the training data. Today there a number of simplified toolkits with easy-to-use software interfaces that allow scientists to apply the power of computer vision to their research (Mathis et al. (2018); Pereira et al. (2019); Graving et al., 2019).

get-tracks_Taylor

July 16, 2020

[980]:
```
import numpy as np
from matplotlib import pyplot as plt
import os
import glob
import pandas as pd
import pickle

# import sys
# To see koger_general_functions
# sys.path.append('/home/golden/coding/drone-tracking/code/objects' )

from Observation import Observation

# sys.path.append('/Users/Taylor/Documents/koger_general_functions.py')

import mapping_functions as kmap
```

[ ]:
```

```

[ ]:
```

```

[987]:
```
local_data = '/Users/Taylor/Documents/herdhover-data'

observations_meta_folder = (
    os.path.join(local_data, 'observations'))
observations_meta_file = os.path.join(observations_meta_folder,␣
↪'observations-meta.csv')
observations_meta = pd.read_csv(observations_meta_file)

tracks_folder = '/Users/Taylor/Documents/tracks/tracks/utm'
tracks_folder = '/Users/Taylor/Documents/herdhover-data/tracks/utm'
```

[988]:
```
observation_species = ['Grevy\'s', 'plains'] # Observations to use

obs_to_use_mask = (observations_meta['species'].isin(observation_species) &
                   (observations_meta['tracked']=='True'))
```

```
observations_meta_used = observations_meta.loc[obs_to_use_mask].copy()
observations_meta_used.sort_values('observation', inplace=True)
observations_meta_used.reset_index()

observations_info = observations_meta_used.loc[:,['observation', 'species']].
 ↪values

print('Using {} observations'.format(len(observations_info)))
```

```
Using 21 observations
```

[989]:
```
observations_folder = '/Users/Taylor/Documents/observations'

orientation_files = glob.glob(os.path.join(observations_folder, '*/
 ↪*-body-points.npy'))
orientation_files.sort()
observation_names = [os.path.basename(file).split('-')[0] for file in␣
 ↪orientation_files]
orientations_list = [np.load(file) for file in orientation_files]
```

[990]:
```
# Species to use within observation
target_species = ['Grevy\'s', 'plains']
save_folder = None
observation_dicts = []
for observation_info in observations_info:
    if observation_info[0] in observation_names:
        body_points_file = os.path.join(observations_folder, '{}/{}-body-points.
 ↪npy'.format(
        observation_info[0], observation_info[0]))

        postures_file = os.path.join(observations_folder, '{}/{}-postures.npy'.
 ↪format(
        observation_info[0], observation_info[0]))
        posture_scores_file = os.path.join(observations_folder, '{}/
 ↪{}-posture-scores.npy'.format(
        observation_info[0], observation_info[0]))

        body_points = np.load(body_points_file)
        postures = np.load(postures_file)
        posture_scores = np.load(posture_scores_file)

        observation_dict = {'name': observation_info[0],
                            'focal_species': observation_info[1],
                           'tracks_folder': tracks_folder,
                            'body_points': body_points,
                            'postures': postures,
                            'posture_scores': posture_scores,
```

```
                            'obs_meta_folder': observations_meta_folder,
                            'target_species': target_species,
                            'save_folder': save_folder}
        observation_dicts.append(observation_dict)
```

[991]:
```
len(observation_dicts)
```

[991]: 15

[992]:
```
observations = []
observations_names = []
for observation_info in observation_dicts[:]:

    observation_name = observation_info['name']
    focal_species = observation_info['focal_species']
    tracks_folder = observation_info['tracks_folder']
    body_points = observation_info['body_points']
    postures = observation_info['postures']
    posture_scores = observation_info['posture_scores']
    observations_meta_folder = observation_info['obs_meta_folder']
    target_species = observation_info['target_species']
    save_folder = observation_info['save_folder']
    print(observation_name)
    tracks = np.load(os.path.join(tracks_folder, '{}-utm.npy'.
 ↪format(observation_name)))
    tracks_meta_file = os.path.join(observations_meta_folder,
                                    observation_name,
                                    '{}-tracks-meta.csv'.
 ↪format(observation_name))
    tracks_meta = pd.read_csv(tracks_meta_file)

    obs_meta_file = os.path.join(observations_meta_folder,
                                    observation_name,
                                    '{}-meta.csv'.format(observation_name))
    obs_meta = pd.read_csv(obs_meta_file)

    observation = Observation(observation_name, tracks, body_points, postures,␣
 ↪posture_scores,
                              tracks_meta, obs_meta, target_species,␣
 ↪focal_species)
    # Use only tracks I want
    observation.combine_and_delete_tracks()
    observation.get_posture_based_positions()
    observations.append(observation)
    observations_names.append(observation_name)
```

```
observation015
```

```
observation027
observation034
observation036
observation044
observation053
observation066
observation070
observation074
observation082
observation083
observation086
observation088
observation102
observation108
```

[ ]:

[993]:
```
positions = []
positions_names = []
before_scare_positions = []
before_scare_positions_names = []
after_scare_positions = []
after_scare_positions_names = []

look_back = 3600

for obs in range(len(observations)):
    positions.append(observations[obs].positions)
    positions_names.append(observations_names[obs])

    if not isinstance(observations[obs].scare_frame, str):
        if observations[obs].positions[:, :observations[obs].scare_frame, :].
↪shape[1] > look_back:
            before_scare = observations[obs].positions[:, :observations[obs].
↪scare_frame, :]
            before_scare_positions.append(before_scare)
            before_scare_positions_names.append(observations_names[obs])

        if observations[obs].positions[:, observations[obs].scare_frame:, :].
↪shape[1] > look_back:
            after_scare = observations[obs].positions[:, observations[obs].
↪scare_frame:, :]
            after_scare_positions.append(after_scare)
            after_scare_positions_names.append(observations_names[obs])
```

[994]:
```
observations[0].name
```

[994]: 'observation015'

[995]:
```
grevy_positions = []
grevy_positions_names = []
grevy_before_scare_positions = []
grevy_before_scare_positions_names = []
grevy_after_scare_positions = []
grevy_after_scare_positions_names = []

plains_positions = []
plains_positions_names = []
plains_before_scare_positions = []
plains_before_scare_positions_names = []
plains_after_scare_positions = []
plains_after_scare_positions_names = []

look_back = 3600

for obs in observations:
    if np.any(obs.tracks_meta['species'].values == 'Grevy\'s'):
        if not np.any(obs.tracks_meta['species'].values == 'plains'):
            grevy_positions.append(obs.positions)
            grevy_positions_names.append(obs.name)

            if not isinstance(obs.scare_frame, str):
                if obs.positions[:, :obs.scare_frame, :].shape[1] > look_back:
                    before_scare = obs.positions[:, :obs.scare_frame, :]
                    grevy_before_scare_positions.append(before_scare)
                    grevy_before_scare_positions_names.append(obs.name)

                if obs.positions[:, obs.scare_frame:, :].shape[1] > look_back:
                    after_scare = obs.positions[:, obs.scare_frame:, :]
                    grevy_after_scare_positions.append(after_scare)
                    grevy_after_scare_positions_names.append(obs.name)

    if np.any(obs.tracks_meta['species'].values == 'plains'):
        if not np.any(obs.tracks_meta['species'].values == 'Grevy\'s'):
            plains_positions.append(obs.positions)
            plains_positions_names.append(obs.name)

            if not isinstance(obs.scare_frame, str):
                if obs.positions[:, :obs.scare_frame, :].shape[1] > look_back:
                    before_scare = obs.positions[:, :obs.scare_frame, :]
                    plains_before_scare_positions.append(before_scare)
                    plains_before_scare_positions_names.append(obs.name)

                if obs.positions[:, obs.scare_frame:, :].shape[1] > look_back:
```

```
                    after_scare = obs.positions[:, obs.scare_frame:, :]
                    plains_after_scare_positions.append(after_scare)
                    plains_after_scare_positions_names.append(obs.name)
```

[1064]:
```
data_names = [positions_names, before_scare_positions_names,␣
↪after_scare_positions_names,
              grevy_positions_names, grevy_before_scare_positions_names,␣
↪grevy_after_scare_positions_names,
              plains_positions_names, plains_before_scare_positions_names,␣
↪plains_after_scare_positions_names]
data_list = [positions, before_scare_positions, after_scare_positions,
              grevy_positions, grevy_before_scare_positions,␣
↪grevy_after_scare_positions,
              plains_positions, plains_before_scare_positions,␣
↪plains_after_scare_positions]

filenames = ['both_species', 'both_species_before_scare',␣
↪'both_species_after_scare',
             'grevy', 'grevy_before_scare', 'grevy_after_scare',
             'plains', 'plains_before_scare', 'plains_after_scare',
             ]
```

[1069]:
```
# positions_names
```

[1066]:
```
map_obs = ['observation083', 'observation108', 'observation086',␣
↪'observation074']

map_data_names = []
map_data_list = []
map_types = []

for obs_names, all_obs_data in zip(data_names, data_list):
    name_index_data = []
    new_data = []
    for obs_name, obs_data in zip(obs_names, all_obs_data):
        if obs_name in map_obs:
            name_index_data.append(obs_name)
            new_data.append(obs_data)
    map_data_names.append(name_index_data)
    map_data_list.append(new_data)
```

[1068]:
```
map_data_names
```

[1068]:
```
[['observation074', 'observation083', 'observation086', 'observation108'],
 ['observation074', 'observation083'],
 ['observation074', 'observation083'],
```

```
 ['observation074', 'observation083', 'observation086', 'observation108'],
 ['observation074', 'observation083'],
 ['observation074', 'observation083'],
 [],
 [],
 []]
```

[998]:
```
observation_name_to_big_map_name_file = '/Users/Taylor/Documents/
 ↪observation_name_to_big_map_name_dict.pkl'
with open(observation_name_to_big_map_name_file, 'rb') as f:
        observation_name_to_big_map_name_dict = pickle.load(f)

herdhover_folder = '/Users/Taylor/Documents/herdhover-data'

observation_maps_dict = {}

for observation_name in map_obs:

    map_name = observation_name_to_big_map_name_dict[observation_name]
    mask_file = os.path.join(herdhover_folder, 'game-trails',
                             '{}-trails_mask.npy'.format(map_name))
    mask_info_file = os.path.join(herdhover_folder, 'game-trails',
                             '{}-mask_info.csv'.format(map_name))

    trail_mask = np.load(mask_file)
    mask_info_df = pd.read_csv(mask_info_file)

    observation_maps_dict[observation_name] = {'trail_mask': trail_mask,
                                               'mask_info_df': mask_info_df}
```

[1025]:
```
vals = []
for name, utms in zip(map_data_names[0], map_data_list[0]):
    trail_mask = observation_maps_dict[name]['trail_mask']
    mask_info_df = observation_maps_dict[name]['mask_info_df']
    utms = utms
    new_coords_l, spatial, time_index, _ = spatial_dis(utms, 5.0, 1)

    print(np.any(np.isnan(np.concatenate(spatial))))

    print([np.any(np.isnan(t)) for t in spatial])
    print([t.shape for t in spatial])

#     for s in spatial:
#         vals.append(get_trail_values_from_utm(trail_mask, mask_info_df, s))
```

```
True
[False, False, False, False, False, False, False, False, False, False, False,
```

```
True]
[(89, 2), (93, 2), (90, 2), (84, 2), (93, 2), (90, 2), (88, 2), (102, 2), (83,
2), (83, 2), (86, 2), (1, 2)]
True
[False, False, False, False, False, False, False, False, True, True, True]
[(41, 2), (23, 2), (44, 2), (105, 2), (59, 2), (130, 2), (72, 2), (46, 2), (1,
2), (1, 2), (1, 2)]
True
[False, False, False, False, False, False, False, False, False, False, True,
True, True, True, True, True, True, True, True, True]
[(34, 2), (52, 2), (54, 2), (61, 2), (61, 2), (58, 2), (33, 2), (59, 2), (44,
2), (31, 2), (1, 2), (1, 2), (1, 2), (1, 2), (1, 2), (1, 2), (1, 2), (1, 2), (1,
2), (1, 2)]
False
[False, False, False, False, False]
[(129, 2), (124, 2), (120, 2), (121, 2), (133, 2)]
```

[1000]:
```
map_data_names[0]
```

[1000]:
```
['observation074', 'observation083', 'observation086', 'observation108']
```

[952]:
```
observation_maps_dict['observation108']['mask_info_df']
```

[952]:

	x_origin	y_origin	pixel_width	pixel_height
0	267613.19317	42680.57681	0.23576	-0.23576

[845]:
```
def random_steps(pts_act, r=None, number_of_steps=5):
    """
    Generates possible x,y positions for every individual at every step by␣
↪pulling random values from
    calculated step sizes and turning angles

    Inputs:
        pts_act: either list of 2D numpy arrays containing the xy coordinates␣
↪for each individual after they have moved at least r meters(spatial␣
↪discretization),
                 or a 3d numpy array with raw xy positions of animals in space␣
↪with shape (individual, frame, 2) (temporal discretization)

        r: int; distance focal individual should travel before a step is␣
↪recorded
    Ouputs:
        coords_lst: list of 3D numpy arrays with new xy positions for each␣
↪individual in pts_act with shape (step number, location, 2). coords_lst[0] =␣
↪xy cooridnates
                    for alternative steps 1-number_of_steps for individual 0
```

```
        coords_list: list of 3D numpy arrays with new xy positions for each␣
↪individual in pts_act. coords_list[0] = xy coordinates
                    for 1st alternative step for each individual

    """

    if isinstance(pts_act, np.ndarray):
        act_diff = np.diff(pts_act, axis=1)
        step_size = np.sqrt((act_diff[:,:,0])**2 + (act_diff[:,:,1])**2)

        act_dir = np.arctan2(act_diff[:,:,1], act_diff[:,:,0])
        d_diff = np.diff(act_dir)

        coords_list = []
        for number in range(number_of_steps):

            coords = np.zeros(pts_act.shape)
            for ind in range(pts_act.shape[0]):

                ss_dropnan = step_size[ind][~np.isnan(step_size[ind])]
                d_diff_dropnan = d_diff[ind][~np.isnan(d_diff[ind])]

                #Once the NaNs are dropped some ind have empty ss_dropnan and␣
↪d_diff_dropnan. Inds that have this issue have been
                #tagged in time_adj and will be dropped from dataset later, but␣
↪in order to preserve labeling they need to stay until the final dataframe is␣
↪made.
                #Thus any ind with this issue will use lst
                if len(d_diff_dropnan)==0:
                    lst = [0]
                    random_step = np.random.choice(lst, pts_act[ind].shape[0])
                    random_turn = np.random.choice(lst, pts_act[ind].shape[0])
                else:
                    random_step = np.random.choice(ss_dropnan, pts_act[ind].
↪shape[0])
                    random_turn = np.random.choice(d_diff_dropnan, pts_act[ind].
↪shape[0])

                x = random_step * (np.cos(random_turn)) + pts_act[ind,:,0]
                y = random_step * (np.sin(random_turn)) + pts_act[ind,:,1]

                coords[ind, :, 0] = x
                coords[ind, :, 1] = y

            coords_list.append(coords)
        return coords_list
```

```
    elif isinstance(pts_act, list):
        coords_lst = []
        for ind in range(len(pts_act)):
            act_diff = np.diff(pts_act[ind], axis=0)
            step_size = r

            act_dir = np.arctan2(act_diff[:,1], act_diff[:,0])
            d_diff = np.diff(act_dir)

            steps = np.zeros((number_of_steps, pts_act[ind].shape[0], 2))
            for number in range(number_of_steps):
                coords = np.zeros(pts_act[ind].shape)

                d_diff_dropnan = d_diff[~np.isnan(d_diff)]

                if len(d_diff_dropnan)==0:
                    lst = [0]
                    random_turn = np.random.choice(lst, pts_act[ind].shape[0])
                else:
                    random_turn = np.random.choice(d_diff_dropnan, pts_act[ind].
↪shape[0])

                x = step_size * (np.cos(random_turn)) + pts_act[ind][:,0]
                y = step_size * (np.sin(random_turn)) + pts_act[ind][:,1]

                coords[:, 0] = x
                coords[:, 1] = y

                steps[number] = coords

            coords_lst.append(steps)

        return coords_lst
```

[60]:
```
def group_center(act_pts): #number_ind=None, subset=False):
    """
    Generates x,y positions for the center of the whole group or the center of␣
↪k number of closest individuals
    relative to each individual in the dataset

    Inputs:
        act_pts: list of 3D numpy arrays containing the xy coordinates for a␣
↪focal individual after it has moved r meters as well
                 as the xy coordinates of every other individual at that␣
↪timepoint. The position in the list identifies the focal
                 individual e.g. in new_act_pts_l[0] individual 0 is the focal␣
↪individual; or a 3d numpy array with raw xy
```

```
                positions of animals in space with shape (individual, frame, 2)


        subset: if True, finds center of k number of closest individuals␣
↪(number_ind) relative to focal
                individual
        number_ind: int 2 or above. Designates number of closest individuals to␣
↪consider


    Output:
        center_l: list of 2D numpy arrays with the xy positions for where the␣
↪center of the group is when using an individual's locations e.g.
                  center_l[0] gives the locations for the center of the group␣
↪when using individual 0's locations

        center: 3D numpy array with x,y positions of the center of the group.␣
↪Broadcasted for each individual



        If subset==True: (wrong, see comments)
            c_diff: 3d numpy array with x,y positions of the center of the k␣
↪number of closest individuals
                    relative to focal individual. Has shape (individuals,␣
↪frames, 2)
            center: 3d numpy array with x,y positions of the center of the k␣
↪number of closest individuals.
                    Has shape (individuals, frames, 2)

    """
    if isinstance(act_pts, np.ndarray):
        center = np.zeros(act_pts.shape)

#         if subset==True:
#             for ind in range(act_pts.shape[0]):
#                 diff = act_pts - act_pts[ind]
#                 dis = np.sqrt((diff[:,:,0])**2 + (diff[:,:,1])**2)

#                 #wrong here
#                 sort = np.argsort(np.where(dis==0, dis.max(), dis), axis=0)
#                 sort = sort[0:number_ind, :]

#                 if number_ind<2 or number_ind>=act_pts.shape[0]:
#                     raise ValueError("ValueError: number of individuals needs␣
↪to be >=2 and < act_pts.shape[0]"
```

```
#                                     + str(len(pts_act)-1))

#                 act_loc = np.zeros((sort.shape[0], sort.shape[1]), 2)

#                 for i in range(sort.shape[0]):
#                     for pt in range(sort.shape[1]):
#                         al = act_pts[sort[i, pt], pt]
#                         act_loc[i, pt ,:] = al

#                 center[ind, :, :] = np.nanmean(act_loc, axis=0)

#         else:
#             center[:,:,:] = np.nanmean(act_pts, axis=0)

        center[:,:,:] = np.nanmean(act_pts, axis=0)
        c_diff = center - act_pts

        return center

    elif isinstance(act_pts, list):
        center_l = []
        c_diff_l = []

        for ind in range(len(act_pts)):
            center = np.zeros(act_pts[ind][ind].shape)
            center = np.nanmean(act_pts[ind], axis=0)
            center_l.append(center)

            c_diff = center - act_pts[ind][ind]
            c_diff_l.append(c_diff)

        return center_l
```

[180]:
```
def dir_change(pts_inf, pts_act, random_step=None, rand_step=False):
    """
    Finds the change in direction between the vector pointing from pts_act to␣
↪pts_inf and
    the vector pointing from pts_act to pts_act

    Inputs:
        pts_inf: 3D numpy array with xy positions of social influencers i.e.␣
↪the center of the group, the
                 closest individual, the center of a subset of individuals, etc.
↪ with shape (individual, frame, 2); OR
                 a list of 2D numpy arrays with xy positions of social␣
↪influencers len(pts_inf) = number of individuals
```

```
        pts_act: 3d numpy array with raw xy positions of animals in space with␣
↪shape (individual, frame, 2); OR a list of 2D
                 numpy arrays with xy positions of each individual len(pts_act)␣
↪= number of individuals

        If rand_step==True:
            calulate the change in direction direction between the vector␣
↪pointing from pts_act to pts_inf and
            the vector pointing from pts_act to random_steps
            random_step: list of 3D numpy arrays with xy positions of␣
↪alternative steps for each ind in pts_act len(random_steps) = number of␣
↪steps;
                         OR list of 3D numpy arrays with xy positions of␣
↪alternative steps for each ind where len(random_steps) = number of␣
↪individuals

    Ouputs:
        dir_diff: 2D numpy array with the direction change between the vector␣
↪pointing from pts_act to pts_inf and
                the vector pointing from pts_act to pts_act
        dir_diff_l: list of 1D numpy arrays with the direction change between␣
↪the vector pointing from pts_act to pts_inf and
                  the vector pointing from pts_act to pts_act. The position␣
↪in the list holds the direciton change values for one individual

        If rand_dtep==True:
            dir_diff_l: list of 2D numpy arrays with the direction change␣
↪between the vector pointing from pts_act to pts_inf and
                        the vector pointing from pts_act to random_step.
                        Has shape (individual, frame)
                        If type(pts_act) is a np.ndarray then len(dir_diff_l) =␣
↪number of steps
                        If type(pts_act) is a list then len(dir_diff_l) =␣
↪number of individuals
    """
    if isinstance(pts_act, np.ndarray):
        #direction of vector from pts_act to pts_inf
        loc_rel = pts_inf - pts_act
        inf_dir = np.arctan2(loc_rel[:,:,1], loc_rel[:,:,0])

        if rand_step==True:
            #direction of vector from pts_act to random_steps
            dir_diff_l = []
            for step in range(len(random_step)):
                loc_rel = random_step[step] - pts_act
                direct = np.arctan2(loc_rel[:,:,1], loc_rel[:,:,0])
```

```
                direct = direct[:,:-1]

                dir_diff = inf_dir[:,:-1] - direct
                dir_diff = np.where(dir_diff < -np.pi, (2*np.pi) + dir_diff ,␣
↪dir_diff)
                dir_diff = np.where(dir_diff > np.pi, dir_diff - (2*np.pi),␣
↪dir_diff)

                dir_diff_l.append(dir_diff)

            return dir_diff_l

        else:
            #direction of vector from pts_act to pts_act
            act_diff = np.diff(pts_act, axis=1) #loose frame here, correct in␣
↪inf_dir and direct when rand_step==True
            direct = np.arctan2(act_diff[:,:,1], act_diff[:,:,0])

            dir_diff = inf_dir[:,:-1] - direct
            dir_diff = np.where(dir_diff < -np.pi, (2*np.pi) + dir_diff ,␣
↪dir_diff)
            dir_diff = np.where(dir_diff > np.pi, dir_diff - (2*np.pi),␣
↪dir_diff)

            return dir_diff

    elif isinstance(pts_act, list):

        dir_diff_l = []

        for ind in range(len(pts_act)):
            #direction of vector from pts_act to pts_inf
            loc_rel = pts_inf[ind] - pts_act[ind]
            inf_dir = np.arctan2(loc_rel[:,1], loc_rel[:,0])

            if rand_step==True:
                #direction of vector from pts_act to random_steps
                loc_rel = random_step[ind] - pts_act[ind]

                direct = np.arctan2(loc_rel[:,:,1], loc_rel[:,:,0])
                direct = direct[:,:-1]

                dir_diff = inf_dir[:-1] - direct
                dir_diff = np.where(dir_diff < -np.pi, (2*np.pi) + dir_diff ,␣
↪dir_diff)
```

```
                dir_diff = np.where(dir_diff > np.pi, dir_diff - (2*np.pi),␣
↪dir_diff)

            else:
                #direction of vector from pts_act to pts_act
                act_diff = np.diff(pts_act[ind], axis=0) #loose frame here,␣
↪correct in inf_dir and direct when rand_step==True
                direct = np.arctan2(act_diff[:,1], act_diff[:,0])

                dir_diff = inf_dir[:-1] - direct
                dir_diff = np.where(dir_diff < -np.pi, (2*np.pi) + dir_diff ,␣
↪dir_diff)
                dir_diff = np.where(dir_diff > np.pi, dir_diff - (2*np.pi),␣
↪dir_diff)
#                 print(dir_diff)
            dir_diff_l.append(np.abs(dir_diff))
#             print(dir_diff_l)

        return dir_diff_l
```

[16]:
```
def ind_df(alt_l, chosen, feature_column_header, start_ind_count_from=0):
    """
    Creates a list of pandas dataframes in the correct format for the R␣
↪function 'mclogit'. Each position in the list is for one individual

    Inputs:
        alt_l: list consisting of 2d numpy arrays
               If type(chosen) = np.ndarray then len(alt_l) = number of␣
↪alternative steps generated
               If type(chosen) = list then len(alt_l) = number of individuals
        chosen: 2D numpy array OR a list of 1D numpy arrays
        feature_column_header: string, name of column header for feature going␣
↪into dataframe
        start_ind_count_from: int, used when looking at multiple obeservations␣
↪together
    Ouputs:
        ind_dfl: a list of pandas dataframes

    """
    if isinstance(chosen, list):
        df1_l = []

        for ind in range(len(chosen)):
            chosen_step_id = np.arange(chosen[ind].shape[0])
            chosen_individual_id = np.ones(chosen[ind].shape[0]) * (ind +␣
↪start_ind_count_from)
```

```
            chosen_column = chosen[ind]

            d1 = {'individual': chosen_individual_id, feature_column_header:␣
↪chosen_column, 'label': np.ones(chosen_column.shape), 'step': chosen_step_id}
            df1 = pd.DataFrame(data=d1)

            df1_l.append(df1)

        ind_df_l = []

        for ind in range(len(alt_l)):
            step_id = np.zeros(alt_l[ind].shape)
            individual_id = np.ones(alt_l[ind].flatten().shape[0]) * (ind +␣
↪start_ind_count_from)

            for f in range(step_id.shape[1]):
                step_id[:,f] = step_id[:,f] + f

            step_id = step_id.flatten()

            #create feature col
            alt_flatten = alt_l[ind].flatten()

            d2 = {'individual': individual_id, feature_column_header:␣
↪alt_flatten, 'label': np.zeros(alt_flatten.shape), 'step': step_id}
            df2 = pd.DataFrame(data=d2)

            #combine chosen and alternative dataframes together
            df = np.abs(pd.concat([df1_l[ind], df2]))
            drop_nan_df = df.dropna()

            ind_df_l.append(df)

    elif isinstance(chosen, np.ndarray):
        df1_l = []

        for ind in range(chosen.shape[0]):
            frame_id = np.arange(chosen[ind].shape[0])
            individual_id = np.ones(chosen[ind].shape[0]) * (ind +␣
↪start_ind_count_from)

            feature_column = chosen[ind]

            d1 = {'individual': individual_id, feature_column_header:␣
↪feature_column, 'label': np.ones(feature_column.shape), 'frame': frame_id}
            df1 = pd.DataFrame(data=d1)
```

```
            df1_l.append(df1)

        all_steps = np.zeros((len(alt_l), alt_l[0].shape[1]))
        alt_l_concatenated = np.concatenate(alt_l, axis=1)

        ind_df_l = []

        for ind in range(alt_l_concatenated.shape[0]):
            individual_id = np.ones(alt_l_concatenated[ind].shape[0]) * (ind +␣
↪start_ind_count_from)

            for f in range(all_steps.shape[1]):
                all_steps[:,f] = all_steps[:,f] + f

            frame_id = all_steps.flatten()

            feature_id = alt_l_concatenated[ind]

            d2 = {'individual': individual_id, feature_column_header:␣
↪feature_id, 'label': np.zeros(feature_id.shape), 'frame': frame_id}
            df2 = pd.DataFrame(data=d2)

            df = np.abs(pd.concat([df1_l[ind], df2]))
            drop_nan_df = df.dropna()

            ind_df_l.append(drop_nan_df)

    return ind_df_l
```

[17]:
```
def ind_df_multi_feature(alt_fl, chosen_fl, feature_col_header_list,␣
↪start_ind_count_from=0):
    """
    Creates a pandas dataframe in the correct format for the R function␣
↪'mclogit' for population level analysis with multiple features

    Inputs:
        alt_fl: a list of lists consisting of 2d numpy arrays. len(alt_fl) =␣
↪number of features.
                len(alt_fl[0]) = number of individuals
                OR
                len(alt_fl[0]) = number of alternative steps


        chosen_fl: list consisting of 2d numpy arrays. len(list) = number of␣
↪features
```

```
                    OR
                    list of lists consisting of of 1D numpy arrays
        start_ind_count_from: int, used when looking at multiple obeservations␣
↪together

    Ouputs:
        feature_dfl: a pandas df with columns 'individual', 'feature' * n,␣
↪'label', and 'frame'

    """

    feature_dfl = []
    for feature in range(len(chosen_fl)):
        ind_dfl = ind_df(alt_fl[feature], chosen_fl[feature],␣
↪feature_col_header_list[feature], start_ind_count_from)
        feature_dfl.append(ind_dfl)

    for feature in range(1, len(feature_col_header_list)):
        for idv_df in range(len(feature_dfl[0])):
            feature_dfl[0][idv_df].insert(feature+1,␣
↪feature_col_header_list[feature],␣
↪feature_dfl[feature][idv_df][feature_col_header_list[feature]])
    ind_df_multi_feature_l = feature_dfl[0]
    return ind_df_multi_feature_l
```

[179]:
```
def dc_feature(inf_pts, act_pts, r=None):
    """
    Calculates the change in direction towards a point of influence for␣
↪collected data and generated data

    Inputs:
        inf_pts: 3D numpy array with xy positions of social influencers i.e.␣
↪the center of the group, the
                 closest individual, the center of a subset of individuals, etc.
↪ with shape (individual, frame, 2); OR
                 a list of 2D numpy arrays with xy positions of social␣
↪influencers len(pts_inf) = number of individuals
        act_pts: either list of 2D numpy arrays containing the xy coordinates␣
↪for each individual after they have moved at least r meters(spatial␣
↪discretization),
                 or a 3d numpy array with raw xy positions of animals in space␣
↪with shape (individual, frame, 2) (temporal discretization)
        r: int; distance focal individual should travel before a step is␣
↪recorded
    Outputs:
        If type(pts_act) is a np.ndarray:
```

```
            act_dc: 2D numpy array with the direction change between the vector␣
↪pointing from pts_act to pts_inf and
                    the vector pointing from pts_act to pts_act
            alt_dcl: list of 2D numpy arrays containing the change in direction␣
↪towards the point of influence for every individuals alternative steps
                     len(alt_dcl) = number of steps

        If type(act_pts) is a list:
            act_dcl: list of 1D numpy arrays with the direction change between␣
↪the vector pointing from pts_act to pts_inf and
                     the vector pointing from pts_act to pts_act. Each position␣
↪in the list holds the direciton change values for each individual
            alt_dcl: list of 2D numpy arrays containing the change in direction␣
↪towards the point of influence for every individuals alternative steps
                     len(alt_dcl) = number of individuals
    """

    if isinstance(act_pts, np.ndarray):

        act_dc = dir_change(inf_pts, act_pts)

        alt_sl = random_steps(act_pts)
        alt_dcl = dir_change(inf_pts, act_pts, alt_sl, rand_step=True)

        return act_dc, alt_dcl

    elif isinstance(act_pts, list):
        act_dcl = dir_change(inf_pts, act_pts)

        alt_sl = random_steps(act_pts, r)
        alt_dcl = dir_change(inf_pts, act_pts, alt_sl, rand_step=True)

        return act_dcl, alt_dcl
```

[225]:
```
def save(ind_dfl, filename, t, drop_lst, list_feature_names,␣
↪start_ind_count_from=0, temporal_data=False,␣
↪save_individual_level_data=True, save_poplation_level_data=True):
    """
    Drops individuals with bad data and saves their dfs. Also returns a␣
↪population df containing only the individuals with good data

    Inputs:
        filename: string

        If temporal_data==True:
```

```
            t: int desigating the new timescale at witch the data is being␣
↪analyzed
        If temporal_data==False (i.e. spatial data is entered):
            t: int designating the step size of the individual

        drop_lst: list of individuals to be dropped from dataset
        ind_dfl: list of dataframes, one for each individual
        list_feature_names: list of strings for column headers of each feature
    Output:
        pop_df: dataframe consisting of all individuals in the dataset with␣
↪good data (individuals with bad data has been dropped)
        drop: list of individuals to be dropped from dataset. Filters␣
↪individuals with too many NaNs, individuals with all the same true and false␣
↪values, and
              individuals with empty dfs
    """

    if save_individual_level_data==True:
        for i in range (len(ind_dfl)):
            if temporal_data==True:
                ind_dfl[i].to_csv(r'/Users/Taylor/Documents/' + filename +␣
↪'_ind' + str(i + start_ind_count_from) +
                                                  'df_t' + str(t) + '.csv', index =␣
↪False, header=True)
            else:
                ind_dfl[i].to_csv(r'/Users/Taylor/Documents/' + filename +␣
↪'_ind' + str(i + start_ind_count_from) +
                                                  'df_' + str(t) + 'meter_step_size.
↪csv', index = False, header=True)

    #look at each feature column, if any of them have the same exact value for␣
↪every input in the column then make note to have that ind dropped
    same_num_dropl = []

    for idv in range(len(ind_dfl)):
        for n in list_feature_names:
            check = is_same(ind_dfl[idv][n])
            if check==True:
                same_num_dropl.append(idv)
                break

    #drop 'bad data' from population df
    drop = np.concatenate((drop_lst, same_num_dropl))
    drop = np.unique(drop)
    drop = drop.astype(int)
```

```
    drop_copy = np.copy(drop)
    for ind in range(drop_copy.shape[0]):
        if len(drop_copy) > 1:
            if drop_copy[ind] >= drop_copy[1]:
                drop_copy[ind] = drop_copy[ind] - ind

        del ind_dfl[drop_copy[ind]]

    if len(ind_dfl) > 0:
        pop_df = pd.concat(ind_dfl)

        if save_poplation_level_data==True:
            if temporal_data==True:
                pop_df.to_csv(r'/Users/Taylor/Documents/' + filename +␣
↪'_population_df_t' +
                                    str(t) + '.csv', index = False, header=True)
            else:
                pop_df.to_csv(r'/Users/Taylor/Documents/' + filename +␣
↪'_population_df_' +
                                    str(t) + 'meter_step_size.csv', index = False,␣
↪header=True)


        if save_individual_level_data==True:
            #drop individuals with 'bad data' from saved files
            for ind in drop:
                if temporal_data==True:
                    os.remove(filename + '_ind' + str(ind +␣
↪start_ind_count_from) + 'df_t' + str(t) + '.csv')
                else:
                    os.remove(filename + '_ind' + str(ind +␣
↪start_ind_count_from) + 'df_' + str(t) + 'meter_step_size.csv')
    else:
        pop_df = 'empty dataframe'
    return pop_df, drop
```

[20]:
```
def is_same(col):
    """
    Boolean testing whether a column in a pandas dataframe has the same exact␣
↪value in every row

    Input:
        col: column of dataframe
    Output:
        Boolean saying whether all values in the column are the same. True if␣
↪true, False if false
```

```
    """
    if col.shape[0]>0:
        col_np = col.to_numpy()
        boolean = (col_np[0] == col_np[1:]).all()
    else:
        boolean = False
    return boolean
```

[715]:
```
def calculate_fraction(act_pts, focal, r, fraction=True):
    """
    Calcualtes fraction of individuals within a given radius of a focal individual

    Inputs:
        act_pts: 3D numpy array (ind, frame, xy); OR list of 3D numpy arrays containing the xy coordinates for a focal individual after it has moved r meters as well
                 as the xy coordinates of every other individual at that timepoint.

        focal: 3D numpy array (ind, frame, xy). Focal indvdual. OR list of 3D numpy arrays containing the xy coordinates for a focal individual after it has moved r meters as well
                as the xy coordinates of every other individual at that timepoint.
        r: int; radius around focal individual
        fraction: if False does not calculate fraction of individuals within a givin radius, gives counts of individuals within radius
    Output:
        counts: 2D numpy array containing the fraction of individuals within the given radius of the focal individual. shape (ind, frame)
        counts_l: list of 1D numpy arrays containing the fraction of individuals within the given radius of the focal individual. The focal ind
                  is equal to the position in act_pts list.
        If fraction==False:
            counts: 2D numpy array containing the absolute number of individuals within the given radius of the focal individual. shape (ind, frame)
            counts_l: list of 1D numpy arrays containing the absolute number of individuals within the given radius of the focal individual. The focal ind
                  is equal to the position in act_pts list.

    """
    if isinstance(act_pts, np.ndarray): #temporal
```

```
        counts = np.zeros((act_pts.shape[0], act_pts.shape[1]-1)) #need drop␣
↪last frame to match len of direction change feature

        for ind in range(act_pts.shape[0]):
            loc = (act_pts[:,:,0] - focal[ind,:,0])**2 + (act_pts[:,:,1] -␣
↪focal[ind,:,1])**2
            loc = np.where((loc<=r**2), 1, 0)
            loc[ind] = 0 #eliminating focal individual from count

            if fraction==True:
                count = np.nansum(loc, axis=0)/(np.sum(~np.isnan(act_pts[:,:
↪,0]), axis=0)-1)
            elif fraction==False:
                count = np.nansum(loc, axis=0)

            counts[ind] = count[:-1] #need drop last frame to match len of␣
↪direction change feature
        return counts

    elif isinstance(act_pts, list): #spatial

        counts_l = []
        for ind in range(len(act_pts)):
            counts = np.zeros((focal[ind].shape[0], act_pts[ind].shape[1]-1))␣
↪#need drop last frame to match len of direction change feature

            for i in range(focal[ind].shape[0]):
                loc = (act_pts[ind][:,:,0] - focal[ind][i,:,0])**2 +␣
↪(act_pts[ind][:,:,1] - focal[ind][i,:,1])**2
                loc = np.where((loc<=r**2), 1, 0)
                loc[ind] = 0 #eliminating focal individual from count
            #changes start
                if fraction==True:
                    count = np.nansum(loc, axis=0)/(np.sum(~np.
↪isnan(act_pts[ind][:,:,0]), axis=0)-1)
                elif fraction==False:
                    count = np.nansum(loc, axis=0)
            #changes end
                counts[i] = count[:-1] #need drop last frame to match len of␣
↪direction change feature

            counts_l.append(counts)

        return counts_l
```

[1039]:
```
def get_trail_values_from_utm(trail_mask, mask_info_df, utms):
    """ Get the trail value for every utm point in trail mask array.

    Args:
        trail_mask: 2D np array with values from 0 to 1 with 1 being a trail
        mask_info_df: pandas dataframe with x_origin, y_origin, pixel_width,␣
↪pixel_height values
        utms: array of shape (steps, 2) or (alt_steps, steps, 2)

    Returns:
        array of shape (steps) or (alt_steps, steps)

    """

    def _get_trail_values_from_track(trail_mask, mask_info_df, utm_track):
        raster_track = kmap.utm_to_raster_track(utm_track, mask_info_df.loc[0,␣
↪'x_origin'],
                                                mask_info_df.loc[0, 'y_origin'],
                                                mask_info_df.loc[0,␣
↪'pixel_width'],
                                                mask_info_df.loc[0,␣
↪'pixel_height'],
                                                1.0)

        map_vals = np.array([np.nan for _ in range(raster_track.shape[0])])

        map_vals[~np.isnan(raster_track[:,0])] = trail_mask[raster_track[~np.
↪isnan(raster_track[:,0]),1].astype(int),
                                                            raster_track[~np.
↪isnan(raster_track[:,0]),0].astype(int)]

        return map_vals

    if len(utms.shape) == 2:
        return _get_trail_values_from_track(trail_mask, mask_info_df, utms)

    elif len(utms.shape) == 3:
        map_vals = []
        for utm_track in utms:
            map_vals.append(_get_trail_values_from_track(trail_mask,␣
↪mask_info_df, utm_track))

        map_vals = np.stack(map_vals)

        return map_vals
    else:
```

```
        print('utms needs to be shape 2 or 3.')



def trail_feature(alt_pts, act_pts, trail_mask, mask_info_df):
    """
    Calculates if focal individiual and its alternative steps are on a trail

    Inputs:
        alt_pts: list of 3D numpy arrays with new xy positions of animals in␣
↪pts_act. alt_pts[0] = xy coordinates
                of an alternative step for each individual
        act_pts: 3D numpy array with raw xy positions of animals in space with␣
↪shape (individual, frame, 2)
        trail_mask: 2D numpy array 0 if no trail 1 if trail in that location
        mask_info_df: info about how trail map relates to utm coordinates␣
↪('x_origin', 'y_origin', 'pixel_height', 'pixel_width')
    Output:
        individual_specific_counts_l: 1D numpy array containing the fraction of␣
↪individuals within the given radius of the focal individual. shape (#steps,)
        alt_countsl: list of 2d numpy arrays containing the fraction of␣
↪individuals within the given radius of a focal individuals alternative step.
                    shape (ind, frame)

    """
    chosen_trail_vals = []
    alt_trail_valsl = []
    for act_p, alt_p in zip(act_pts, alt_pts):
        if not np.any(np.isnan(act_p)):
            chosen_trail_vals.append(get_trail_values_from_utm(trail_mask,␣
↪mask_info_df, act_p))
            alt_trail_valsl.append(get_trail_values_from_utm(trail_mask,␣
↪mask_info_df, alt_p))

    return chosen_trail_vals, alt_trail_valsl
```

[972]:
```
def sd_feature(alt_pts, act_pts, r, fraction=True):
    """
    Calculates fraction of all group mates within a given radius of a focal␣
↪individual and its alternative steps

    Inputs:
        alt_pts: list of 3D numpy arrays with new xy positions of animals in␣
↪pts_act. alt_pts[0] = xy coordinates
                of an alternative step for each individual
```

```
        act_pts: 3D numpy array with raw xy positions of animals in space with␣
↪shape (individual, frame, 2)
        r: int; radius around focal individual
    Output:
        individual_specific_counts_l: 1D numpy array containing the fraction of␣
↪individuals within the given radius of the focal individual. shape (#steps,)
        alt_countsl: list of 2d numpy arrays containing the fraction of␣
↪individuals within the given radius of a focal individuals alternative step.
                    shape (ind, frame)

    """
    if isinstance(act_pts, np.ndarray):
        counts = calculate_fraction(act_pts, act_pts, r, fraction)

        alt_countsl = []
        for step in range(len(alt_pts)):
            alt_counts = calculate_fraction(act_pts, alt_pts[step], r, fraction)
            alt_countsl.append(alt_counts)

        return counts, alt_countsl

    if isinstance(act_pts, list):
        counts_l = calculate_fraction(act_pts, act_pts, r, fraction)

        #counts_l is a 3D np array. code below extracts the focal ind data from␣
↪that array, other info is not needed
        individual_specific_counts_l = []

        for ind in range(len(act_pts)):
            counts = counts_l[ind][ind]
            individual_specific_counts_l.append(counts)

        alt_countsl = calculate_fraction(act_pts, alt_pts, r, fraction)

        return individual_specific_counts_l, alt_countsl
```

[712]:
```
def count_ind(act_pts, focal, r, time_index, time, temporal_data=False,␣
↪double_count=True, fraction=False):
    """
    Calcualtes number of individuals that have occupied a potential location␣
↪within the past time

    Inputs:
        act_pts: 3D numpy array (ind, frame, xy)
        focal: 3D numpy array (ind, frame, xy). Focal indvdual
        r: int; radius around focal individual
```

```
        time: int. How many frames to look back through
    Output:
        counts: 2D numpy array containing the number of individuals that have␣
↪occupied a potential location within the past time
        If fraction==True:
            counts: 2D numpy array containing the fraction of individuals that␣
↪have occupied a poetential location within the past. Individuals will be
                    counted more than once
        If double_count==False:
            counts: 2D numpy array containing the number of individuals that␣
↪have occupied a potential location within the past time. Individuals will not
                    be double counted
            If fraction==True:
                counts: 2D numpy array containing the fraction of individuals␣
↪that have occupied a poetential location within the past. Individuals will␣
↪not be
                        counted more than once
    """
    if time > act_pts.shape[1]:
        raise ValueError('time exceeds length of video')

    if temporal_data==True:
        counts = np.zeros((act_pts.shape[0], act_pts.shape[1]))

        #need to fill in with NaNs to keep indexing intact when making a df
        counts[:,0:time] = counts[:,0:time] * np.NaN

        for ind in range(act_pts.shape[0]):
            for f in range(act_pts.shape[1]-time):
                loc = (act_pts[:,f:time+f,0] - focal[ind,time+f,0])**2 +␣
↪(act_pts[:,f:time+f,1] - focal[ind,time+f,1])**2
                loc = np.where((loc<=r**2), 1, 0)
                loc[ind] = 0 #eliminating focal individual from count

                if double_count==False:

                    loc = np.any(loc, axis=1)

                    if fraction==False:
                        count = np.sum(loc)
                    elif fraction==True:
                        num_ind_timeframe = np.sum(np.any(~np.isnan(act_pts[:,f:
↪time+f,0]), axis=1))
                        count = np.sum(loc)/(num_ind_timeframe-1)
                else:
                    if fraction==False:
```

```
                    count = np.sum(loc)
                elif fraction==True:
                    count = np.sum(loc)/(np.sum(np.sum(~np.isnan(act_pts[:
↪,f:time+f,0]), axis=0))-1)

            counts[ind,f+time] = count
        return counts[:,:-1]

    else: #for spatial
        if isinstance(focal, np.ndarray): #for chosen steps
            counts_l = []
            for ind in range(len(time_index)):
                counts = np.zeros(time_index[ind].shape[0])

                for f in range(time_index[ind].shape[0]):
                    first_frame = time_index[ind][f]-time
                    end_frame = time_index[ind][f]

                    loc = (act_pts[:, first_frame : end_frame ,0] - focal[ind,␣
↪end_frame ,0])**2 + (act_pts[:, first_frame : end_frame ,1]
                                                                              ␣
↪                    - focal[ind, end_frame ,1])**2
                    loc = np.where((loc<=r**2), 1, 0)
                    loc[ind] = 0 #eliminating focal individual from count

                    if double_count==False:

                        loc = np.any(loc, axis=1)
                    #only thing I changed about code starts here
                        if fraction==False:
                            count = np.sum(loc) #this was directly from old␣
↪code (with nice values)
                        elif fraction==True:
                            #divided by the number of individuals that could␣
↪have potentially been counted throughout the timeframe
                            #i.e. if any individual shows up at all in the␣
↪timeframe it will be used in the denominator
                            #never exceeds possible number of indiviuals

                            num_ind_timeframe = np.sum(np.any(~np.
↪isnan(act_pts[:,first_frame : end_frame,0]), axis=1))
                            count = np.sum(loc)/(num_ind_timeframe-1)
                    else:
                        if fraction==False:
                            count = np.sum(loc)
                        elif fraction==True:
```

```
                        count = np.sum(loc)/(np.sum(np.sum(~np.
↪isnan(act_pts[:,first_frame : end_frame,0]), axis=0))-1)
                    #ends here
                    counts[f] = count

                counts_l.append(counts[:-1])

            return counts_l

        if isinstance(focal, list): #for alternative steps
            counts_l = []
            for ind in range(len(time_index)):
                counts = np.zeros((focal[ind].shape[0], focal[ind].shape[1]))

                for step in range(focal[ind].shape[0]):
                    for f in range(focal[ind].shape[1]):
                        first_frame = time_index[ind][f]-time
                        end_frame = time_index[ind][f]

                        loc = (act_pts[:, first_frame : end_frame ,0] -␣
↪focal[ind][step, f ,0])**2 + (act_pts[:, first_frame : end_frame ,1]
                                                                        ␣
↪                       - focal[ind][step, f ,1])**2
                        loc = np.where((loc<=r**2), 1, 0)
                        loc[ind] = 0 #eliminating focal individual from count

                        if double_count==False:

                            loc = np.any(loc, axis=1)
                        #changes start (same as above)
                            if fraction==False:
                                count = np.sum(loc)
                            elif fraction==True:
                                num_ind_timeframe = np.sum(np.any(~np.
↪isnan(act_pts[:,first_frame : end_frame,0]), axis=1))
                                count = np.sum(loc)/(num_ind_timeframe-1)

                        else:
                            if fraction==False:
                                count = np.sum(loc)
                            elif fraction==True:
                                count = np.sum(loc)/(np.sum(np.sum(~np.
↪isnan(act_pts[:,first_frame : end_frame,0]), axis=0))-1)
                        #changes end
                        counts[step, f] = count
                counts_l.append(counts[:,:-1])
```

```
            return counts_l
```

[24]:
```
##still working on this

def cal_ind_figure(act_pts, focal_pt, resolution, buffer, frame, individual, r,␣
↪fraction=True):
    """
    Calcualtes number of individuals within a given radius of a focal individual

    Inputs:
        act_pts: 3D numpy array (ind, frame, xy)
        focal_pt: 1D numpy array containing x coordinate and y coordinate of␣
↪individual that will be center of figure
        resolution: how many points per 1 increment moved
        buffer: max distance away from center in both the +-x and +-y direction
        frame: int; at which frame the xy positions for each individual are␣
↪being extracted
        individual: individual being selected as the focal individual
        r: int; radius around focal individual
        num_ind_drop: number of individuals to drop from the data (problem with␣
↪too many NaNs)
    Output:
        counts: 2D numpy array containing the fraction of individuals within␣
↪the given radius of the focal individual

    """
    counts = np.zeros((int((buffer*resolution)*2+1),␣
↪int((buffer*resolution)*2+1)))
    column = 0
    row = int((buffer*resolution)*2)

    for x in np.linspace(focal_pt[0]-buffer, focal_pt[0]+buffer,␣
↪int((buffer*resolution)*2+1)):
        for y in np.linspace(focal_pt[1]-buffer, focal_pt[1]+buffer,␣
↪int((buffer*resolution)*2+1)):

            loc = (act_pts[:, 0:frame, 0] - x)**2 + (act_pts[:, 0:frame, 1] -␣
↪y)**2 #change is instead of looking at all ind locations in 1 frame look at␣
↪all ind locations up until frame
            loc = np.where((loc<=r**2), 1, 0)
            loc[individual] = 0 #eliminating focal individual from count

            if fraction==True:
                count = np.nansum(loc, axis=0)/(np.sum(~np.isnan(act_pts[:,:
↪,0]), axis=0)-1)
            elif fraction==False:
```

```
                count = np.nansum(loc, axis=0)

            counts[row, column] = count
            row = row - 1
        column = column + 1
        row = int((buffer*resolution)*2)
    return counts
```

[713]:
```
def ru_feature(alt_pts, act_pts, r, time_index, time, temporal_data=False,␣
↪double_count=True, fraction=False):
    """
    Calculates number of individuals that have occupied a potential location␣
↪within the past 4.5min for a focal indivdiual and its alternative steps

    Inputs:
        time: int. How many frames to look back through (30 frames = 1sec)
        alt_pts: list of 3D numpy arrays with new xy positions of animals in␣
↪pts_act. alt_pts[0] = xy coordinates
                of an alternative step for each individual
        act_pts: 3D numpy array with raw xy positions of animals in space with␣
↪shape (individual, frame, 2)
        r: int; radius around focal individual
    Output:
        counts: 1D numpy array containing the fraction of individuals within␣
↪the given radius of the focal individual. shape (ind, frame)
        alt_countsl: list of 2d numpy arrays containing the fraction of␣
↪individuals within the given radius of a focal individuals alternative step.
                    shape (ind, frame)

    """
    if temporal_data==True:
        counts = count_ind(act_pts, act_pts, r, time_index, time,␣
↪temporal_data, double_count, fraction)

        alt_countsl = []
        for step in range(len(alt_pts)):
            alt_counts = count_ind(act_pts, alt_pts[step], r, time_index, time,␣
↪temporal_data, double_count, fraction)
            alt_countsl.append(alt_counts)
        return counts, alt_countsl

    else:
        counts_l = count_ind(act_pts, act_pts, r, time_index, time,␣
↪temporal_data, double_count, fraction)
```

```
        alt_countsl = count_ind(act_pts, alt_pts, r, time_index, time,␣
↪temporal_data, double_count, fraction)

        return counts_l, alt_countsl
```

[26]:
```
def cal_fraction_figure(act_pts, focal_pt, resolution, buffer, frame,␣
↪individual, r, num_ind_drop=0, fraction=True):
    """
    Calcualtes fraction of individuals within a given radius of a focal␣
↪individual

    Inputs:
        act_pts: 3D numpy array (ind, frame, xy)
        focal_pt: 1D numpy array containing x coordinate and y coordinate of␣
↪individual that will be center of figure
        resolution: how many points per 1 increment moved
        buffer: max distance away from center in both the +-x and +-y direction
        frame: int; at which frame the xy positions for each individual are␣
↪being extracted
        individual: individual being selected as the focal individual
        r: int; radius around focal individual
        num_ind_drop: number of individuals to drop from the data (problem with␣
↪too many NaNs)
    Output:
        counts: 2D numpy array containing the fraction of individuals within␣
↪the given radius of the focal individual

    """
    counts = np.zeros((int((buffer*resolution)*2+1),␣
↪int((buffer*resolution)*2+1)))
    column = 0
    row = int((buffer*resolution)*2)

    for x in np.linspace(focal_pt[0]-buffer, focal_pt[0]+buffer,␣
↪int((buffer*resolution)*2+1)):
        for y in np.linspace(focal_pt[1]-buffer, focal_pt[1]+buffer,␣
↪int((buffer*resolution)*2+1)):

            loc = (act_pts[:, frame, 0] - x)**2 + (act_pts[:, frame, 1] - y)**2
            loc = np.where((loc<=r**2), 1, 0)
            loc[individual] = 0 #eliminating focal individual from count

            if fraction==True:
                count = np.nansum(loc, axis=0)/((act_pts.shape[0]-1) -␣
↪num_ind_drop)
            if fraction==False:
```

```
                count = np.nansum(loc, axis=0)

            counts[row, column] = count
            row = row - 1
        column = column + 1
        row = int((buffer*resolution)*2)
    return counts
```

[27]:
```
def spatial_dis(act_pts, r, resolution=1):
    """
    Adjusts the scale from temporal to spatial. Points are now recorded at a␣
 ↪constant spatial rate e.g. one point for every r meters moved.
    Locations are stored in spatial. The corresponding frame or timepoint for␣
 ↪the location in spatial is stored in time_index

    Inputs:
        act_pts: 3D numpy array (ind, frame, xy)
        r: int; distance focal individual should travel before a step is␣
 ↪recorded
        resolution: int, at every 'blank' frame, data will be pulled
    Output:
        new_act_pts_l:  list of 3D numpy arrays containing the xy coordinates␣
 ↪for a focal individual after it has moved r meters as well
                        as the xy coordinates of every other individual at that␣
 ↪timepoint. The position in the list identifies the focal
                        individual e.g. in new_act_pts_l[0] individual 0 is the␣
 ↪focal individual
        time_index: list of 1D numpy arrays containing the indexs for the frame␣
 ↪at which each individual has moved at least r meters
        spatial: list of 2D numpy arrays containing the xy coordinates for each␣
 ↪individual after they have moved at least r meters
        drop_lst: list of individuals to be dropped from dataset. Will be␣
 ↪dropped for insufficient amount of data

    """
    time_index = []
    spatial = []
    drop_lst = []

    for ind in range(act_pts.shape[0]):
        frame = 0
        temp = []
        spat = []
#         print(ind)
        #should these be here? puts in the first xy coordinate
        temp.append(0)
```

```
        spat.append(act_pts[ind,0,:])
        for f in range(0, act_pts.shape[1], resolution):
            loc = (act_pts[ind,f,0] - act_pts[ind,frame,0])**2 +␣
↪(act_pts[ind,f,1] - act_pts[ind,frame,1])**2
            if loc>=r**2:
                frame = f
                temp.append(f)
                spat.append(act_pts[ind,f,:])

        temp = np.asarray(temp)

        #Less than equal to 3 because need at least 3 xy locations to do the␣
↪calc change in direction
        if len(temp)<=3:
            spat = np.asarray(spat)
            drop_lst.append(ind)
        else:
            spat = np.vstack(spat)

        time_index.append(temp)
        spatial.append(spat)

    new_act_pts_l = []

    for ind in range(len(time_index)):
        coords = np.zeros((len(spatial), spatial[ind].shape[0], 2))
        for s in range(time_index[ind].shape[0]):
            new_act_pts = act_pts[:, time_index[ind][s], :]
            coords[:, s, :] = new_act_pts
        new_act_pts_l.append(coords)

    return new_act_pts_l, spatial, time_index, drop_lst
```

[28]:
```
def beta_df(act_pts, scale_adjustment, drop_lstl, filename):
    """
    Generates a df consisting of beta values for each individual, for each␣
↪feature, at every time series

    Inputs:
        act_pts: 3D numpy array with raw xy positions of animals in space with␣
↪shape (individual, frame, 2)
        scale_adjustment: list of time scale adjustments (ints)
        drop_lstl: list of lists containing individuals to drop for each time␣
↪series in times
        filename: string, name of file from mclogit
    Output:
```

```
        beta_df: df consisting of beta values for each individual, for each␣
↪feature, at every time series

    """
    if isinstance(act_pts, np.ndarray):
        ind_idl = []
        t_idl = []

        for time in range(len(scale_adjustment)):

            #create 'individual' and 'timescale' column
            ind_id = np.zeros((1, act_pts.shape[0]+1))
            t_id = np.zeros((1, act_pts.shape[0]+1))

            for i in range(ind_id.shape[1]):
                ind_id[:,i] = ind_id[:,i] + i
            #     ind_id[:,-1] = -1 #bad when plotting

            for i in range(t_id.shape[0]):
                t_id[i] = t_id[i] + scale_adjustment[time]

            #drop individual with 'bad data'
            ind_id = np.delete(ind_id, drop_lstl[time], axis=1)
            t_id = np.delete(t_id, drop_lstl[time], axis=1)

            ind_id = np.concatenate((ind_id), axis=0)
            t_id = np.concatenate((t_id), axis=0)

            ind_idl.append(ind_id)
            t_idl.append(t_id)

        ind_id = np.concatenate(ind_idl)
        t_id = np.concatenate(t_idl)

        #put all columns together in one dataframe
        beta_df = pd.read_csv(filename + "_output.csv")
        beta_df.insert(0, 'individual', ind_id, True)
        beta_df.insert(beta_df.shape[1], 'timescale', t_id, True)

    elif isinstance(act_pts, list):

        ind_idl = []
        step_size_idl = []

        for scale in range(len(scale_adjustment)):
            ind_id = np.arange(len(act_pts)+1)
            step_size_id = np.ones(len(act_pts)+1) * scale_adjustment[scale]
```

```
            ind_id = np.delete(ind_id, drop_lstl[scale])
            step_size_id = np.delete(step_size_id, drop_lstl[scale])

            ind_idl.append(ind_id)
            step_size_idl.append(step_size_id)

        ind_id = np.concatenate(ind_idl)
        step_size_id = np.concatenate(step_size_idl)

        #put all columns together in one dataframe
        beta_df = pd.read_csv(filename + "_output.csv")
        beta_df.insert(0, 'individual', ind_id, True)
        beta_df.insert(beta_df.shape[1], 'step_size', step_size_id, True)

    return beta_df
```

[29]:
```
def multi_observations_beta_df(scale_adjustment, radius_around_ind, filename):
    """
    Generates a df consisting of beta values for entire observations, for each␣
↪feature

    Inputs:
        scale_adjustment: list of scale adjustments (ints)
        filename: string, name of file from mclogit
    Output:
        beta_df: df consisting of beta values for each observations, for each␣
↪feature

    """
    step_size = []
    radius_id = []
    for scale in range(len(scale_adjustment)):
        step_size_id = np.ones(len(radius_around_ind)) * scale_adjustment[scale]
        radius_around_ind_id = np.zeros(len(radius_around_ind)) + np.
↪asarray(radius_around_ind)

        step_size.append(step_size_id)
        radius_id.append(radius_around_ind_id)

    radius_id = np.concatenate(radius_id)
    step_size_id = np.concatenate(step_size)

    #put all columns together in one dataframe
    beta_df = pd.read_csv(filename + "_output.csv")
    beta_df.insert(0, 'step_size', step_size_id, True)
    beta_df.insert(beta_df.shape[1], 'radius_size_around_ind', radius_id, True)
```

```
    return beta_df
```

[716]:

[751]:
```
# import sys
# # To see koger_general_functions
```

[752]:

[774]:
```
data_names = [positions_names, before_scare_positions_names,␣
 ↪after_scare_positions_names,
              plains_positions_names, plains_before_scare_positions_names,␣
 ↪plains_before_scare_positions_names,
             grevy_positions_names, grevy_before_scare_positions_names,␣
 ↪grevy_after_scare_positions_names]

data_names = ['observation083', 'observation108', 'observation086',␣
 ↪'observation074']
utm_arrays = [positions[10], positions[14], positions[11], positions[8]]


#             print(np.nansum(map_vals), np.nansum(vals))
```

```
observation083
observation108
observation086
observation074
```

[899]:

[900]:
```
name_index
```

[900]:
```
[['observation074', 'observation083', 'observation086', 'observation108'],
 ['observation074', 'observation083'],
 ['observation074', 'observation083'],
 ['observation074', 'observation083', 'observation086', 'observation108'],
 ['observation074', 'observation083'],
 ['observation074', 'observation083'],
 [],
 [],
 []]
```

[870]:
```
name_index
```

[870]:
```
[[8, 10, 11, 14], [4, 6], [3, 4], [], [], [], [3, 4, 5, 7], [2, 3], [2, 3]]
```

[841]:
```
print(len(data_list_map))
```

```
9
```

[741]:
```
data_list = [positions, before_scare_positions, after_scare_positions,
       plains_positions, plains_before_scare_positions,␣
 ↪plains_after_scare_positions,
       grevy_positions, grevy_before_scare_positions,␣
 ↪grevy_after_scare_positions]

data_list = [plains_before_scare_positions]

for data in range(len(data_list)):
#     for distance in step_size:
#         for radius in radius_around_ind:
#             diff_radius = []
#             start_ind_id_count = 0
    for observation in range(len(data_list[data])):
        new_coords_l, spatial, time_index, drop_lst =␣
 ↪spatial_dis(data_list[data][observation], distance, resolution)
        alt_steps = random_steps(spatial, distance)

    #     chosen_gtl = []
    #     alt_gtl = []

    #     for ind in range(len(spatial)):
    #         ind_trail_values = get_trail_values_from_utm(trail_mask,␣
 ↪mask_info_df, spatial[ind])
    #         alt_step_trail_values = get_trail_values_from_utm(trail_mask,␣
 ↪mask_info_df, spatial[ind])

    #         chosen_gtl.append(ind_trail_values)
    #         alt_gtl.append(alt_step_trail_values)
```

[949]:
```
mask_info_df
```

[949]:

	x_origin	y_origin	pixel_width	pixel_height
0	266493.55101	37863.51728	0.21296	-0.21296

0.0.1 Spatial discretization

[193]:
```
folder_names = ['all_features_model']

for folder in folder_names:
    os.makedirs(folder)
```

[205]:
```
print(len(after_scare_positions))
for obs in after_scare_positions:
    print(obs.shape)
```

```
6
(5, 8531, 2)
(9, 19624, 2)
(10, 5600, 2)
(12, 18979, 2)
(11, 9645, 2)
(18, 18029, 2)
```

Model

[277]:
```
data_list = [positions, before_scare_positions, after_scare_positions,
        plains_positions, plains_before_scare_positions,␣
 ↪plains_after_scare_positions,
        grevy_positions, grevy_before_scare_positions,␣
 ↪grevy_after_scare_positions]
# data_list = [plains_before_scare_positions]

folder_path = '/Users/Taylor/Documents/all_features_model/'

filenames = ['df_', 'before_scare_df_', 'after_scare_df_',
             'plains_df_', 'plains_before_scare_df_', 'plains_after_scare_df_',
             'grevy_df_', 'grevy_before_scare_df_', 'grevy_after_scare_df_']
# filenames = ['plains_before_scare_df_']

step_size = [5,10]
feature_col_headers = ['group_center', 'social_density', 'recently_used_space']
resolution = 1
radius_around_ind = [1, 5, 10, 15, 20]
look_back = 3600

for data in range(len(data_list)):
    for distance in step_size:
        for radius in radius_around_ind:
            diff_radius = []
            start_ind_id_count = 0
            for observation in range(len(data_list[data])):
                new_coords_l, spatial, time_index, drop_lst =␣
 ↪spatial_dis(data_list[data][observation], distance, resolution)
                alt_steps = random_steps(spatial, distance)

                #group center
                cl = group_center(new_coords_l)
                chosen_gcl, alt_gcl = dc_feature(cl, spatial, distance)
```

```
                if len(chosen_gcl) == len(drop_lst):
                    continue

                # trails
                obs_name = data_names[data][observation]
                trail_mask = observation_maps_dict[obs_name]['trail_mask']
                mask_info_df = observation_maps_dict[obs_name]['mask_info_df']
                chosen_trail, alt_trail = trail_feature(alt_steps, spatial,␣
↪trail_mask, mask_info_df)


                #social density
                chosen_sdl, alt_sdl = sd_feature(alt_steps, new_coords_l,␣
↪radius, fraction=False)

                #recently used space
                chosen_rul, alt_rul = ru_feature(alt_steps,␣
↪data_list[data][observation], radius, time_index, look_back,␣
↪temporal_data=False, double_count=False)

                alt_fl = [alt_gcl, alt_sdl, alt_rul]
                chosen_fl = [chosen_gcl, chosen_sdl, chosen_rul]

                filename = 'multi_feature_positions' + str(observation)

                multi_feature_ind_df_list = ind_df_multi_feature(alt_fl,␣
↪chosen_fl, feature_col_headers, start_ind_id_count)

                pop_df, drop = save(multi_feature_ind_df_list, filename,␣
↪distance, drop_lst, feature_col_headers, start_ind_id_count,␣
↪save_individual_level_data=False, save_poplation_level_data=False)

                if len(drop)==len(chosen_rul):
                    continue

                diff_radius.append(pop_df)

                start_ind_id_count = start_ind_id_count +␣
↪data_list[data][observation].shape[0]

            if len(diff_radius) == 0:
                continue
            all_observations_one_radius = pd.concat(diff_radius)
```

```
            all_observations_one_radius.to_csv(folder_path + filenames[data] +␣
↪str(distance) + 'm_step_size_' + str(radius) + 'm_radius_size.csv',␣
↪index=False, header=True)

    print('dataset in data_list[' + str(data) + '] processed')
```

```
<ipython-input-21-c2915904d6eb>:48: RuntimeWarning: invalid value encountered in
less_equal
  loc = np.where((loc<=r**2), 1, 0)
<ipython-input-23-81f3d7d227e0>:67: RuntimeWarning: invalid value encountered in
less_equal
  loc = np.where((loc<=r**2), 1, 0)
<ipython-input-23-81f3d7d227e0>:106: RuntimeWarning: invalid value encountered
in less_equal
  loc = np.where((loc<=r**2), 1, 0)
```

```
dataset in data_list[0] processed
dataset in data_list[1] processed
dataset in data_list[2] processed
dataset in data_list[3] processed
dataset in data_list[4] processed
dataset in data_list[5] processed
dataset in data_list[6] processed
dataset in data_list[7] processed
dataset in data_list[8] processed
```

[696]:
```
ncols=2
nrows=1

fig, ax = plt.subplots(nrows, ncols, sharex=False, sharey=True,␣
 ↪constrained_layout=True)
# fig.delaxes(ax[1,2])
ax[0].plot(t, np.sin(2 * np.pi * t))
# ax[0,0].hist(np.arange(5))
# ax[0,0].plot(t, np.sin(0.5 * np.pi * t))
# ax[0,1].plot(t, np.sin(0.5 * np.pi * t))
plt.close()
# ax[1,0].plot(t, np.cos(2 * np.pi * t))
# plt.setp(ax2.get_xticklabels(), visible=False)
# fig.savefig('test')
```

[ ]:
```

```

[337]:
```
import matplotlib.gridspec as gridspec
```

[1]:
```
filenames = ['df_spatial_model', 'before_scare_df_spatial_model',␣
 ↪'after_scare_df_spatial_model',
```

```
            'grevy_df_spatial_model', 'grevy_before_scare_df_spatial_model',␣
 ↪'grevy_after_scare_df_spatial_model',
           'plains_df_spatial_model', 'plains_before_scare_df_spatial_model',␣
 ↪'plains_after_scare_df_spatial_model']

feature_key = ['gc']

figure_names = ['both species', 'both species before scare', 'both species␣
 ↪after scare',
              'Grevy\'s', 'Grevy\'s' + ' before scare', 'Grevy\'s' + ' after␣
 ↪scare',
              'plains', 'plains before scare', 'plains after scare']
ncols=3
nrows=3

step_size = [5,10]
radius_around_ind = [1, 5, 10, 15, 20]

for feature in range(len(feature_key)):
    fig, ax = plt.subplots(nrows, ncols, sharex=False, sharey=True,␣
 ↪constrained_layout=False, figsize=(15, 10))
    fig.delaxes(ax[2,2])
    fig_row_index = 0

    for name in range(len(filenames)):
        if filenames[name]=='plains_after_scare_df_spatial_model':
            continue
        obsv_df = multi_observations_beta_df(step_size, radius_around_ind, ('/
 ↪Users/Taylor/Desktop/' + filenames[name]))
        group = obsv_df.groupby('step_size')
        fig_col_index = name

        if fig_col_index == ncols:
            fig_row_index = fig_row_index + 1

        elif fig_col_index == 2*ncols:
            fig_row_index = fig_row_index + 1

        if (fig_col_index >= ncols) & (fig_col_index <= (ncols*2)-1):
            fig_col_index = fig_col_index - ncols

        elif fig_col_index >= ncols*2:
            fig_col_index = fig_col_index - ncols*2

        for i in range(len(step_size)):
            if step_size[i] == 5:
```

```
                symbol = 'o'
            else:
                symbol = 'D'
            grp = group.get_group(step_size[i])

            if feature_key[feature]=='all_features':

                ax[fig_row_index, fig_col_index].
↪scatter(grp['radius_size_around_ind'], grp['group_center'],␣
↪label=str(step_size[i])+'m gc', marker=symbol, c='blue')
                ax[fig_row_index, fig_col_index].
↪scatter(grp['radius_size_around_ind'], grp['social_density'],␣
↪label=str(step_size[i])+'m sd', marker=symbol, c="red")
                ax[fig_row_index, fig_col_index].
↪scatter(grp['radius_size_around_ind'], grp['recently_used_space'],␣
↪label=str(step_size[i])+'m ru', marker=symbol, c='green')

            elif feature_key[feature]=='sd':
                ax[fig_row_index, fig_col_index].
↪scatter(grp['radius_size_around_ind'], grp['social_density'],␣
↪label=str(step_size[i])+'m', marker=symbol)
            elif feature_key[feature]=='ru':
                ax[fig_row_index, fig_col_index].
↪scatter(grp['radius_size_around_ind'], grp['recently_used_space'],␣
↪label=str(step_size[i])+'m', marker=symbol)
            else:
                if step_size[i] == 5:
                    continue
                else:
                    symbol = 'o'
                ax[fig_row_index, fig_col_index].
↪scatter(grp['radius_size_around_ind'].iloc[1:3], grp['group_center'].iloc[1:
↪3], marker=symbol)
                ax[fig_row_index, fig_col_index].
↪plot(grp['radius_size_around_ind'].iloc[1:3], grp['group_center'].iloc[1:3])

#             ax[fig_row_index, fig_col_index].legend(loc='upper right',␣
↪prop={'size': 8})
            ax[fig_row_index, fig_col_index].title.set_text(figure_names[name])

    handles, labels = ax[fig_row_index, fig_col_index].
↪get_legend_handles_labels()
    fig.legend(handles, labels)

    if feature_key[feature]=='gc':
        fig.text(0.47, 0, 'step size (meters)', ha='center', fontsize=12)
```

```
    else:
        fig.text(0.47, 0, 'radius size around indivdual (meters)', ha='center',␣
↪fontsize=12)

    fig.tight_layout(rect=[0,0,0.9,1])
#     fig.tight_layout()

    fig.savefig('/Users/Taylor/Documents/model_figures/' + feature_key[feature]␣
↪+ '_one_legend')
#     fig.savefig('/Users/Taylor/Documents/model_figures/' +␣
↪feature_key[feature] + '_multiple_legends')
```

[566]:
```
# obsv_df[['step_size', 'social_density']]
obsv_df
```

[566]:

	step_size	group_center	radius_size_around_ind
0	5.0	-0.083890	1.0
1	5.0	-0.074834	5.0
2	5.0	-0.063264	10.0
3	5.0	-0.076045	15.0
4	5.0	-0.085219	20.0
5	10.0	-0.190719	1.0
6	10.0	-0.079717	5.0
7	10.0	-0.076346	10.0
8	10.0	-0.079362	15.0
9	10.0	-0.089988	20.0

0.0.2 Probability

[51]:
```
from collections import Counter
```

[1141]:
```
def calculate_probability(step_size, resolution, list_of_observations,␣
↪radius_around_ind=None,
                          look_back=None, fraction=True,␣
↪group_center_feature=False, social_density=False,
                          recently_used_space=False, temporal_data=False,␣
↪double_count=False,
                         trails=False, list_of_observation_names=None,␣
↪observation_maps_dict=None):
    """
    Inputs:
        step_size: list of ints, how far animal has travelled each step
        resolution: int, at every 'blank' frame, data will be pulled
        list_of_observations: list of 3D numpy arrays containing tracks for␣
↪each individual in each observation
        radius_around_ind: list of ints; radius around focal individual
```

```
        look_back: int. How many frames to look back through (30 frames = 1sec)
        trails: True if calculate trail probabilities
        list_of_observation_names: name of each observation in␣
↪list_of_observations
        observation_maps_dict: if calculcating trail probabilities: has␣
↪trail_masks and info

    Output:
        probabilities: list of probabilities associated with
        keys:
        event_counts:
    """

    #chosen_l: list of 1D arrays; value for every step/individual;␣
↪len(chosen_l)=number of individuals; chosen_l[0] corresponds to ind0
    #alt_l: list of 2D arrays; values for each alternative step for each ind;␣
↪len(alt_l)=num of individuals; alt_l[0].shape= (5,#steps)


    probabilities = []
    keys = []
    event_counts = []

    if group_center_feature==True:

        for distance in step_size:
            all_observations_one_step_size = []
            for obsv in list_of_observations:
                new_coords_l, spatial, _, _ = spatial_dis(obsv, distance,␣
↪resolution)
                alt_steps = random_steps(spatial, distance)
                cl = group_center(new_coords_l)

                chosen_l, alt_l = dc_feature(cl, spatial, distance)

                diff_l = []
                for ind in range(len(chosen_l)):
                    step = np.random.randint(0, len(alt_l[ind]))
                    diff = chosen_l[ind] - alt_l[ind][step]
                    diff_l.append(diff)
                diff_array = np.concatenate(diff_l)
                diff_array = np.around(diff_array, decimals=1)

                all_observations_one_step_size.append(diff_array)
            all_observations = np.concatenate(all_observations_one_step_size)

            outcome_count = Counter(all_observations)
```

```
            event_count = Counter(np.abs(all_observations))

            probabilities_list = []
            keys_list = []
            for key, value in outcome_count.items():
                if key == 0:
                    probability = 0.5
                else:
                    probability = value/event_count[np.abs(key)]
                probabilities_list.append(probability)
                keys_list.append(key)

            probabilities.append(np.asarray(probabilities_list))
            keys.append(np.asarray(keys_list))
            event_counts.append(all_observations)

    elif trails:
        for distance in step_size:
            diff_all_obsv_one_step = [] #all observations for a single radius␣
↪and step size
            for obsv, obs_name in zip(list_of_observations,␣
↪list_of_observation_names):
                new_coords_l, spatial, time_index, _ = spatial_dis(obsv,␣
↪distance, resolution)
                alt_steps = random_steps(spatial, distance)


                trail_mask = observation_maps_dict[obs_name]['trail_mask']
                mask_info_df = observation_maps_dict[obs_name]['mask_info_df']
                chosen_l, alt_l = trail_feature(alt_steps, spatial, trail_mask,␣
↪mask_info_df)



                diff_l = []
                for ind in range(len(chosen_l)):
                    step = np.random.randint(0, len(alt_l[ind]))
                    diff = chosen_l[ind] - alt_l[ind][step]
                    diff_l.append(diff)
                diff_array = np.concatenate(diff_l)
                diff_all_obsv_one_step.append(diff_array)
            if not diff_all_obsv_one_step:
                probabilities.append(None)
                keys.append(None)
                event_counts.append(None)
                continue
```

```
            diff_all_obsv_one_step_concatenated = np.
↪concatenate(diff_all_obsv_one_step)

            outcome_count = Counter(diff_all_obsv_one_step_concatenated)
            event_count = Counter(np.abs(diff_all_obsv_one_step_concatenated))

#             print(outcome_count)

            probabilities_list = []
            keys_list = []
            for key, value in outcome_count.items():
                if key == 0:
                    probability = 0.5
                elif event_count[np.abs(key)] == 0:
                    probability = 0
                else:
                    probability = value/event_count[np.abs(key)]
                probabilities_list.append(probability)
                keys_list.append(key)

            probabilities.append(np.asarray(probabilities_list))
            keys.append(np.asarray(keys_list))
            event_counts.append(diff_all_obsv_one_step_concatenated)


    else:
        for distance in step_size:
            for radius in radius_around_ind:
                diff_all_obsv_one_radius = [] #all observations for a single␣
↪radius and step size
                for obsv in list_of_observations:
                    new_coords_l, spatial, time_index, _ = spatial_dis(obsv,␣
↪distance, resolution)
                    alt_steps = random_steps(spatial, distance)

                    if social_density==True:
                        chosen_l, alt_l = sd_feature(alt_steps, new_coords_l,␣
↪radius, fraction)

                    elif recently_used_space==True:
                        chosen_l, alt_l = ru_feature(alt_steps, obsv, radius,␣
↪time_index, look_back, temporal_data, double_count, fraction)

                    diff_l = []
                    for ind in range(len(chosen_l)):
                        step = np.random.randint(0, len(alt_l[ind]))
                        diff = chosen_l[ind] - alt_l[ind][step]
```

```
                        diff_l.append(diff)
                    diff_array = np.concatenate(diff_l)

                    if fraction==True:
                        diff_array = np.around(diff_array, decimals=1)
                    diff_all_obsv_one_radius.append(diff_array)
                diff_all_obsv_one_radius_concatenated = np.
↪concatenate(diff_all_obsv_one_radius)

                outcome_count = Counter(diff_all_obsv_one_radius_concatenated)
                event_count = Counter(np.
↪abs(diff_all_obsv_one_radius_concatenated))

                probabilities_list = []
                keys_list = []
                for key, value in outcome_count.items():
                    if key == 0:
                        probability = 0.5
                    else:
                        probability = value/event_count[np.abs(key)]
                    probabilities_list.append(probability)
                    keys_list.append(key)

                probabilities.append(np.asarray(probabilities_list))
                keys.append(np.asarray(keys_list))
                event_counts.append(diff_all_obsv_one_radius_concatenated)

    #add in missing zero values for when probability is 1
    indexes = []
    for radius in range(len(probabilities)):
        indexes.append(np.argwhere(probabilities[radius]==1))

    for indx in range(len(indexes)):
        for diff in range(indexes[indx].shape[0]):
            missing_zero_values_figure = keys[indx][indexes[indx][diff,0]] * -1

            keys[indx] = np.append(keys[indx], missing_zero_values_figure)
            probabilities[indx] = np.append(probabilities[indx], 0)

    return keys, probabilities, event_counts
```

[2]:
```
map_data_names
```

[1142]:
```
data_list = [positions, before_scare_positions, after_scare_positions,
             grevy_positions, grevy_before_scare_positions,␣
↪grevy_after_scare_positions,
```

```
        plains_positions, plains_before_scare_positions,␣
↪plains_after_scare_positions
        ]

data_list = [positions]


filenames = ['both_species', 'both_species_before_scare',␣
↪'both_species_after_scare',
             'grevy', 'grevy_before_scare', 'grevy_after_scare',
             'plains', 'plains_before_scare', 'plains_after_scare',
             ]

folder_paths = ['/Users/Taylor/Documents/recently_used_space/', '/Users/Taylor/
↪Documents/social_density/',
                '/Users/Taylor/Documents/group_center/', '/Users/Taylor/
↪Documents/trail_use/']


step_size = [5, 10]
resolution = 1
radius_around_ind = [1, 5, 10, 15, 20]
look_back = 3600
fraction = False

for data in range(3,6):
    for trial in range(50):
        ru_keys, ru_probabilities, ru_occurances =␣
↪calculate_probability(step_size, resolution, data_list[data],
                                                                    ␣
↪radius_around_ind, look_back, fraction=fraction,
                                                                    ␣
↪recently_used_space=True)
        sd_keys, sd_probabilities, sd_occurances =␣
↪calculate_probability(step_size, resolution, data_list[data],
                                                                    ␣
↪radius_around_ind, fraction=fraction, social_density=True)


        trail_keys, trail_probabilities, trail_occurances =␣
↪calculate_probability(step_size, resolution, map_data_list[data],
                                                                    ␣
↪radius_around_ind, trails=True,
                                                                    ␣
↪  list_of_observation_names=map_data_names[data],
```

```
                                                                                ␣
↪  observation_maps_dict=observation_maps_dict)

          gc_keys, gc_probabilities, gc_occurances =␣
↪calculate_probability(step_size, resolution, data_list[data],␣
↪group_center_feature=True)

      for size in range(len(step_size)):
          np.save(folder_paths[2] + filenames[data] + '_gc_keys_' +␣
↪str(step_size[size]) + 'm_step_size', gc_keys[size])
          np.save(folder_paths[2] + filenames[data] + '_gc_probabilities_' +␣
↪str(step_size[size]) + 'm_step_size', gc_probabilities[size])
          np.save(folder_paths[2] + filenames[data] + '_gc_occurances_' +␣
↪str(step_size[size]) + 'm_step_size', gc_occurances[size])

      for radius_len in range(len(radius_around_ind)*len(step_size)):
          radius_name = radius_len
          if radius_len > 4:
              radius_name = radius_len - len(radius_around_ind)
              size = 10
          else:
              size = 5
          if fraction == False:
              np.save(folder_paths[0] + filenames[data] + '_ru_keys_' +␣
↪str(size) + 'm_step_size_' + str(radius_around_ind[radius_name])
                      + 'm_radius_size'  + '_whole_counts',␣
↪ru_keys[radius_len])
              np.save(folder_paths[0] + filenames[data] +␣
↪'_ru_probabilities_' + str(size) + 'm_step_size_' +␣
↪str(radius_around_ind[radius_name])
                      + 'm_radius_size' + '_whole_counts',␣
↪ru_probabilities[radius_len])
              np.save(folder_paths[0] + filenames[data] + '_ru_occurances_' +␣
↪str(size) + 'm_step_size_' + str(radius_around_ind[radius_name])
                      + 'm_radius_size' + '_whole_counts',␣
↪ru_occurances[radius_len])

              np.save(folder_paths[1] + filenames[data] + '_sd_keys_' +␣
↪str(size) + 'm_step_size_' + str(radius_around_ind[radius_name])
                      + 'm_radius_size' + '_whole_counts',␣
↪sd_keys[radius_len])
              np.save(folder_paths[1] + filenames[data] +␣
↪'_sd_probabilities_' + str(size) + 'm_step_size_' +␣
↪str(radius_around_ind[radius_name])
                      + 'm_radius_size' + '_whole_counts',␣
↪sd_probabilities[radius_len])
```

```
                np.save(folder_paths[1] + filenames[data] + '_sd_occurances_' +␣
↪str(size) + 'm_step_size_' + str(radius_around_ind[radius_name])
                        + 'm_radius_size' + '_whole_counts',␣
↪sd_occurances[radius_len])
            else:
                np.save(folder_paths[0] + filenames[data] + '_ru_keys_' +␣
↪str(size) + 'm_step_size_' + str(radius_around_ind[radius_name])
                        + 'm_radius_size', ru_keys[radius_len])
                np.save(folder_paths[0] + filenames[data] +␣
↪'_ru_probabilities_' + str(size) + 'm_step_size_' +␣
↪str(radius_around_ind[radius_name])
                        + 'm_radius_size', ru_probabilities[radius_len])
                np.save(folder_paths[0] + filenames[data] + '_ru_occurances_' +␣
↪str(size) + 'm_step_size_' + str(radius_around_ind[radius_name])
                        + 'm_radius_size', ru_occurances[radius_len])

                np.save(folder_paths[1] + filenames[data] + '_sd_keys_' +␣
↪str(size) + 'm_step_size_' + str(radius_around_ind[radius_name])
                        + 'm_radius_size', sd_keys[radius_len])
                np.save(folder_paths[1] + filenames[data] +␣
↪'_sd_probabilities_' + str(size) + 'm_step_size_' +␣
↪str(radius_around_ind[radius_name])
                        + 'm_radius_size', sd_probabilities[radius_len])
                np.save(folder_paths[1] + filenames[data] + '_sd_occurances_' +␣
↪str(size) + 'm_step_size_' + str(radius_around_ind[radius_name])
                        + 'm_radius_size', sd_occurances[radius_len])
        for step_num in range(len(step_size)):
            np.save(folder_paths[3] + filenames[data] + '_trail_keys_' +␣
↪str(step_size[step_num])
                    + 'm_step_size_' + str(0) + 'm_radius_size' + str(trial) +␣
↪'_trial', trail_keys[step_num])
            np.save(folder_paths[3] + filenames[data] + '_trail_probabilities_'␣
↪+ str(step_size[step_num])
                    + 'm_step_size_' + str(0) + 'm_radius_size' + str(trial) +␣
↪'_trial', trail_probabilities[step_num])
            np.save(folder_paths[3] + filenames[data] + '_trail_occurances_' +␣
↪str(step_size[step_num])
                    + 'm_step_size_' + str(0) + 'm_radius_size' + str(trial) +␣
↪'_trial', trail_occurances[step_num])
```

[1074]:
```
trail_keys
```

[1074]:
```
[None, None]
```

[3]:
```
figure_names = ['both species', 'both species before scare', 'both species␣
↪after scare',
```

```
                'Grevy\'s', 'Grevy\'s' + ' before scare', 'Grevy\'s' + ' after␣
↪scare',
                'plains', 'plains before scare', 'plains after scare']
# figure_names = ['both species before scare']

folder_paths = ['/Users/Taylor/Documents/recently_used_space/',
                '/Users/Taylor/Documents/social_density/',

               ]
folder_paths = ['/Users/Taylor/Documents/group_center/']


file_types = ['both_species', 'both_species_before_scare',␣
↪'both_species_after_scare',
              'grevy', 'grevy_before_scare', 'grevy_after_scare',
              'plains', 'plains_before_scare', 'plains_after_scare']
# file_types = ['both_species']

fraction = True
radius_around_ind_name = [1, 5, 10, 15, 20, 1, 5, 10, 15, 20]
step_size_l = [5, 10]

for fraction in [True, False]:

    for folder in folder_paths:

        if 'social_density' in folder:
            feature_key = 'sd'
        if 'recently_used' in folder:
            feature_key = 'ru'
        if 'group_center' in folder:
            feature_key = 'gc'

        for file_type in file_types:
            if fraction==False:
                keys_files = glob.glob(
                    os.path.join(folder, file_type + '_' + feature_key +␣
↪'_keys*' + '_whole_counts.npy'))
                keys_files.sort(key=lambda f: (int(f.split('m_step_size')[0].
↪split('_')[-1]),
                                               int(f.split('m_radius_size')[0].
↪split('_')[-1])))
                keys = [np.load(file) for file in keys_files]

                probabilities_files = glob.glob(
                    os.path.join(folder, file_type + '_' + feature_key +␣
↪'_probabilities*'+ '_whole_counts.npy'))
```

```
                probabilities_files.sort(key=lambda f: (int(f.
↪split('m_step_size')[0].split('_')[-1]),
                                                        int(f.
↪split('m_radius_size')[0].split('_')[-1])))
                probabilities = [np.load(file) for file in probabilities_files]

                occurances_files = glob.glob(
                    os.path.join(folder, file_type + '_' + feature_key +␣
↪'_occurances*' + '_whole_counts.npy'))
                occurances_files.sort(key=lambda f: (int(f.
↪split('m_step_size')[0].split('_')[-1]),
                                                     int(f.
↪split('m_radius_size')[0].split('_')[-1])))
                occurances = [np.load(file) for file in occurances_files]

            else:
                keys_files = glob.glob(os.path.join(folder, file_type + '_' +␣
↪feature_key + '_keys*radius_size.npy'))
                keys_files.sort(key=lambda f: (int(f.split('m_step_size')[0].
↪split('_')[-1]),
                                               int(f.split('m_radius_size')[0].
↪split('_')[-1])))
                keys = [np.load(file) for file in keys_files]

                probabilities_files = glob.glob(os.path.join(folder, file_type␣
↪+ '_' + feature_key + '_probabilities*radius_size.npy'))
                probabilities_files.sort(key=lambda f: (int(f.
↪split('m_step_size')[0].split('_')[-1]),
                                                        int(f.
↪split('m_radius_size')[0].split('_')[-1])))
                probabilities = [np.load(file) for file in probabilities_files]

                occurances_files = glob.glob(os.path.join(folder, file_type +␣
↪'_' + feature_key + '_occurances*radius_size.npy'))
                occurances_files.sort(key=lambda f: (int(f.
↪split('m_step_size')[0].split('_')[-1]),
                                                     int(f.
↪split('m_radius_size')[0].split('_')[-1])))
                occurances = [np.load(file) for file in occurances_files]


            if feature_key=='gc':

                keys_files = glob.glob(os.path.join(folder, file_type + '_' +␣
↪feature_key + '_keys*.npy'))
```

```
                keys_files.sort(key=lambda f: (int(f.split('m_step_size')[0].
↪split('_')[-1])))
                keys = [np.load(file) for file in keys_files]

                probabilities_files = glob.glob(os.path.join(folder, file_type␣
↪+ '_' + feature_key + '_probabilities*.npy'))
                probabilities_files.sort(key=lambda f: (int(f.
↪split('m_step_size')[0].split('_')[-1])))
                probabilities = [np.load(file) for file in probabilities_files]

                occurances_files = glob.glob(os.path.join(folder, file_type +␣
↪'_' + feature_key + '_occurances*.npy'))
                occurances_files.sort(key=lambda f: (int(f.
↪split('m_step_size')[0].split('_')[-1])))
                occurances = [np.load(file) for file in occurances_files]


                nrows = 1
                ncols = 2
                fig, ax = plt.subplots(nrows, ncols, sharex=False, sharey=True,␣
↪constrained_layout=False, figsize=(10, 3.5))

                for size in range(len(keys)):
                    if size == 0:
                        symbol = 'o'
                    else:
                        symbol = 'D'

                    fig_col_index = size

                    ax[fig_col_index].scatter(keys[size], probabilities[size],␣
↪marker=symbol, label=str(step_size_l[size]) + 'm')
                    ax[fig_col_index].legend()
                    ax[fig_col_index].set_xlabel('chosen location - alternative␣
↪location')
                    ax[fig_col_index].set_ylabel('probability chosen location')
                    ax[fig_col_index].set_title(file_type.replace('_', ' ') + '␣
↪' + feature_key)

                    ax2 = fig.add_subplot(ax[fig_col_index])
                    ax2 = ax[fig_col_index].twinx()  # instantiate a second␣
↪axes that shares the same x-axis

                    bin_values = np.array(sorted(keys[size])) - 0.5
                    ax2.hist(occurances[size], alpha=0.3, bins=bin_values)
```

```
                    fig1.tight_layout()

                fig.savefig('/Users/Taylor/Dropbox/Thesis_materials/
↪probability_figures/' + file_type + '_' + feature_key)
#                   plt.close()
            else:
                ncols=5
                nrows=2

                fig, ax = plt.subplots(nrows, ncols, sharex=False, sharey=True,␣
↪constrained_layout=False, figsize=(20, 5))
                fig_row_index = 0

                for radius_len in range(len(keys)):
                    fig_col_index = radius_len
                    if radius_len <= 4:
                        symbol = 'o'
                        step_size = 5
                    else:
                        symbol = 'D'
                        step_size = 10

                    if fig_col_index == ncols:
                        fig_row_index = fig_row_index + 1

                    if fig_col_index >= ncols:
                        fig_col_index = fig_col_index - ncols

                    label = str(step_size) + 'm, ' +␣
↪str(radius_around_ind_name[radius_len]) + 'm radius'
                    ax[fig_row_index, fig_col_index].scatter(keys[radius_len],␣
↪probabilities[radius_len],
                                                             marker=symbol,␣
↪label=label)
                    ax[fig_row_index, fig_col_index].legend()
                    ax[fig_row_index, fig_col_index].set_xlabel('chosen␣
↪location - alternative location')
                    ax[fig_row_index, fig_col_index].set_ylabel('probability␣
↪chosen location')
                    ax[fig_row_index, fig_col_index].set_title(file_type.
↪replace('_', ' ') + ' ' + feature_key)

                    ax2 = fig.add_subplot(ax[fig_row_index, fig_col_index])
                    ax2 = ax[fig_row_index, fig_col_index].twinx()  #␣
↪instantiate a second axes that shares the same x-axis
```

```
                    sorted_keys = np.array(sorted(keys[radius_len]))
                    if len(sorted_keys) > 1:
                        bin_values = sorted_keys - (sorted_keys[1] -␣
↪sorted_keys[0]) / 2
                    else:
                        bin_values = 1
                    ax2.hist(occurances[radius_len], alpha=0.3, bins=bin_values)

                    fig.tight_layout()
                if feature_key=='sd' or feature_key=='ru':
                    if fraction == True:
                        fig.savefig('/Users/Taylor/Dropbox/Thesis_materials/
↪probability_figures/'
                                    + file_type + '_' + feature_key +␣
↪'_fraction')
                    if fraction == False:
                        fig.savefig('/Users/Taylor/Dropbox/Thesis_materials/
↪probability_figures/' + file_type + '_' + feature_key)

                plt.close()
```

[4]:
```
figure_names = [
               'Grevy\'s', 'Grevy\'s' + ' before scare', 'Grevy\'s' + ' after␣
↪scare',
               ]

folder = '/Users/Taylor/Documents/trail_use/'

file_types = [
              'grevy', 'grevy_before_scare', 'grevy_after_scare']
# file_types = ['both_species']

step_size_l = [5, 10]


feature_key = 'trail'

num_trials = 50

for file_type in file_types[:]:


    ncols=2
    nrows=1

    fig, ax = plt.subplots(nrows, ncols, sharex=False, sharey=True,␣
↪constrained_layout=False, figsize=(15, 5))
```

```
    fig_row_index = 0

    prob_sums = [np.zeros(len(file_types)) for _ in range(len(step_size_l))]


    for trial in range(num_trials):

        keys_files = glob.glob(os.path.join(folder, file_type + '_' +␣
↪feature_key + '_keys*radius_size'+str(trial)+'*.npy'))
        keys_files.sort(key=lambda f: (int(f.split('m_step_size')[0].
↪split('_')[-1]),
                                       int(f.split('m_radius_size')[0].
↪split('_')[-1])))
        keys = [np.load(file, allow_pickle=True) for file in keys_files]

        probabilities_files = glob.glob(os.path.join(folder, file_type + '_' +␣
↪feature_key + '_probabilities*radius_size'+str(trial)+'*.npy'))
        probabilities_files.sort(key=lambda f: (int(f.split('m_step_size')[0].
↪split('_')[-1]),
                                                int(f.split('m_radius_size')[0].
↪split('_')[-1])))
        probabilities = [np.load(file, allow_pickle=True) for file in␣
↪probabilities_files]

        occurances_files = glob.glob(os.path.join(folder, file_type + '_' +␣
↪feature_key + '_occurances*radius_size'+str(trial)+'*.npy'))
        occurances_files.sort(key=lambda f: (int(f.split('m_step_size')[0].
↪split('_')[-1]),
                                             int(f.split('m_radius_size')[0].
↪split('_')[-1])))
        occurances = [np.load(file, allow_pickle=True) for file in␣
↪occurances_files]


        if np.any(occurances[0])==None:
            continue

        for step_ind, step_size in enumerate(step_size_l):


            if step_size == 5:
                symbol = 'o'
            elif step_size == 10:
                symbol = 'D'

            key_inds = keys[step_ind].argsort()
```

```
            keys_s = keys[step_ind][key_inds]
            prob_s = probabilities[step_ind][key_inds]
            prob_sums[step_ind] += prob_s


            fig_col_index = 0


            label = str(step_size) + 'm, ' + str(0) + 'm radius'

            ax[step_ind].scatter(keys_s, prob_s,
                                 marker=symbol, c='blue', alpha=.2)
#             ax[fig_row_index].legend()

    for step_ind, step_size in enumerate(step_size_l):
        ax[step_ind].set_xlabel('chosen location - alternative location')
        ax[step_ind].set_ylabel('probability chosen location')
        ax[step_ind].set_title(file_type.replace('_', ' ') + ' ' + feature_key␣
↪+ ' following probability'
                               + ' {} meter step size'.format(step_size))

        ax[step_ind].scatter(keys_s, prob_sums[step_ind]/num_trials,
                                 marker=symbol, label='{} meter step size'.
↪format(step_size), c='red', alpha=1.0)

        print(prob_sums[step_ind]/num_trials)

        ax[step_ind].set_ylim(0, 1.0)

        ax2 = fig.add_subplot(ax[fig_row_index])
        ax2 = ax[step_ind].twinx()  # instantiate a second axes that shares the␣
↪same x-axis

        sorted_keys = sorted(keys[radius_len])
        sorted_keys.append(sorted_keys[-1] + 1)
        sorted_keys = np.array(sorted_keys)
        if len(sorted_keys) > 1:
            bin_values = sorted_keys - (sorted_keys[1] - sorted_keys[0]) / 2

        else:
            bin_values = 1
        ax2.hist(occurances[step_ind], alpha=0.3, bins=bin_values)

    fig.tight_layout()

    fig.savefig('/Users/Taylor/Dropbox/Thesis_materials/probability_figures/' +␣
↪file_type + '_' + feature_key)
```

```
    plt.close()
```

[1124]:
```
type(keys[step_ind], prob_sums[step_ind]/num_trials
```

[1124]:
```
(array([ 0., -1.,  1.]), array([0.5      , 0.6028123, 0.3971877]))
```

[1132]:
```
key_inds = keys[0].argsort()
keys_s = keys[0][key_inds]
prob_s = probabilities[0][key_inds]

print(keys_s, keys[0])
print(prob_s, probabilities[0])
```

```
[-1.  0.  1.] [ 0.  1. -1.]
[0.31077694 0.5        0.68922306] [0.5        0.68922306 0.31077694]
```

[1079]:
```
keys
```

[1079]:
```
[array([ 0., -1.,  1.]), array([ 0.,  1., -1.])]
```

[181]:
```
step_size = [5, 10]
resolution = 1

gc_keys, gc_probabilities, gc_occurances = calculate_probability(step_size,␣
↪resolution, positions, fraction=True, group_center_feature=True)
```

[607]:
```
print(len(keys))
print(len(probabilities))
print(len(occurances))
```

```
2
2
2
```

[5]:
```
step_size = [5, 10]

for size in range(len(step_size)):
    if size == 0:
        symbol = 'o'
    else:
        symbol = 'D'

    fig, ax1 = plt.subplots()

    ax1.scatter(keys[size], probabilities[size], marker=symbol,␣
↪label=str(step_size[size]) + 'm')
```

```
    ax1.legend()
    ax1.set_title('group center')

    ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

    bin_values = np.array(sorted(gc_keys[size])) - 0.5

    ax2.hist(gc_occurances[size], alpha=0.3, bins=bin_values)

    ax1.set_xlabel('chosen location - alternative location')
    ax1.set_ylabel('probability chosen location')
    fig.tight_layout()  # otherwise the right y-label is slightly clipped
    plt.show()
```

Ari eLife 2017 figures

[ ]:
```
frame = 2500
resolution = 15
buffer = 20
individual = 0
r = 1.5
radius_size_list = [1, 5, 10, 15, 20, 1, 5, 10, 15, 20]

fraction = cal_fraction_figure(positions[0], positions[0][individual,frame, :],␣
 ↪resolution, buffer, frame, individual, r)

for radius_size in range(obsv_df.shape[0]):
    preference = obsv_df['social_density'].iloc[radius_size] * fraction
    plt.imshow(preference)
    plt.colorbar()
    if radius_size<=4:
        step_size=5
    else:
        step_size=10

    plt.title(str(step_size) + 'm_step_size_' +␣
 ↪str(radius_size_list[radius_size]) + 'm_radius_around_ind')
    plt.show()
```

mclogit

July 16, 2020

[48]:
```
install.packages("mclogit")
```

```
The downloaded binary packages are in
/var/folders/8x/2f9v1tz55p13v8q7zt0842tc0000gp/T//RtmpzJSmbD/downloaded_packages
```

[49]:
```
library(mclogit)
```

```
Loading required package: Matrix
```

link to using regular expressions to load files https://stackoverflow.com/questions/47189714/reading-multiple-csv-files-from-a-folder-with-r-using-regex

0.0.1 Workflow test on dummy data

- Read in multiple files with different timescales
- Perform mclogit
- Create csv file with betas in the format below

1 - Load data from python

[56]:
```
load <- function(time) {
    my_datal = list()
    for (t in seq_along(time)){
        my_files = list.files(pattern = paste('mfdf.*?t', toString(time[[t]]),␣
 ↪'.csv', sep=''))
        my_filesl = lapply(my_files, read.csv)
        my_datal[[t]] = my_filesl
    }
    return(my_datal)
}
```

[58]:
```
time = list(4)
my_datal = load(time)
```

[59]:
```
my_files
```

1. 'mfdf_test_ind0df_t4.csv' 2. 'mfdf_test_ind1df_t4.csv' 3. 'mfdf_test_ind2df_t4.csv' 4. 'mfdf_test_ind3df_t4.csv' 5. 'mfdf_test_pop_t4.csv'

2 - Perform conditional logistic regression

[118]:
```
multiple_times = list()
output_dfs = list()

for (t in seq_along(my_datal)) {
    mclogit_output = list()
    for (i in seq_along(my_datal[[t]])) {
        clogit = coef(mclogit(cbind(my_datal[[t]][[i]]$label,␣
 ↪my_datal[[t]][[i]]$frame) ~ feature0 + feature1, data=my_datal[[t]][[i]]))
        mclogit_output[[i]] = as.data.frame(clogit)
    }
    multiple_times[[t]] = mclogit_output
    t_df = cbind(multiple_times[[t]])
    output_dfs[[t]] = t_df
}
```

3 - Create csv file

[48]:
```
output_df = do.call(rbind, output_dfs)
df = data.frame(matrix(unlist(output_df), nrow=length(output_df), byrow=T),␣
 ↪stringsAsFactors=FALSE)
write.table(df, file='/Users/taylorcarter/Desktop/mfdf_test_output.csv', row.
 ↪names=FALSE, col.names=list('beta1', 'beta2'), sep=",")
```

[46]:
```
output_df
```

-0.1683163, -0.2576266
-0.57280550, -0.02581237
-0.71163352, 0.06215971
-1.55722146, 0.01253735
-0.64130944, -0.03153729

0.0.2 Spatial Data

[108]:
```
conditional_logistic_regression <- function(folder, dataset_name, step_size,␣
 ↪col_names, observation, multiple_observations=FALSE) {

        my_datal = list()

        if (multiple_observations==TRUE){
                for (distance in seq_along(step_size)){
                    my_files = list.files(folder, pattern=paste('^',␣
 ↪dataset_name, toString(step_size[[distance]]), "m*", sep=''), full.
 ↪names=TRUE)
```

```
                    my_files = my_files[order(nchar(my_files), my_files)]
                    print(my_files)
                    my_filesl = lapply(my_files, read.csv)
                    my_datal[[distance]] = my_filesl
            }

            mclogit_output_per_step_size = list()
            output_dfs = list()

            for (t in seq_along(my_datal)) {
                mclogit_output = list()
                for (i in seq_along(my_datal[[t]])) {
                    print(t)
                    print(i)

                    if(length(col_names) == 1 & col_names[[1]] ==␣
↪'group_center'){
                        clogit = coef(mclogit(cbind(my_datal[[t]][[i]]$label,␣
↪my_datal[[t]][[i]]$step) ~ group_center,
                                                                    ␣
↪data=my_datal[[t]][[i]]))
                    }
                    else if (length(col_names) == 1 & col_names[[1]] ==␣
↪'social_density'){
                        clogit = coef(mclogit(cbind(my_datal[[t]][[i]]$label,␣
↪my_datal[[t]][[i]]$step) ~ social_density,
                                                                    ␣
↪data=my_datal[[t]][[i]]))
                    }
                    else if (length(col_names) == 1 & col_names[[1]] ==␣
↪'recently_used_space'){
                        clogit = coef(mclogit(cbind(my_datal[[t]][[i]]$label,␣
↪my_datal[[t]][[i]]$step) ~ recently_used_space,
                                                                    ␣
↪data=my_datal[[t]][[i]]))
                    }
                    else if (length(col_names) == 3){
                        clogit = coef(mclogit(cbind(my_datal[[t]][[i]]$label,␣
↪my_datal[[t]][[i]]$step) ~ group_center + social_density +␣
↪recently_used_space,
                                                                    ␣
↪data=my_datal[[t]][[i]]))
                    }
                    else if (length(col_names) == 2 & col_names[[2]] ==␣
↪'social_density'){
```

```
                        clogit = coef(mclogit(cbind(my_datal[[t]][[i]]$label,␣
↪my_datal[[t]][[i]]$step) ~ group_center + social_density,
                                                                    ␣
↪data=my_datal[[t]][[i]]))
                    }
                    else if (length(col_names) == 2 & col_names[[2]] ==␣
↪'recently_used_space'){
                        clogit = coef(mclogit(cbind(my_datal[[t]][[i]]$label,␣
↪my_datal[[t]][[i]]$step) ~ group_center + recently_used_space,
                                                                    ␣
↪data=my_datal[[t]][[i]]))
                    }
                    else if (length(col_names) == 2 & col_names[[1]] ==␣
↪'social_density'){
                        clogit = coef(mclogit(cbind(my_datal[[t]][[i]]$label,␣
↪my_datal[[t]][[i]]$step) ~ social_density + recently_used_space,
                                                                    ␣
↪data=my_datal[[t]][[i]]))
                    }
                    mclogit_output[[i]] = as.data.frame(clogit)
                }
                mclogit_output_per_step_size[[t]] = mclogit_output
                spatial_df = cbind(mclogit_output_per_step_size[[t]])
                output_dfs[[t]] = spatial_df
            }

            output_df = do.call(rbind, output_dfs)
            df = data.frame(matrix(unlist(output_df), nrow=length(output_df),␣
↪byrow=T), stringsAsFactors=FALSE)

            write.table(df, file=paste('/Users/Taylor/Desktop/', dataset_name,␣
↪'spatial_model_output.csv', sep=''), row.names=FALSE, col.names=col_names,␣
↪sep=",")
        }

        else {
            for (obsv in observation){
                for (distance in seq_along(step_size)){
                    my_files = list.files(pattern=paste(filename,␣
↪toString(obsv),'.*?df_', toString(step_size[[distance]]), '.*?.csv', sep=''))
                    my_files = my_files[order(nchar(my_files), my_files)]
                    print(my_files)
                    my_filesl = lapply(my_files, read.csv)
                    my_datal[[distance]] = my_filesl
                }
```

```
                mclogit_output_per_step_size = list()
                output_dfs = list()

                for (t in seq_along(my_datal)) {
                    mclogit_output = list()
                    for (i in seq_along(my_datal[[t]])) {
                        print(t)
                        print(i)

                        if(length(col_names) == 1 & col_names[[1]] ==␣
↪'group_center'){
                            clogit =␣
↪coef(mclogit(cbind(my_datal[[t]][[i]]$label, my_datal[[t]][[i]]$step) ~␣
↪group_center,
                                                                    ␣
↪data=my_datal[[t]][[i]]))
                        }
                        else if (length(col_names) == 1 & col_names[[1]] ==␣
↪'social_density'){
                            clogit =␣
↪coef(mclogit(cbind(my_datal[[t]][[i]]$label, my_datal[[t]][[i]]$step) ~␣
↪social_density,
                                                                    ␣
↪data=my_datal[[t]][[i]]))
                        }
                        else if (length(col_names) == 1 & col_names[[1]] ==␣
↪'recently_used_space'){
                            clogit =␣
↪coef(mclogit(cbind(my_datal[[t]][[i]]$label, my_datal[[t]][[i]]$step) ~␣
↪recently_used_space,
                                                                    ␣
↪data=my_datal[[t]][[i]]))
                        }
                        else if (length(col_names) == 3){
                            clogit =␣
↪coef(mclogit(cbind(my_datal[[t]][[i]]$label, my_datal[[t]][[i]]$step) ~␣
↪group_center + social_density + recently_used_space,
                                                                    ␣
↪data=my_datal[[t]][[i]]))
                        }
                        else if (length(col_names) == 2 & col_names[[2]] ==␣
↪'social_density'){
                            clogit =␣
↪coef(mclogit(cbind(my_datal[[t]][[i]]$label, my_datal[[t]][[i]]$step) ~␣
↪group_center + social_density,
```

```
                                                                    ␣
↪data=my_datal[[t]][[i]]))
                        }
                        else if (length(col_names) == 2 & col_names[[2]] ==␣
↪'recently_used_space'){
                            clogit =␣
↪coef(mclogit(cbind(my_datal[[t]][[i]]$label, my_datal[[t]][[i]]$step) ~␣
↪group_center + recently_used_space,
                                                                    ␣
↪data=my_datal[[t]][[i]]))
                        }
                        else if (length(col_names) == 2 & col_names[[1]] ==␣
↪'social_density'){
                            clogit =␣
↪coef(mclogit(cbind(my_datal[[t]][[i]]$label, my_datal[[t]][[i]]$step) ~␣
↪social_density + recently_used_space,
                                                                    ␣
↪data=my_datal[[t]][[i]]))
                        }
                        mclogit_output[[i]] = as.data.frame(clogit)
                    }
                    mclogit_output_per_step_size[[t]] = mclogit_output
                    spatial_df = cbind(mclogit_output_per_step_size[[t]])
                    output_dfs[[t]] = spatial_df
                }

                output_df = do.call(rbind, output_dfs)
                df = data.frame(matrix(unlist(output_df),␣
↪nrow=length(output_df), byrow=T), stringsAsFactors=FALSE)

                write.table(df, file=paste('/Users/Taylor/Desktop/',␣
↪dataset_name, toString(obsv), '_spatial_model_output.csv', sep=''), row.
↪names=FALSE, col.names=col_names, sep=",")
            }
        }
}
```

[107]:
```
for (distance in seq_along(step_size)){
                    my_files = list.files(folder, pattern=paste('^',␣
↪dataset_name, toString(step_size[[distance]]), "m*", sep=''), full.
↪names=TRUE)
                    my_files = my_files[order(nchar(my_files), my_files)]
                    print(my_files)
                    my_filesl = lapply(my_files, read.csv)
                    my_datal[[distance]] = my_filesl
            }
```

```
[1] "all_features_model/df_5m_step_size_1m_radius_size.csv"
[2] "all_features_model/df_5m_step_size_5m_radius_size.csv"
[3] "all_features_model/df_5m_step_size_10m_radius_size.csv"
[4] "all_features_model/df_5m_step_size_15m_radius_size.csv"
[5] "all_features_model/df_5m_step_size_20m_radius_size.csv"
[1] "all_features_model/df_10m_step_size_1m_radius_size.csv"
[2] "all_features_model/df_10m_step_size_5m_radius_size.csv"
[3] "all_features_model/df_10m_step_size_10m_radius_size.csv"
[4] "all_features_model/df_10m_step_size_15m_radius_size.csv"
[5] "all_features_model/df_10m_step_size_20m_radius_size.csv"
```

0.0.3 Both Species

All Features

[119]:
```
step_size = list(5,10)
col_names = list('group_center', 'social_density', 'recently_used_space')
observation = list(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) #not used but␣
 ↪didn't know how to put the R equivalent of the python None
folder = 'all_features_model'

dataset_name = list('df_', 'before_scare_df_', 'after_scare_df_',
             'plains_df_', 'plains_before_scare_df_',
             'grevy_df_', 'grevy_before_scare_df_', 'grevy_after_scare_df_')

for (name in seq_along(dataset_name)){
    print(dataset_name[[name]])
    conditional_logistic_regression(folder, dataset_name[[name]], step_size,␣
 ↪col_names, observation, multiple_observations=TRUE)
}
```

Recently Used Space

[120]:
```
step_size = list(5,10)
col_names = list('recently_used_space')
observation = list(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) #not used but␣
 ↪didn't know how to put the R equivalent of the python None
folder = 'all_features_model'

dataset_name = list('df_', 'before_scare_df_', 'after_scare_df_',
             'plains_df_', 'plains_before_scare_df_',
             'grevy_df_', 'grevy_before_scare_df_', 'grevy_after_scare_df_')

for (name in seq_along(dataset_name)){
    print(dataset_name[[name]])
    conditional_logistic_regression(folder, dataset_name[[name]], step_size,␣
 ↪col_names, observation, multiple_observations=TRUE)
}
```

Social Density

[121]:
```
step_size = list(5,10)
col_names = list('social_density')
observation = list(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) #not used but␣
 ↪didn't know how to put the R equivalent of the python None
folder = 'all_features_model'

dataset_name = list('df_', 'before_scare_df_', 'after_scare_df_',
             'plains_df_', 'plains_before_scare_df_',
             'grevy_df_', 'grevy_before_scare_df_', 'grevy_after_scare_df_')

for (name in seq_along(dataset_name)){
    print(dataset_name[[name]])
    conditional_logistic_regression(folder, dataset_name[[name]], step_size,␣
 ↪col_names, observation, multiple_observations=TRUE)
}
```

Group Center

[122]:
```
step_size = list(5,10)
col_names = list('group_center')
observation = list(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) #not used but␣
 ↪didn't know how to put the R equivalent of the python None
folder = 'all_features_model'

dataset_name = list('df_', 'before_scare_df_', 'after_scare_df_',
             'plains_df_', 'plains_before_scare_df_',
             'grevy_df_', 'grevy_before_scare_df_', 'grevy_after_scare_df_')

for (name in seq_along(dataset_name)){
    print(dataset_name[[name]])
    conditional_logistic_regression(folder, dataset_name[[name]], step_size,␣
 ↪col_names, observation, multiple_observations=TRUE)
}
```

0.0.4 Grevy

All features

[123]:
```
step_size = list(5,10)
col_names = list('group_center', 'social_density', 'recently_used_space')
observation = list(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) #not used but␣
 ↪didn't know how to put the R equivalent of the python None
filename = 'multi_observations_grevy'

conditional_logistic_regression(filename, step_size, col_names, observation)
```

Social Density

[124]:
```
step_size = list(5,10)
col_names = list('social_density')
observation = list(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) #not used but␣
↪didn't know how to put the R equivalent of the python None
filename = 'multi_observations_grevy'

conditional_logistic_regression(filename, step_size, col_names, observation)
```

Recently Used Space

[125]:
```
step_size = list(5,10)
col_names = list('recently_used_space')
observation = list(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) #not used but␣
↪didn't know how to put the R equivalent of the python None
filename = 'multi_observations_grevy'

conditional_logistic_regression(filename, step_size, col_names, observation)
```

0.0.5 Plains

All features

[126]:
```
step_size = list(5,10)
col_names = list('group_center', 'social_density', 'recently_used_space')
observation = list(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) #not used but␣
↪didn't know how to put the R equivalent of the python None
filename = 'multi_observations_plains'

conditional_logistic_regression(filename, step_size, col_names, observation)
```

Recently Used Space

[127]:
```
step_size = list(5,10)
col_names = list('recently_used_space')
observation = list(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) #not used but␣
↪didn't know how to put the R equivalent of the python None
filename = 'multi_observations_plains'

conditional_logistic_regression(filename, step_size, col_names, observation)
```

Social Density

[128]:
```
step_size = list(5,10)
col_names = list('social_density')
observation = list(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) #not used but␣
↪didn't know how to put the R equivalent of the python None
filename = 'multi_observations_plains'
```

```
conditional_logistic_regression(filename, step_size, col_names, observation)
```

Group center feature

[3]:
```
step_size = list(5,10)
my_datal = list()
for (distance in seq_along(step_size)){
    my_files = list.files(pattern = paste('dc_positions0.*?df_',␣
 ↪toString(step_size[[distance]]), '.*?.csv', sep=''))
    my_files = my_files[order(nchar(my_files), my_files)]
    my_filesl = lapply(my_files, read.csv)
    my_datal[[distance]] = my_filesl
}
```

[4]:
```
my_files
```

1. 'dc_positions0_ind0df_10meter_step_size.csv' 2. 'dc_positions0_ind1df_10meter_step_size.csv'
3. 'dc_positions0_ind2df_10meter_step_size.csv' 4. 'dc_positions0_ind3df_10meter_step_size.csv'
5. 'dc_positions0_ind4df_10meter_step_size.csv' 6. 'dc_positions0_ind5df_10meter_step_size.csv'
7. 'dc_positions0_ind6df_10meter_step_size.csv' 8. 'dc_positions0_population_df_10meter_step_size.csv'

[130]:
```
multiple_times = list()
output_dfs = list()

for (t in seq_along(my_datal)) {
    mclogit_output = list()
    for (i in seq_along(my_datal[[t]])) {
        print(t)
        print(i)
        clogit = coef(mclogit(cbind(my_datal[[t]][[i]]$label,␣
 ↪my_datal[[t]][[i]]$step) ~ direction_change, data=my_datal[[t]][[i]]))
        mclogit_output[[i]] = as.data.frame(clogit)
    }
    multiple_times[[t]] = mclogit_output
    spatial_df = cbind(multiple_times[[t]])
    output_dfs[[t]] = spatial_df
}
```

[112]:
```
output_df = do.call(rbind, output_dfs)
df = data.frame(matrix(unlist(output_df), nrow=length(output_df), byrow=T),␣
 ↪stringsAsFactors=FALSE)
write.table(df, file='/Users/taylorcarter/Desktop/dc_positions0_spatial_output.
 ↪csv', row.names=FALSE, col.names='beta', sep=",")
```

Recently Used Space Feature

[105]:
```
step_size = list(5,10)
my_datal = list()
for (distance in seq_along(step_size)){
    my_files = list.files(pattern = paste('ru_positions0.*?df_',␣
 ↪toString(step_size[[distance]]), '.*?.csv', sep=''))
    my_files = my_files[order(nchar(my_files), my_files)]
    my_filesl = lapply(my_files, read.csv)
    my_datal[[distance]] = my_filesl
}
```

[106]:
```
my_files
```

1. 'ru__positions0__ind0df__10meter__step__size.csv' 2. 'ru__positions0__ind2df__10meter__step__size.csv'
3. 'ru__positions0__ind3df__10meter__step__size.csv' 4. 'ru__positions0__ind4df__10meter__step__size.csv'
5. 'ru__positions0__ind5df__10meter__step__size.csv' 6. 'ru__positions0__ind6df__10meter__step__size.csv'
7. 'ru__positions0__population__df__10meter__step__size.csv'

[131]:
```
multiple_times = list()
output_dfs = list()

for (t in seq_along(my_datal)) {
    mclogit_output = list()
    for (i in seq_along(my_datal[[t]])) {
        print(t)
        print(i)
        clogit = coef(mclogit(cbind(my_datal[[t]][[i]]$label,␣
 ↪my_datal[[t]][[i]]$step) ~ recently_used_space, data=my_datal[[t]][[i]]))
        mclogit_output[[i]] = as.data.frame(clogit)
    }
    multiple_times[[t]] = mclogit_output
    spatial_df = cbind(multiple_times[[t]])
    output_dfs[[t]] = spatial_df
}
```

[108]:
```
output_df = do.call(rbind, output_dfs)
df = data.frame(matrix(unlist(output_df), nrow=length(output_df), byrow=T),␣
 ↪stringsAsFactors=FALSE)
write.table(df, file='/Users/taylorcarter/Desktop/ru_positions0_spatial_output.
 ↪csv', row.names=FALSE, col.names='beta', sep=",")
```

Social Density Feature

[101]:
```
step_size = list(5,10)
my_datal = list()
for (distance in seq_along(step_size)){
    my_files = list.files(pattern = paste('sd_positions0.*?df_',␣
 ↪toString(step_size[[distance]]), '.*?.csv', sep=''))
```

```
    my_files = my_files[order(nchar(my_files), my_files)]
    my_filesl = lapply(my_files, read.csv)
    my_datal[[distance]] = my_filesl
}
```

[102]:
```
my_files
```

1. 'sd_positions0_ind0df_10meter_step_size.csv' 2. 'sd_positions0_ind1df_10meter_step_size.csv'
3. 'sd_positions0_ind3df_10meter_step_size.csv' 4. 'sd_positions0_ind4df_10meter_step_size.csv'
5. 'sd_positions0_ind5df_10meter_step_size.csv' 6. 'sd_positions0_population_df_10meter_step_size.csv'

[132]:
```
multiple_times = list()
output_dfs = list()

for (t in seq_along(my_datal)) {
    mclogit_output = list()
    for (i in seq_along(my_datal[[t]])) {
        print(t)
        print(i)
        clogit = coef(mclogit(cbind(my_datal[[t]][[i]]$label,␣
 ↪my_datal[[t]][[i]]$step) ~ social_density, data=my_datal[[t]][[i]]))
        mclogit_output[[i]] = as.data.frame(clogit)
    }
    multiple_times[[t]] = mclogit_output
    spatial_df = cbind(multiple_times[[t]])
    output_dfs[[t]] = spatial_df
}
```

[104]:
```
output_df = do.call(rbind, output_dfs)
df = data.frame(matrix(unlist(output_df), nrow=length(output_df), byrow=T),␣
 ↪stringsAsFactors=FALSE)
write.table(df, file='/Users/taylorcarter/Desktop/sd_positions0_spatial_output.
 ↪csv', row.names=FALSE, col.names='beta', sep=",")
```

0.0.6 Multiple Features + Multiple Observations

[133]:
```
step_size = list(5,10)
my_datal = list()
col_names = list('group_center', 'social_density', 'recently_used_space')
observation = list(0, 1, 2)
for (obsv in observation){
    for (distance in seq_along(step_size)){
        my_files = list.files(pattern = paste('multi_feature_positions',␣
 ↪toString(obsv),'.*?df_', toString(step_size[[distance]]), '.*?.csv', sep=''))
        my_files = my_files[order(nchar(my_files), my_files)]
        print(my_files)
        my_filesl = lapply(my_files, read.csv)
```

```
        my_datal[[distance]] = my_filesl
    }

    multiple_times = list()
    output_dfs = list()

    for (t in seq_along(my_datal)) {
        mclogit_output = list()
        for (i in seq_along(my_datal[[t]])) {
            print(t)
            print(i)
            clogit = coef(mclogit(cbind(my_datal[[t]][[i]]$label,␣
 ↪my_datal[[t]][[i]]$step) ~ direction_change + social_density +␣
 ↪recently_used_space,
                                  data=my_datal[[t]][[i]]))
            mclogit_output[[i]] = as.data.frame(clogit)
        }
        multiple_times[[t]] = mclogit_output
        spatial_df = cbind(multiple_times[[t]])
        output_dfs[[t]] = spatial_df
    }

    output_df = do.call(rbind, output_dfs)
    df = data.frame(matrix(unlist(output_df), nrow=length(output_df), byrow=T),␣
 ↪stringsAsFactors=FALSE)
    write.table(df, file='/Users/Taylor/Desktop/multi_feat_positions',␣
 ↪toString(obsv), '_spatial_output.csv', row.names=FALSE, col.names=col_names,␣
 ↪sep=",")
}
```

0.0.7 Understanding model

1 - True data greater

[134]:
```
step_size = list(5)
# radius_around_ind = list(1,5,10)
my_datal = list()
col_names = list('true_data_greater', 'false_data_greater')
for (distance in seq_along(step_size)){
    my_files = list.files(pattern = paste('model_testing.*?_step_size.csv',␣
 ↪sep=''))
    my_files = my_files[order(nchar(my_files), my_files)]
    print(my_files)
    my_filesl = lapply(my_files, read.csv)
    my_datal[[distance]] = my_filesl
}
```

```
multiple_times = list()
output_dfs = list()

for (t in seq_along(my_datal)) {
    mclogit_output = list()
    for (i in seq_along(my_datal[[t]])) {
        print(t)
        print(i)
        clogit = coef(mclogit(cbind(my_datal[[t]][[i]]$label,␣
 ↪my_datal[[t]][[i]]$step) ~ true_data_greater,
                              data=my_datal[[t]][[i]]))
        mclogit_output[[i]] = as.data.frame(clogit)
    }
    multiple_times[[t]] = mclogit_output
    spatial_df = cbind(multiple_times[[t]])
    output_dfs[[t]] = spatial_df
}

output_df = do.call(rbind, output_dfs)
df = data.frame(matrix(unlist(output_df), nrow=length(output_df), byrow=T),␣
 ↪stringsAsFactors=FALSE)
write.table(df, file='/Users/taylorcarter/Desktop/
 ↪model_testing_true_greater_output.csv', row.names=FALSE, col.
 ↪names='true_data_greater', sep=",")
```

2 - False data greater

[135]:
```
step_size = list(5)
# radius_around_ind = list(1,5,10)
my_datal = list()
col_names = list('true_data_greater', 'false_data_greater')
for (distance in seq_along(step_size)){
    my_files = list.files(pattern = paste('model_testing.*?_step_size.csv',␣
 ↪sep=''))
    my_files = my_files[order(nchar(my_files), my_files)]
    print(my_files)
    my_filesl = lapply(my_files, read.csv)
    my_datal[[distance]] = my_filesl
}

multiple_times = list()
output_dfs = list()

for (t in seq_along(my_datal)) {
    mclogit_output = list()
    for (i in seq_along(my_datal[[t]])) {
        print(t)
```

```
        print(i)
        clogit = coef(mclogit(cbind(my_datal[[t]][[i]]$label,␣
 ↪my_datal[[t]][[i]]$step) ~ false_data_greater,
                              data=my_datal[[t]][[i]]))
        mclogit_output[[i]] = as.data.frame(clogit)
    }
    multiple_times[[t]] = mclogit_output
    spatial_df = cbind(multiple_times[[t]])
    output_dfs[[t]] = spatial_df
}

output_df = do.call(rbind, output_dfs)
df = data.frame(matrix(unlist(output_df), nrow=length(output_df), byrow=T),␣
 ↪stringsAsFactors=FALSE)
write.table(df, file='/Users/taylorcarter/Desktop/
 ↪model_testing_false_greater_output.csv', row.names=FALSE, col.
 ↪names='false_data_greater', sep=",")
```

3 - Combining true_data_greater 'feature' with false data greater 'feature'

[136]:
```
step_size = list(5)
# radius_around_ind = list(1,5,10)
my_datal = list()
col_names = list('true_data_greater', 'false_data_greater')
for (distance in seq_along(step_size)){
    my_files = list.files(pattern = paste('model_testing.*?_step_size.csv',␣
 ↪sep=''))
    my_files = my_files[order(nchar(my_files), my_files)]
    print(my_files)
    my_filesl = lapply(my_files, read.csv)
    my_datal[[distance]] = my_filesl
}

multiple_times = list()
output_dfs = list()

for (t in seq_along(my_datal)) {
    mclogit_output = list()
    for (i in seq_along(my_datal[[t]])) {
        print(t)
        print(i)
        clogit = coef(mclogit(cbind(my_datal[[t]][[i]]$label,␣
 ↪my_datal[[t]][[i]]$step) ~ true_data_greater + false_data_greater,
                              data=my_datal[[t]][[i]]))
        mclogit_output[[i]] = as.data.frame(clogit)
    }
    multiple_times[[t]] = mclogit_output
```

```
    spatial_df = cbind(multiple_times[[t]])
    output_dfs[[t]] = spatial_df
}

output_df = do.call(rbind, output_dfs)
df = data.frame(matrix(unlist(output_df), nrow=length(output_df), byrow=T),␣
 ↪stringsAsFactors=FALSE)
write.table(df, file='/Users/taylorcarter/Desktop/
 ↪model_testing_true_and_false_output.csv', row.names=FALSE, col.
 ↪names=col_names, sep=",")
```

[ ]: