Spatial analyses on a round(ish) planet

Søren Faurby

9/23/2022

Dependencies:

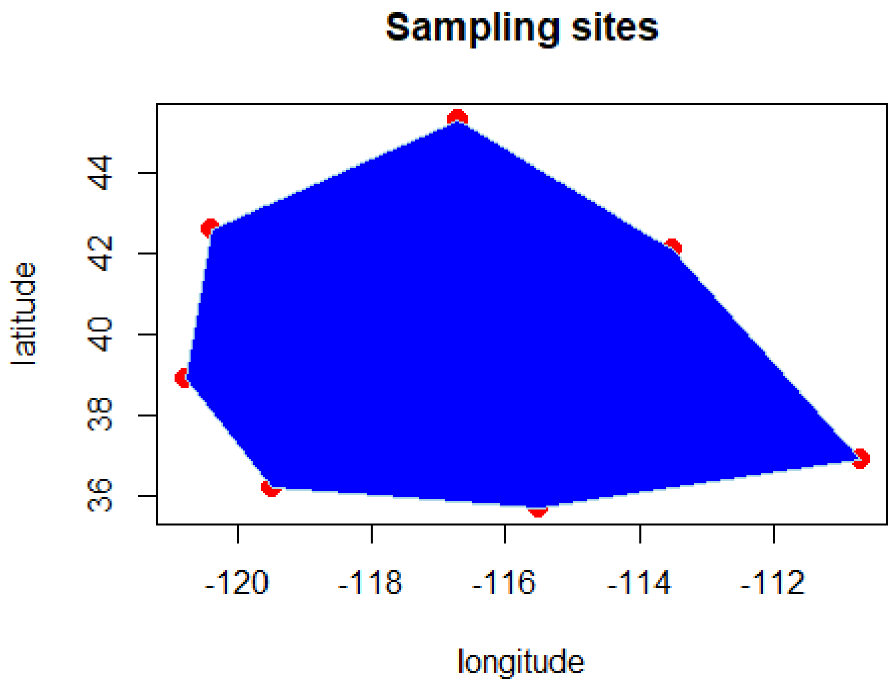
```
library(geosphere)
library(raster)
library(sf)
library(ncf)
library(spdep)
library(wiqid)
library(spatialreg)
```

1 Incorporating a round earth into what we learned Monday

1a: Convex hulls Incorporating a round earth into what we learned Monday

Monday we plotting a convex hull of some fake data

```
# generate sampling locations
longitude = c(-116.7, -120.4, -116.7, -113.5, -115.5,
               -120.8, -119.5, -113.7, -113.7, -110.7)
latitude = c(45.3, 42.6, 38.9, 42.1, 35.7, 38.9,
              36.2, 39, 41.6, 36.9)
# this command simply combines the two arrays longitude and latitude into a
shared matrix
sampling_sites = cbind(longitude, latitude)
plot(sampling_sites, cex=2, pch=20, col="red", main= 'Sampling sites')
polygon(sampling_sites[chull(sampling_sites),], col ="blue", border= "light
blue")
```

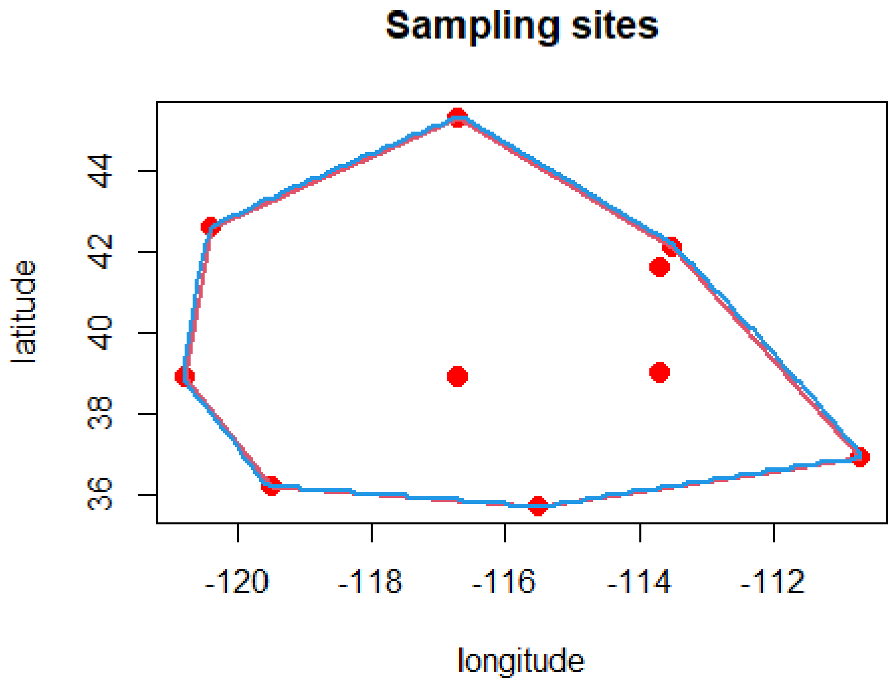


I noted that the hull actually is marginally wrong on a spherical planet, but it is a lot trickier to calculate the hull properly than to use the method we used Monday (I could not find a package doing it properly but the code below will in most cases give the right answer)

```r
library(geosphere)
#First we identify a large number of points along the great circle route
between all the sample sites
INTERMEDIATE=c()
for (i in 1:nrow(sampling_sites)) for (j in i:nrow(sampling_sites))
INTERMEDIATE=rbind(INTERMEDIATE,
gcIntermediate(sampling_sites[i,],sampling_sites[j,], n=10000))
INTERMEDIATE=round(INTERMEDIATE,1)
INTERMEDIATE=unique(INTERMEDIATE)
#Next we identify the edges of the polygon. Since we have rounded all
latitudes to 1 decimal degree this would be the smallest and largest latitude
for each longitude
UNIQIUE_LONG=sort(unique(INTERMEDIATE[,1]))
EDGES=c()
for (i in UNIQIUE_LONG) EDGES=rbind(EDGES,  c(i,
min(INTERMEDIATE[,2][INTERMEDIATE[,1]==i])), c(i,
max(INTERMEDIATE[,2][INTERMEDIATE[,1]==i])))
#Finally we need to orient the vertices in a resulting polygon
CENTER=c(mean(EDGES[,1]), mean(EDGES[,2]))
ANGLES=numeric(nrow(EDGES))
for (i in 1:nrow(EDGES)) ANGLES[i]=atan2(CENTER[1]-EDGES[i,1], CENTER[2]-
EDGES[i,2])
True_Hull=EDGES[order(ANGLES),]
```

We can now plot the difference

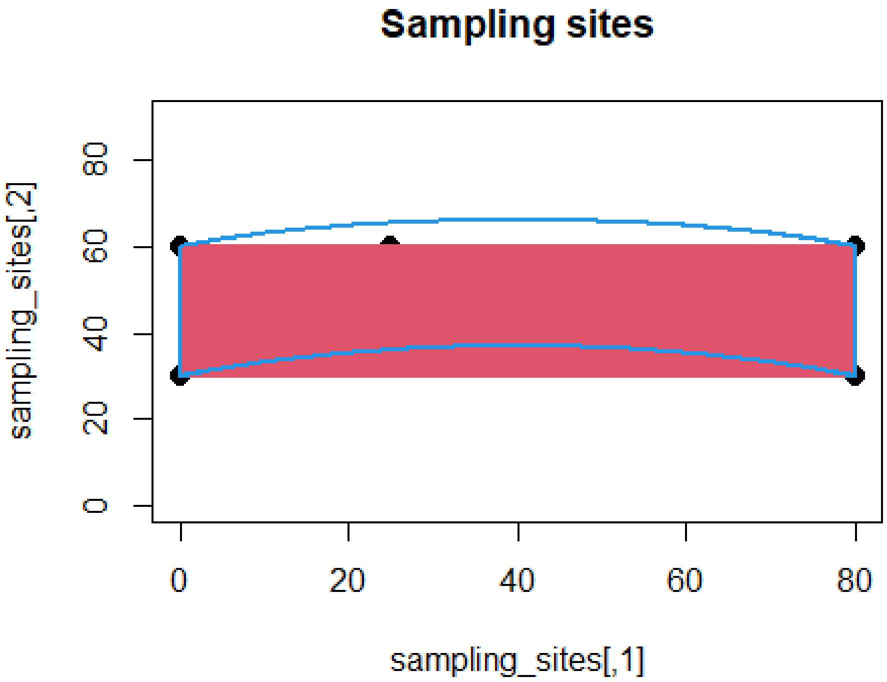
```
# generate sampling locations
plot(sampling_sites, cex=2, pch=20, col="red", main="Sampling sites")
polygon(sampling_sites[chull(sampling_sites),], border=2, lwd=2)
polygon(True_Hull, border=4, lwd=2)
```



We see that the difference in this case is minute (and in fact we have created some rounding errors with the round command which might be as important as the problem we tried to fix here, for this set of coordinates). The effect can be much greater if we have few records which are far apart. Below is one example with the convex hull as defined Monday in red and the proper one plotted as a blue line. **Note that the code shown here is experimental and while it here gives the right answer, it can under some circumstances produce clearly nonsensical results so if you use it you always have to visually inspect the the produced hulls to make sure they make sense**

```
# generate sampling locations
sampling_sites = rbind(c(80,30), c(80,60), c(25,60), c(0, 60), c(0,30))

INTERMEDIATE=c()
for (i in 1:nrow(sampling_sites)) for (j in i:nrow(sampling_sites))
INTERMEDIATE=rbind(INTERMEDIATE,
gcIntermediate(sampling_sites[i,],sampling_sites[j,], n=20000))
INTERMEDIATE=round(INTERMEDIATE,1)
INTERMEDIATE=unique(INTERMEDIATE)
UNIQIUE_LONG=sort(unique(INTERMEDIATE[,1]))
```

```
EDGES=c()
for (i in UNIQIUE_LONG) EDGES=rbind(EDGES,  c(i,
min(INTERMEDIATE[,2][INTERMEDIATE[,1]==i])), c(i,
max(INTERMEDIATE[,2][INTERMEDIATE[,1]==i])))
CENTER=c(mean(EDGES[,1]), mean(EDGES[,2]))
ANGLES=numeric(nrow(EDGES))
for (i in 1:nrow(EDGES)) ANGLES[i]=atan2(CENTER[1]-EDGES[i,1], CENTER[2]-
EDGES[i,2])
True_Hull=EDGES[order(ANGLES),]

plot(sampling_sites, cex=2, pch=20, main="Sampling sites", ylim=c(0,90))
polygon(sampling_sites[chull(sampling_sites),], border=2, lwd=2, col=2)
polygon(True_Hull, border=4, lwd=2)
```

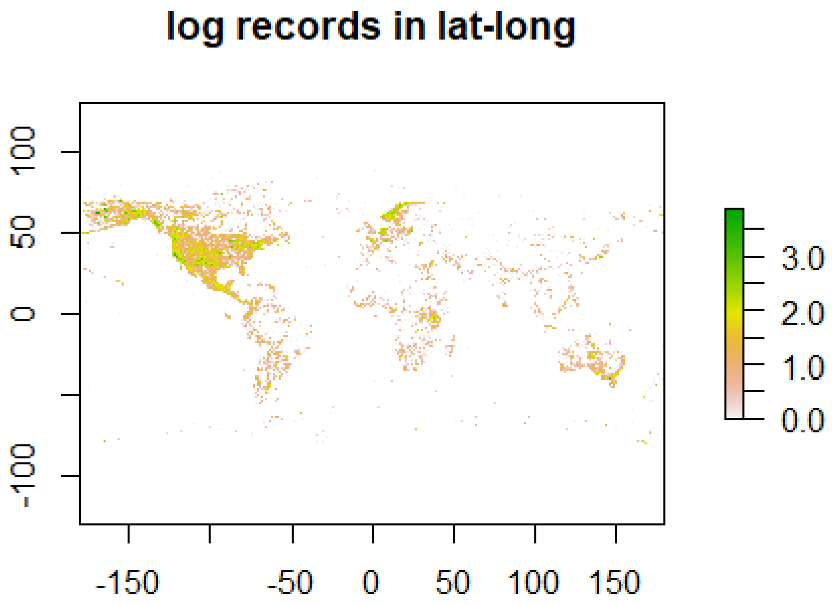


1b: Projection during rasterization.

Monday we were rasterizing the number of records of carnivora matching specimens in museums on a lat long projection. Let us first redo this.

```
library(raster)
Carnivora_Data=read.table("C:\\Users\\xfauso\\Documents\\Teaching\\2022\\PhDR
\\Data\\GBIF_Carnivora.txt")
locations = cbind(Carnivora_Data$decimalLongitude,
Carnivora_Data$decimalLatitude)
species_occurences = SpatialPoints(locations, proj4string
=CRS("+proj=longlat +datum=WGS84"))
raw_raster = raster(nrow=180, ncol=360) #An empty  raster with a resolution
```

```
of 1 by 1 degree.
counts_per_cell = rasterize(species_occurrences,raw_raster,fun='count')
plot(log10(counts_per_cell), main="log records in lat-long")
```



We can change the projection of the raster using `projectRaster()` which behaves somewhat similar to the other projection functions we have used. We will try to project it into the CEA projection which is used in the Phylacine data. We will start by loading any random raster of the Phylacine data.

```
library(raster)
Source_Raster=raster("C:\\Users\\xfauso\\Documents\\Phylacine\\1.21\\Data\\Ra
nges\\Present_natural\\Abditomys_latidens.tif")
#This is just the first file in the folder which happens to be a rodent from
the Philippines

counts_proj = projectRaster(counts_per_cell,crs=projection(Source_Raster))
```

You will likely get a warning similar to “3740 projected point(s) not finite”. The reasons for this is effectively rounding errors and is telling us that some points in the behrman projection appaer to correspond to non existing latitudes some as e.g. 90.1 degress north. This warning can be ignored.

There is not a one to one matching between cells in each projection

```
print(dim(Source_Raster))

## [1] 142 360   1
```

```
#This means that the raster has 360 columns and 142 rows
print(dim(counts_proj))

## [1] 125 374   1

#This means that the raster has 374 columns and 125 rows
```

Let us change the resolution of the counts_proj into the same as the one for Source_Raster. We will do this with resample().

```
library(raster)
counts_proj=resample(counts_proj, Source_Raster, method='bilinear')
```

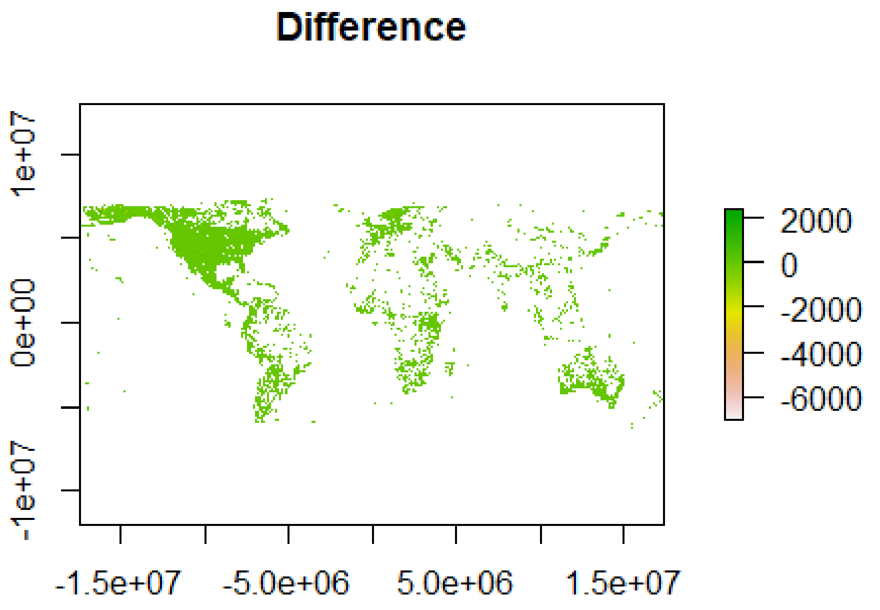
Let us re-rasterize but this time we will rasterize directly in the equal area projection.

```
library(sf)
library(raster)
locations_Proj =
st_transform(st_as_sf(species_occurrences),projection(Source_Raster))
counts_Direct = rasterize(locations_Proj,Source_Raster,fun='count')
Difference=counts_proj-counts_Direct
plot(Difference, main="Difference")
```

We can now calculate the difference between the two and plot it.

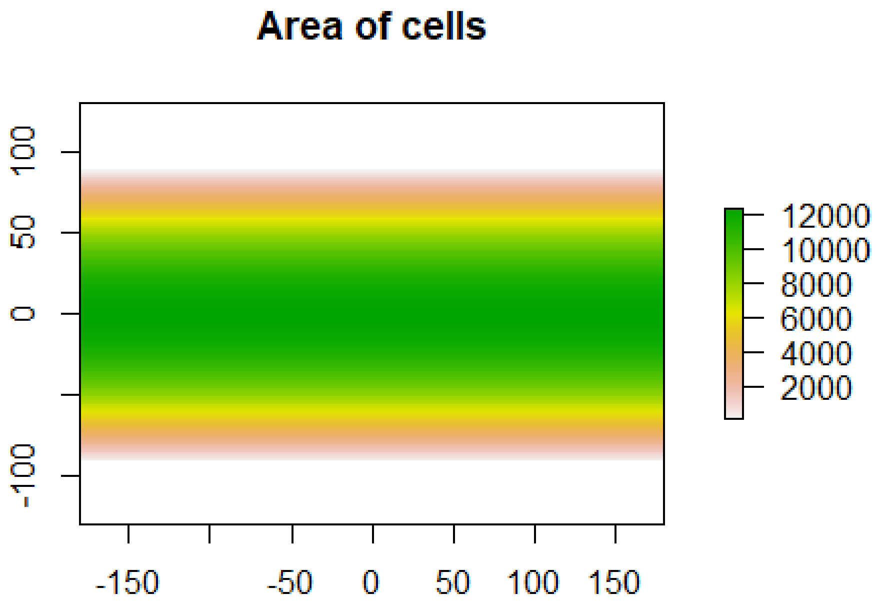
```
library(raster)
Difference=counts_proj-counts_Direct
plot(Difference, main="Difference")
```

As you can see nearly all cells have differences. This means that whenever possible rasters should be created in the projection we want to analyze rather than created and then projected.



To look at this in another way we can plot the area of all cells in lat-long projection.

```
plot(area(counts_per_cell), main="Area of cells")
```



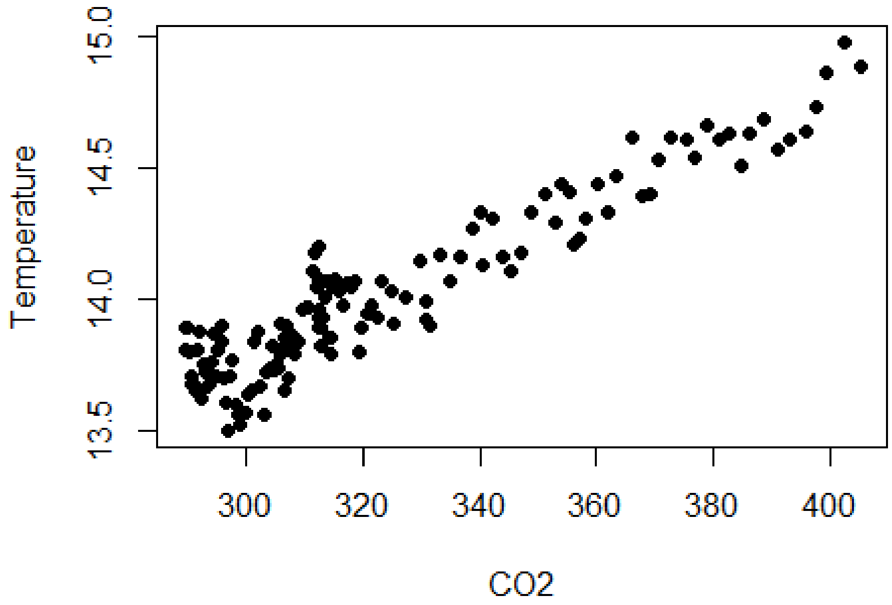
What we see here is that because the area of cells in a cell of e.g. 1 by 1 degree varies substantially across latitude any analyses in lat-long may create substantial biases.

2 Understanding autocorrelation

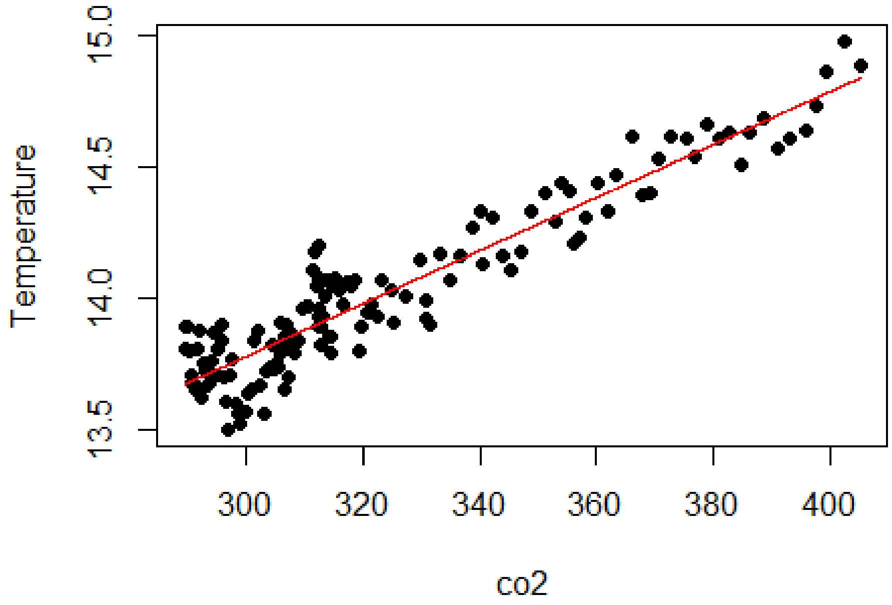
Disclaimer: Some of the following examples are borrowed from the tutorial on auto-correlation at https://rspatial.org/raster/analysis/3-spauto.html.

Let us first look at the linear relationship between global temperature and CO2 based on data, which you should already have downloaded from GitHub.

```
global_temp =
read.csv('C:\\Users\\xfauso\\Documents\\Teaching\\2022\\PhDR\\Data\\global_te
mp.txt',sep='\t')

co2 =
read.csv('C:\\Users\\xfauso\\Documents\\Teaching\\2022\\PhDR\\Data\\co2.txt',
sep='\t')

plot(co2$co2, global_temp$Temperature, pch = 16, xlab = "CO2", ylab =
"Temperature")
```



```
combined = cbind(global_temp,co2)
model = glm(Temperature ~ co2, family = gaussian ,data = combined)
xweight = seq(range(co2$co2)[1], range(co2$co2)[2], 0.01)
yweight = predict(model, list(co2 = xweight),type="response")
plot(co2$co2,global_temp$Temperature, pch = 16, xlab = "co2", ylab =
"Temperature")
lines(xweight, yweight,col='red')
```



Let us now look at temporal auto-correlation which hopefully helps us understand the concept of autocorrelation in general. If your data are auto-correlated, it means that they are not independent data points but that they are correlated to some extent. When using auto-correlated data without correcting for the correlation, one usually greatly overestimates the effective sample size.

For example if you want to measure a person's weight through time, you would expect two measurements which are close to each other in time to also be similar in the measured variable. To measure the degree of association over time, we can compute the correlation of each observation with the next observation.

We can check for temporal auto-correlation. We are going to compute the "one-lag" auto-correlation, which means that we compare each value to its immediate neighbor, and not to other nearby values.

Temporal auto-correlation

For CO2:

```
values = co2$co2
a = values[-length(values)] #We here use  "-" in the square brackets. This 
notation means that tne relevant datapoint(s) are removed
b = values[-1]
print(cor(a,b))
```

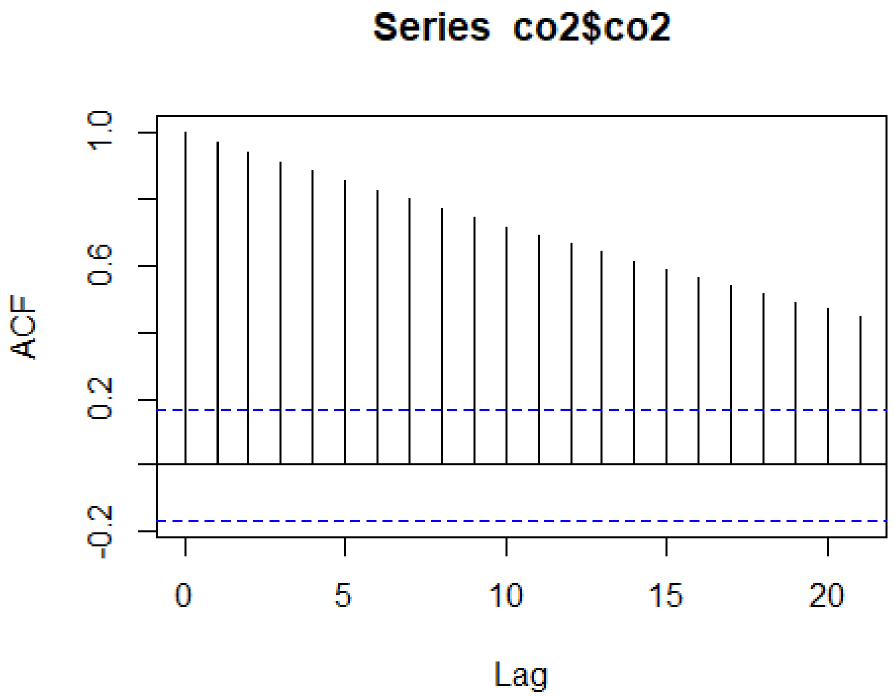
```
## [1] 0.99992
```

And for temperature:

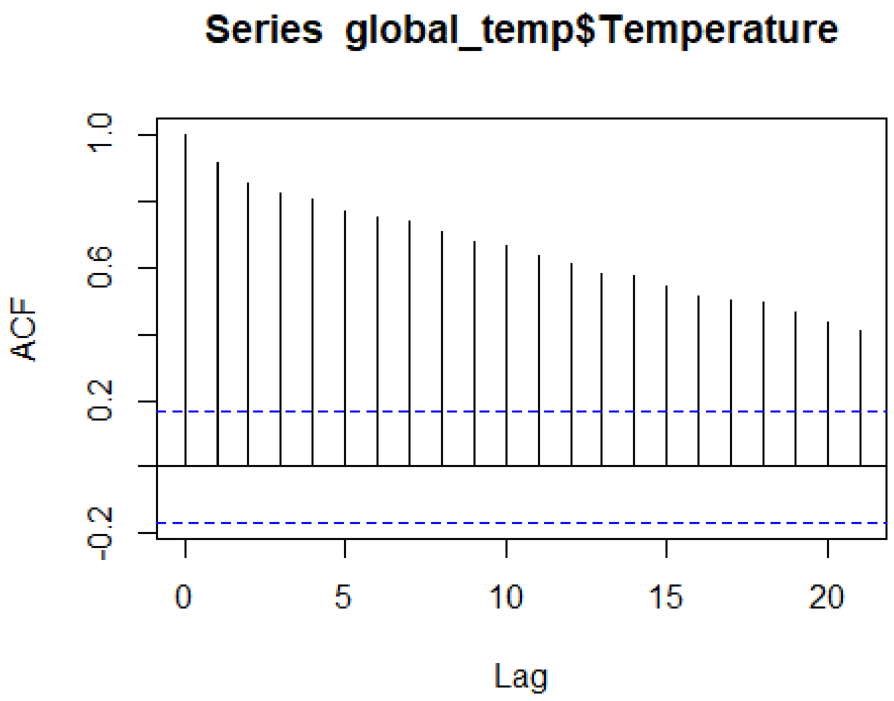
```
values = global_temp$Temperature
a = values[-length(values)]
b = values[-1]
cor(a,b)

## [1] 0.9426816
```

These values indicate a very strong positive temporal auto-correlation for the "one-lag" method. There are also integrated and more elegant ways in R for determining auto-correlation than our manual implementation of the "one-lag" auto-correlation. For example the `acf()` function computes the autocorrelation for several lags:

```
acf(co2$co2)
```



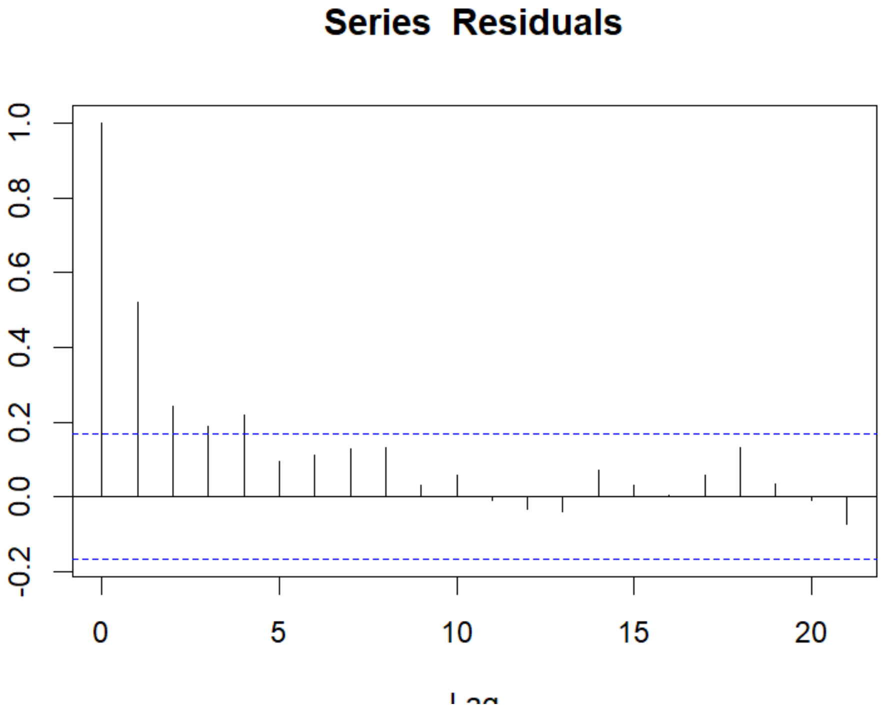
```
acf(global_temp$Temperature)
```



We see that both global temperature and CO2 content of the atmosphere each show a strongly positive temporal auto-correlation.

Analytically this auto-correlation is not always a problem. More specifically we care that there is no autocorrelation in the model residuals. Measurements can still be considered statistically independent if measurements close in time are similar, as long as this similarity is predicted by the model. We can however calculate if the temporal autocorrelation in temperature and CO2 would also be an analytical problem if we wanted to model the changes through time in CO2.

```
Residuals=resid(glm(Temperature ~ co2, data = combined))
acf(Residuals)
```

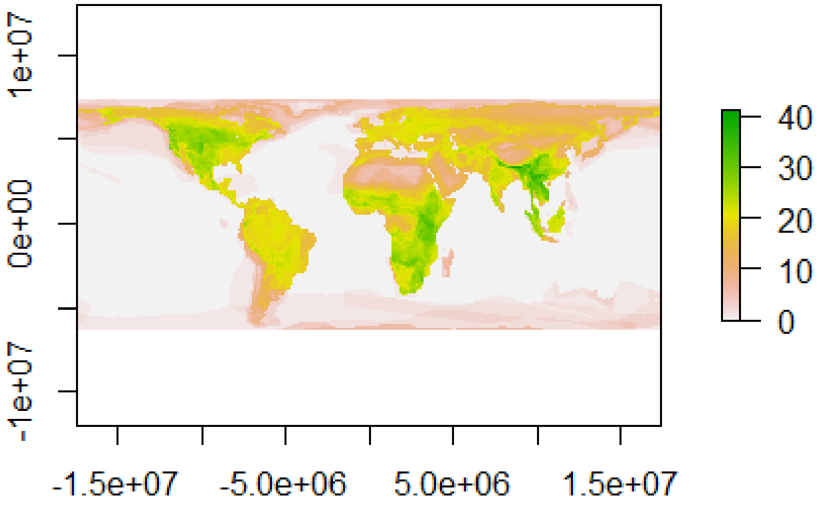


We see that the residuals are significantly autocorrelated at least with low lags of 1 or 2. This means that values of CO2 1 or 2 years apart are significantly more similar than would be expected by our very simple model of a linear model of changes in CO2 over time. This autocorrelation would not mean that the line of change we added to an earlier plot is biased but it does mean that we are inflating the precision with which we can say how much CO2 is predicted to increase over time.

We will now start the main topic of the day which is how to analytically deal with spatial autocorrelation which means that areas close to each other are inherently similar.

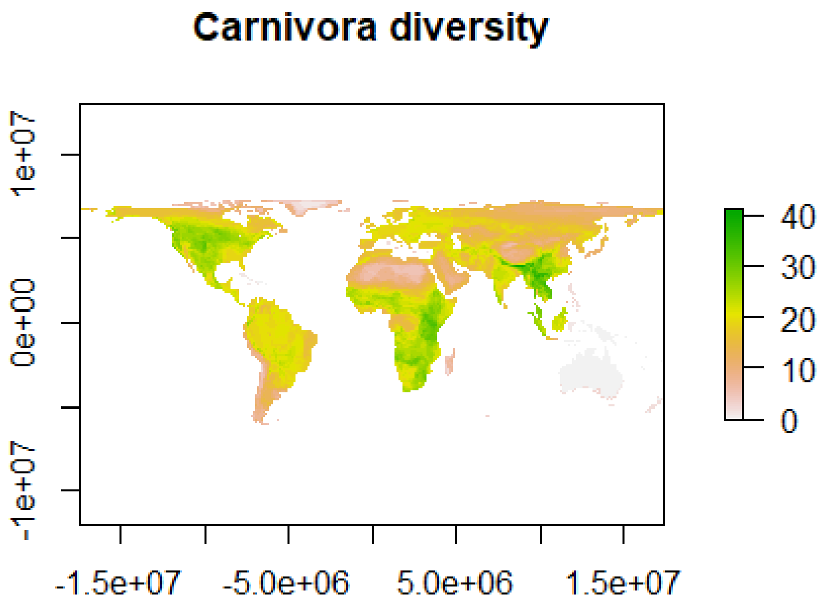
3 Input data

In our example mammal species diversity is the response variable, i.e. we want to figure out how species diversity responds to different predictors. You will need to produce the species diversity raster by adding up the presence/absence rasters belonging to each of the species of your mammal group of choice. Start by crating a raster of the diversity of the desired clade. You did this Monday and should be able to reuse your code. Below you see the result of how it would look if you use all species of the order Carnivora



We will also load a shape of a world map to crop out only land cells (using the `mask()` function). This is very similar to what we did Monday when we removed all parts of the raster not inside the selected country. Here we will use it to remove all marine parts.

```
library(raster)
library(sf)

world_map =
st_read("C:\\Users\\xfauso\\Documents\\Teaching\\2022\\PhDR\\Data\\global\\ne
_110m_land\\ne_110m_land.shp")

## Reading layer `ne_110m_land' from data source `
C:\\Users\\xfauso\\Documents\\Teaching\\2022\\PhDR\\Data\\global\\ne_110m_lan
d\\ne_110m_land.shp' using driver `ESRI Shapefile'
## Simple feature collection with 127 features and 3 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: -180 ymin: -90 xmax: 180 ymax: 83.64513
## Geodetic CRS:  WGS 84

transformed_world = st_transform(world_map,projection(Diversity_raster))
world_spatial = as(transformed_world, "Spatial")
Diversity_raster = mask(Diversity_raster,world_spatial)
plot(Diversity_raster,main="Carnivora diversity")
```



```
## class       : RasterLayer
## dimensions  : 142, 360, 51120  (nrow, ncol, ncell)
## resolution  : 96486.27, 96514.96  (x, y)
## extent      : -17367529, 17367529, -6356742, 7348382  (xmin, xmax, ymin,
ymax)
## crs         : +proj=cea +lat_ts=30 +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84
+units=m +no_defs
## source      : memory
## names       : layer
## values      : 0, 41  (min, max)
```

Check the dimension of you raster using the `dim()` command, which tells you how many cells the raster contains along the x and y axis.

```
dim(Diversity_raster)

## [1] 142 360   1
```

And the resolution of the raster:

```
res(Diversity_raster)

## [1] 96486.27 96514.96
```

The resolution is scaled in meters, so this means that the cell size is ~96x96km i this raster. It is however slightly more complex than this because these are the projected dimensions. If we imagine that we saw the cells on the circle of a globe, they would be isosceles trapezoids. The range on the x axis is always one degree (but this corresponds to different sizes is kilometers) at both the top and bottom of the cell while the heights will vary between rows. The projected size is only identical to the true size at 30 degrees (You will also see that I am bad at following my own advice as I gave it Monday because this raster actually does not have fully quadratic cells).

You can get an idea of the coordinate system your raster data is stored in by checking the extent of the raster:

```
extent(Diversity_raster)

## class      : Extent
## xmin       : -17367529
## xmax       : 17367529
## ymin       : -6356742
## ymax       : 7348382
```

Full information about the projection can be seem with the command projection()

```
library(raster)
projection(Diversity_raster)

## [1] "+proj=cea +lat_ts=30 +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m
+no_defs"
```

This projection is often referred to as Behrmann projection. We can briefly go through what the information means

“cea” means that the type of projection is a cylindrical equal area projection (as I described in the lecture).

“+lat_ts=30” means that the only latitude where no distortion of the shape is seen in 30 degrees North or South.

“+lon_0=0” specifies the longitude assigned the value 0.

“+x_0=0 +y_0=0” specifies what is called false easting and false northing which are values added to all cells which sometimes is done to make sure all values are positive which might be beneficial for some purposes"

“+datum=WGS84” specifies the underlying coordinate system. The logic of the coordinate system is described in detail in Wikipedia

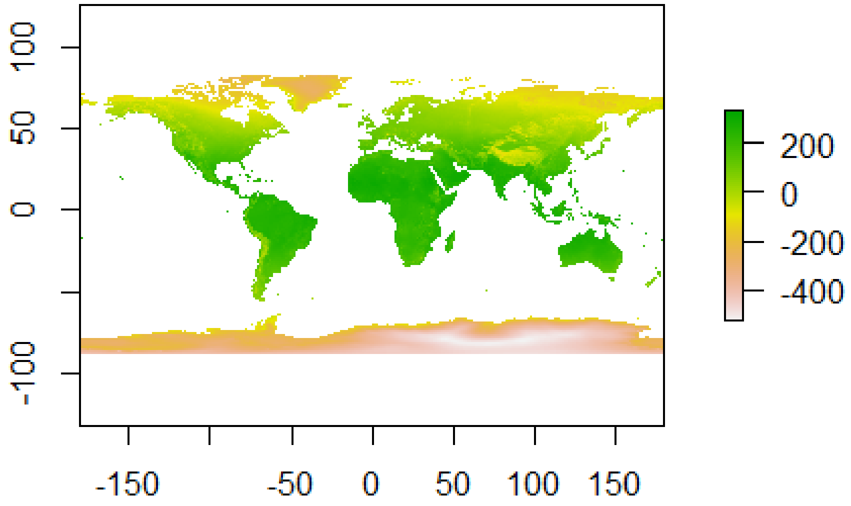
“+units=m” specified that the data is measures in meters.

“+no_defs” means no defaults and means that it does not use any information stores elsewhere in R.

Predictor data

Now it’s time to load the data for the predictor variables, which are the factors we want to test for correlation with mammalian species diversity. You can download whatever variables you want to check (for example get annual precipitation data if you want to test if this is a predictor of species diversity). A lot of useful climatic data can be found at **Chelsa** https://chelsa-climate.org/bioclim/, but feel free to use any data source you want to. In this example I will work with annual average temperature grid data, but you can pick other predictors instead:

```
library(raster)
temperature_raster = 
raster("C:\\Users\\xfauso\\Documents\\Teaching\\2022\\PhDR\\Other_data\\CHELSA_bio10_01.tif")
plot(temperature_raster)
```



```
extent(temperature_raster)

## class      : Extent 
## xmin       : -180.0001 
## xmax       : 179.9999 
## ymin       : -90.00014 
## ymax       : 83.99986

temperature_raster

## class      : RasterLayer 
## dimensions : 20880, 43200, 902016000  (nrow, ncol, ncell)
## resolution : 0.008333333, 0.008333333  (x, y)
## extent     : -180.0001, 179.9999, -90.00014, 83.99986  (xmin, xmax, ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +no_defs
## source     : 
C:\\Users\\xfauso\\Documents\\Teaching\\2022\\PhDR\\CHELSA_bio10_01.tif 
## names      : CHELSA_bio10_01 
## values     : -32768, 32767  (min, max)
```

What we have plotted here is annual mean temperature. The units are degrees Celcius times 10. This might seem like a peculiar unit but it means that the data can be stored as integers which is computationally convenient. Decimal numbers actually can only very rarely be stored correctly in R even though the errors generally do not matter in biology because we nearly always have greater problems.

Too see the issue look at these lines (in R == tests if the two statements on each side are identical)

```
(3 - 2.9)==0.1

## [1] FALSE

(30 - 29)/10==0.1

## [1] TRUE

(30 - 29)/10   -  (3 - 2.9) #Here e-17 means times 10 raised to the power of 
-17

## [1] -8.326673e-17
```

If you are using the data from Chelsa you will see that the resolution of the raster is incredibly high, which makes for very slow processing times for all operations on the raster. Let's therefore first reduce the data to a lower resolution.

If you want to rescale your raster to a coarser resolution you can use the aggregate() function. For purposes which will become familiar later we will aggregate in multiple steps. It would not be needed to aggregate in multiple steps if we only were interested in the final product. You can aggregate by multiple different functions and sometimes it might e.g. be desired to aggregate by max or min if you want to know the most extreme values within a cell but the default is aggregating by mean which we will also do here. (This step will take a while)

```
Agg_Temp_20=aggregate(temperature_raster, 6) #This step is rather time 
demanding
Agg_Temp_10=aggregate(Agg_Temp_20, 2)
Agg_Temp_2=aggregate(Agg_Temp_10, 5)
Agg_Temp_1=aggregate(Agg_Temp_2, 2)
```

We need to make sure that our predictor data (temperature) is in the same projection as the response variable data (species diversity). We therefore want to project the climate raster to the projection of the diversity raster. We noted before that there is not a one to one match between cells in different projections. Technically it is desirable to project in as high resolution but for computational reasons this might not be realistically possible and as we will see now it generally enough to project the data in a resolution ~10 times higher than we want the final product in.

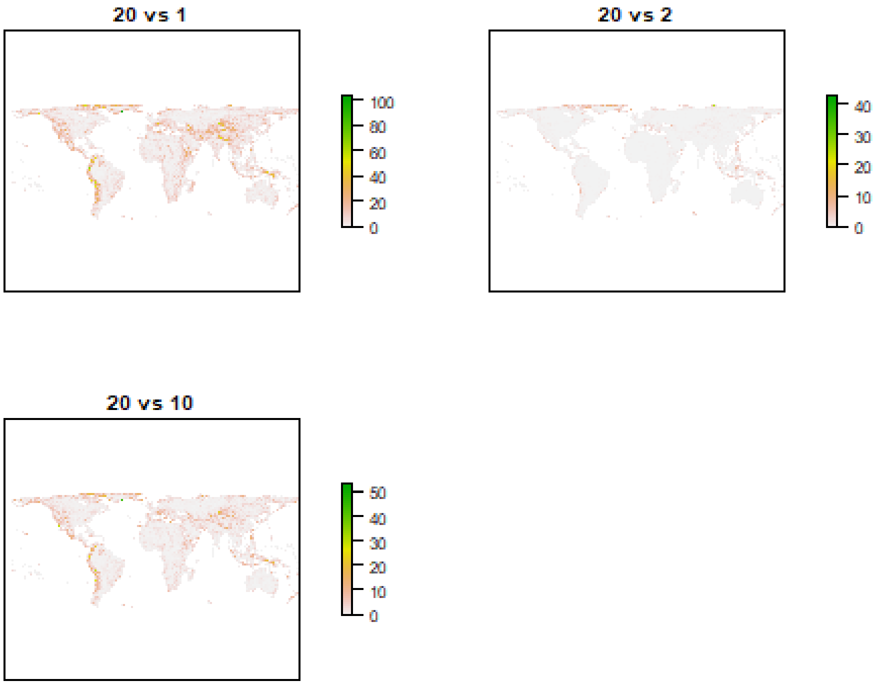
Now we will project the data and then aggregate the product into a resolution matching the target raster. Aggregate only works with integers so we first need to resample the projected raster into another raster which we can create with disaggregate of the Diversity_raster.

```
library(raster)
Proj_Temp_20= projectRaster(Agg_Temp_20,crs=projection(Diversity_raster))
DisAgg=disaggregate(Diversity_raster, 20)
Proj_Temp_20=resample(Proj_Temp_20, DisAgg, method='bilinear')
Proj_Temp_20=aggregate(Proj_Temp_20, 20)

Proj_Temp_10= projectRaster(Agg_Temp_10,crs=projection(Diversity_raster))
DisAgg=disaggregate(Diversity_raster, 10)
Proj_Temp_10=resample(Proj_Temp_10, DisAgg, method='bilinear')
Proj_Temp_10=aggregate(Proj_Temp_10, 10)

Proj_Temp_2= projectRaster(Agg_Temp_2,crs=projection(Diversity_raster))
DisAgg=disaggregate(Diversity_raster, 2)
Proj_Temp_2=resample(Proj_Temp_2, DisAgg, method='bilinear')
Proj_Temp_2=aggregate(Proj_Temp_2, 2)

Proj_Temp_1= projectRaster(Agg_Temp_1,crs=projection(Diversity_raster))
Proj_Temp_1=resample(Proj_Temp_1, Diversity_raster, method='bilinear')
```

We have here created four rasters (**Proj_Temp_20**, **Proj_Temp_10**, **Proj_Temp_2** and **Proj_Temp_1**). The first of these is the most precise while the last was fastest to create. We will use **Proj_Temp_20** for further analyses but let us first compare the four rasters. We can do this by plotting the absolute difference between the highest resoluton and any of the three others. After this we can calculate the mean difference between the rasters.

```
library(raster)

par(mfrow=c(2,2))
par(mar=c(2,2,2,3))
par(cex=0.5)
plot(abs(Proj_Temp_20-Proj_Temp_1), axes=F,  main="20 vs 1")
par(cex=0.5)
plot(abs(Proj_Temp_20-Proj_Temp_10), axes=F, main="20 vs 2")
par(cex=0.5)
plot(abs(Proj_Temp_20-Proj_Temp_2), axes=F, main="20 vs 10")

mean(rasterToPoints(abs(Proj_Temp_20-Proj_Temp_1))[,3])

## [1] 4.568019

mean(rasterToPoints(abs(Proj_Temp_20-Proj_Temp_2))[,3])
```

```
## [1] 1.796065
mean(rasterToPoints(abs(Proj_Temp_20-Proj_Temp_10))[,3])
## [1] 0.3523511
```



What we have learned here is that if we aggregate completely before projecting we misidentify the temperature by about 0.5 degree and by looking at the map we can see that these errors are most pronounced in mountain regions.

To remind you what we just did: - we loaded the raw climate data raster, which had a very high resolution - we reduced the resolution of the raster to a more handleable resolution using `aggregate()` - we projected the raster into the CEA projection, using `projectRaster()` - we rescaled the raster - we aggregated the raster to match the resolution of our diversity data.

4 Dealing with spatial autocorrelation

Similar to temporal auto-correlation, spatial auto-correlation means that two points or raster cells that are close to each other in space have similar values. In our example, in case our raster cells are spatially auto-correlated, we expect the species diversity values of two neighboring cells to be more similar to each other than to further away cells (on average).

If anyone of you are familiar with phylogenetic analyses the logic behind todays analyses is conceptually very close to the logic of why a PGLS is better than a normal linear model when analyzing related species. (If you do not have experience with phylogenetic analyses then just ignore this comment).

This auto-correlation can be exogenous (caused by some unknown/untested effects that effect neighboring cells in a similar manner) or endogenous (caused by the variable we're testing, e.g. temperature).

We will here assume the autocorrelation to be exogenous which generally lead to less biased conclusions. The more math focused of you can read further [here] (https://onlinelibrary.wiley.com/doi/full/10.1111/j.1466-8238.2007.00334.x)

In the following we're trying to quantify the degree to which neighboring raster cells are similar to each other. More specifically we're determining how similar their residuals are. If you do not remember the residuals are the difference between the actual values and what we would expect based on the set of predictors we use. We will investigate residuals using different definitions/thresholds of "neighborhood". We then include the determined autocorrelation into our model in order to account for the sum of the exogenous auto-correlation caused by unknown factors. Only by accounting for this can we measure the true effect of our tested variables on species diversity.

But before getting into neighborhoods etc. let us first fit a general linear model to our species diversity data, to see if they correlate with our predictor variable (temperature), without worrying about spatial auto-correlation and neighborhoods.

First we need to bring our raster data into a dataframe format. For this we extract the coordinates (raster cell centroids) using the `coordinates()` function. The coordinates should be identical for both of our rasters, check if that is the case by repeating the command below for your raster of predictor values. You should get the same coordinates and number of points for both rasters.

```
coordinates = coordinates(Diversity_raster)
```

Now extract the corresponding values from both rasters, using the `values()` function. Then we put the coordinates and the values from both rasters together into one dataframe.

```
species_div = values(Diversity_raster)
temperature = values(Proj_Temp_20)
data_merged = as.data.frame(cbind(coordinates,species_div,temperature))
```

There are a lot of NA values resulting from the water cells in the two rasters, which didn't have any values assigned to them. We need to remove those before continuing to work with the data. Therefore let's just remove all rows from the dataframe that contain NA's using the `complete.cases()` function, which only extracts complete rows with data:

```
final_data = data_merged[complete.cases(data_merged),]
```

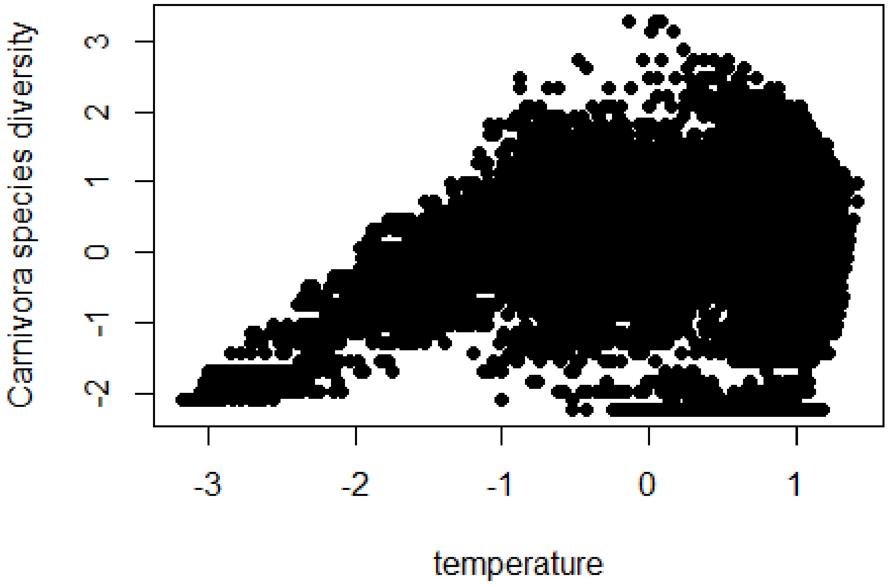
We will further scale the values for both the response and the predictor variables to be centered in 0 using the function `scale()`. This is in general good practice for understanding and comparing effect sizes:

```
final_data$species_div=scale(final_data$species_div)[,1]
final_data$temperature=scale(final_data$temperature)[,1]
```

Linear model (LM)

Now plot the final values of the predictor we want to test against the response variable to get a first impression on their relationship:

```
plot(final_data$temperature,final_data$species_div, pch = 16, xlab =
"temperature", ylab = "Carnivora species diversity")
```

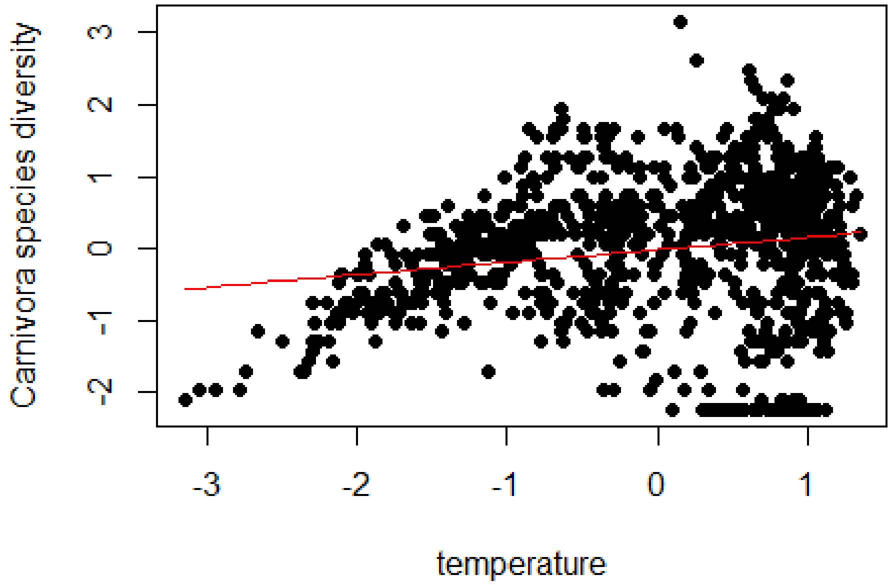


For faster computation, we'll go through the following steps just using a subsample of the data. Note that if this was for publication purposes you would need to run the final model using all of the data, but for now let's stick to the subsample (we are setting a seed with the `set.seed()` command to make sure the same samples will be selected for the tutorial purpose, in order to make sure that my explanations of the results make sense. You won't need to set a seed).

```
set.seed(42)
# take a random sample of n points from the dataframe
subsample = final_data[sample(nrow(final_data), 1000), ]
```

Now fit the linear model to the data and plot the results:

```
lm_model = lm(species_div~temperature,data=subsample)
xweight = seq(range(subsample$temperature)[1],
range(subsample$temperature)[2], 0.1)
yweight = predict(lm_model, list(temperature = xweight),type="response")
plot(subsample$temperature,subsample$species_div, pch = 16, xlab =
"temperature", ylab = "Carnivora species diversity")
lines(xweight, yweight, col="red")
```



You can check how strong your tested predictor affects the response variable by using the `summary()` command on the model:

```
summary(lm_model)

## 
## Call:
## lm(formula = species_div ~ temperature, data = subsample)
## 
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -2.4166 -0.5024  0.1893  0.6004  3.1443 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -0.01629    0.03132  -0.520    0.603    
## temperature  0.17275    0.03118   5.541 3.85e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.9903 on 998 degrees of freedom
## Multiple R-squared:  0.02985,    Adjusted R-squared:  0.02887 
## F-statistic:  30.7 on 1 and 998 DF,  p-value: 3.849e-08
```

In the `Coefficients` section you can find the effect size of your predictor (value after the predictor variable name). Also you can see how significant the effect is by looking at the p-value (`Pr(>|t|)`). In our case it is highly significant although not that strong of a predictor. By looking at the "Adjusted R-squared" you will see that only around 2.9% of variation in diversity is explained by temperature.

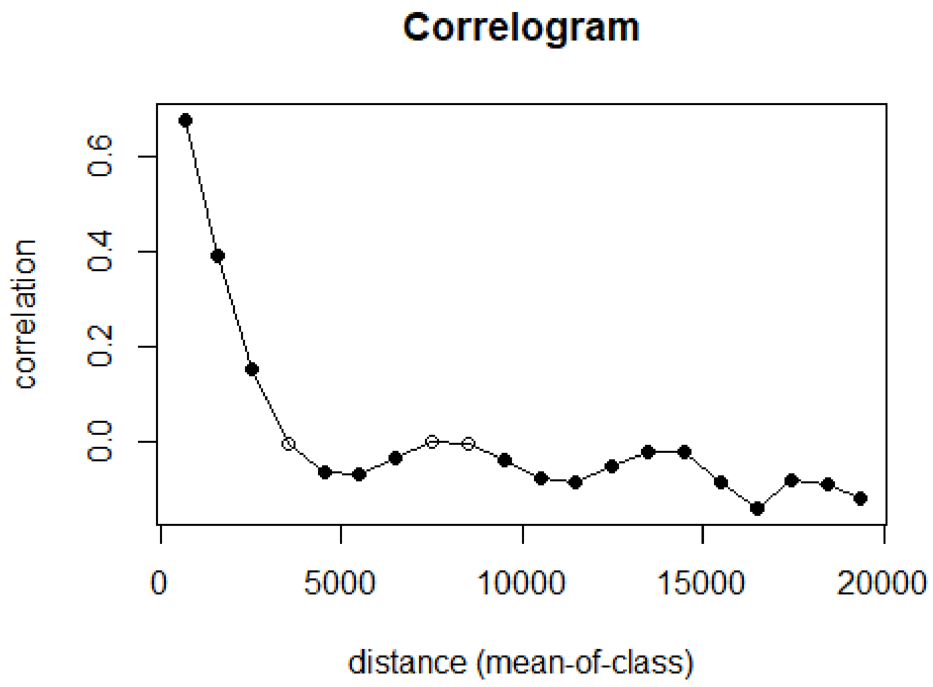
Earlier in the temporal auto-correlation example we determined the auto-correlation at different lags using the `acf()` function. For spatial data we can use the `correlog()` function which calculates the Moran's I (as mentioned in the introduction slides) for different distances of points. This is a measure of the spatial auto-correlation of the residuals at different distances.

First step is to transform our coordinates back from Behrman (CEA) into lat-lon format, since this is required to properly calculate the distance between cells. As was noted in the lecture CEA is area true but produces wrong distances between cells.

```
coordinates = cbind(subsample$x,subsample$y)
# define the current projection (Behrman)
coordinates_sp = SpatialPoints(coordinates,proj4string =
CRS(projection(Diversity_raster)))
# transform to lat-lon projection
coordinates_transformed = spTransform(coordinates_sp, CRS("+proj=longlat
+datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"))
# turn back into data frame
coordinates_df = as.data.frame(coordinates_transformed)
subsample$x = coordinates_df$coords.x1
subsample$y = coordinates_df$coords.x2
```

We can now estimate the degree of spatial autocorrelation

```
library(ncf)
autocorrelation_lm = correlog( subsample$x, subsample$y, lm_model$residuals,
increment=1000, latlon=T, resamp=100)

## 10  of  100 20  of  100 30  of  100 40  of  100 50  of  100 60  of  100 70
of  100 80  of  100 90  of  100 100  of  100

plot(autocorrelation_lm)
```



You can see that for distances (< 5000 km) we find a very strong positive spatial auto-correlation. This is a common pattern for spatial data and is the reason why we need to bother with accounting for this auto-correlation in our model.

Linear models including neighbourhoods (SAR model)

In order to formalize this spatial auto-correlation and integrate it into our linear model, we need some measure of neighborhood (i.e. a measure of what is considered a "nearby point").

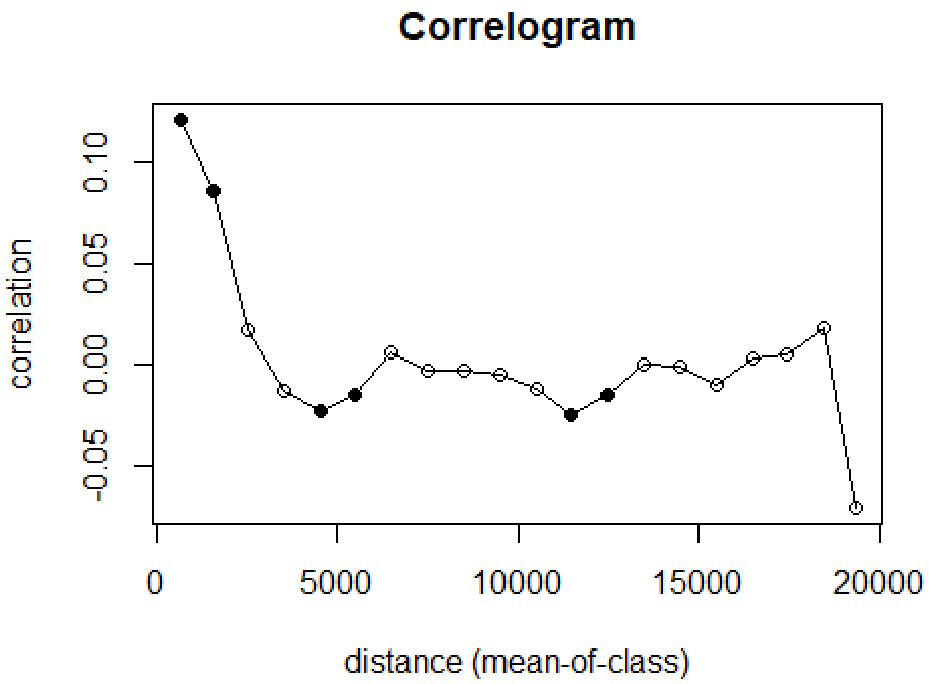
We can use the `knearneigh()` function to identify the n closest neighboring cells for each given cell. In the example below we extract the 2 closest neighbors for each cell (you can change the value to anything you like, e.g. you can extract the 10 closest neighbors instead). Note the argument `longlat = T` which tells the function that the input data is in longitude and latitude format which is the only format where completely accurate distances can be calculated (based on spherical geometry):

```
library(spdep)
nearest_neighbours_2 = knearneigh(cbind(subsample$x, subsample$y),2,longlat =
T)
# This function sorts out the spatial elements
neighbourlist_2_closests = knn2nb(nearest_neighbours_2)
# This illustrates what we have done
plot(neighbourlist_2_closests, cbind(subsample$x, subsample$y))
```

This looks very abstract, but you can roughly see the continents or islands the points in our random subsample were drawn from and the two closest neighbors of each point connected by lines.

Now we will apply the SAR model (`errorsarlm()`), which allows us to incorporate the neighbourhood of points. The formula of the SAR model is a modification of the general linear model (`y = X beta + u, u = lambda W u + e`), adding the error term `u`. The `W` in this error term are our modeled neighbourhood relationships, which are parsed to the function using the `listw()` argument.

```
library(spatialreg)
sar_model = errorsarlm(species_div~temperature, data=subsample,
listw=nb2listw(neighbourlist_2_closests), tol.solve = 1e-12, zero.policy =T)
autocorrelation_sar = correlog(subsample$x, subsample$y, sar_model$residuals,
increment=1000, latlon=T, resamp=100)

## 10  of  100 20  of  100 30  of  100 40  of  100 50  of  100 60  of  100 70
of  100 80  of  100 90  of  100 100  of  100

plot(autocorrelation_sar)
```



In this example, we see improvement even though we still have an issue with autocorrelation.

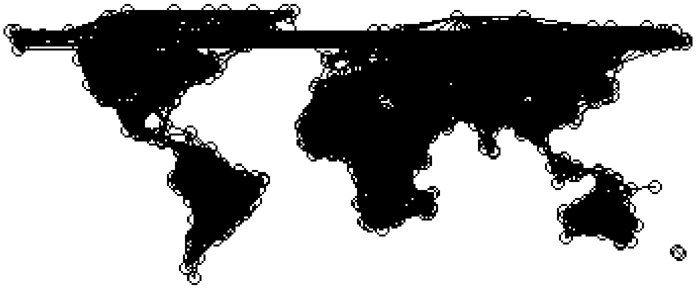
Let’s also check the model summary:

```
summary(sar_model)

##
## Call:errorsarlm(formula = species_div ~ temperature, data = subsample,
##     listw = nb2listw(neighbourlist_2_closests), zero.policy = T,
##     tol.solve = 1e-12)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -2.331928 -0.214757  0.013491  0.201755  2.100142
##
## Type: error
## Coefficients: (asymptotic standard errors)
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.026027   0.070240 -0.3705  0.71097
## temperature  0.100644   0.044077  2.2834  0.02241
##
## Lambda: 0.83099, LR test value: 1516.7, p-value: < 2.22e-16
## Asymptotic standard error: 0.0094063
##     z-value: 88.344, p-value: < 2.22e-16
## Wald statistic: 7804.6, p-value: < 2.22e-16
##
```

```
## Log likelihood: -649.8238 for error model
## ML residual variance (sigma squared): 0.14093, (sigma: 0.3754)
## Number of observations: 1000
## Number of parameters estimated: 4
## AIC: 1307.6, (AIC for lm: 2822.3)
```

However we can improve this model further by testing different definitions of neighborhood. Instead of calculating the distance to the n closest neighbors with the `knearneigh()` function, we can instead use the `dnearneigh()` function, which is based on a distance threshold and extracts all points within that distance. For example the following command extracts all points that are within a distance of 1500km of each point:

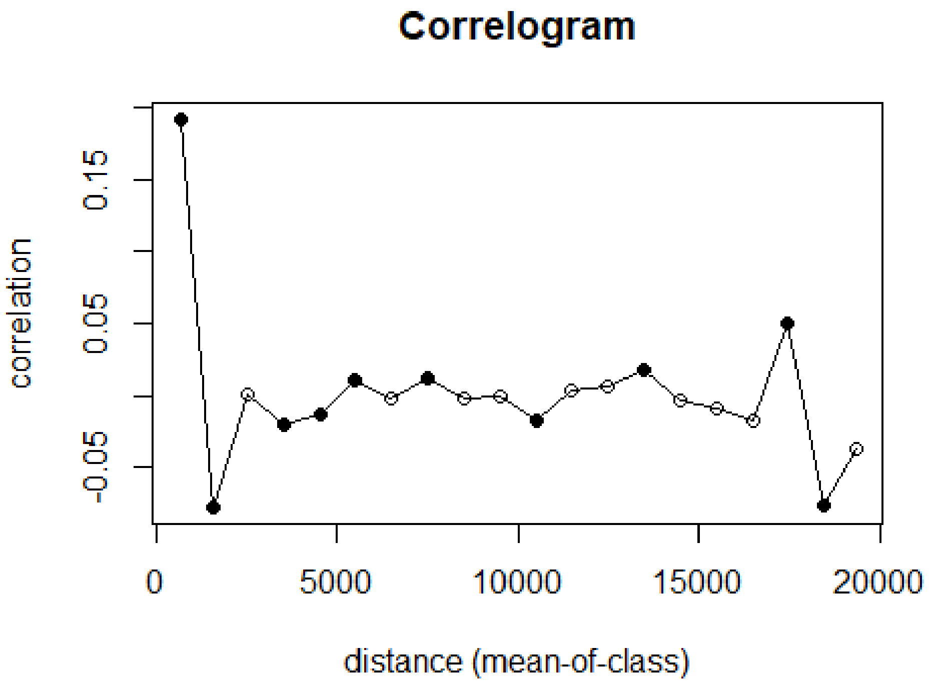
```
neighbours_1500km = dnearneigh(cbind(subsample$x,subsample$y), 0,1500,
longlat = T)
# This illustrates what we have done
plot(neighbours_1500km, cbind(subsample$x, subsample$y))
```



You see that the neighborhoods look very different, since only points within 1500 km are connected. This also leads to some points having no neighbors at all, since they are too isolated. Also, note that the longer lines going across the whole map are a result of us using lon-lat input data (remember, the Earth is a globe).

Let’s see how this neighborhood definition performs compared to the n closest model from before. Note: The `zero.policy =T` argument in the SAR model and in the `nb2listw()` function is necessary to account for potential points without neighbours (which particular would be relevant if we were using a smaller neighborhood e.g. only including cells within 200 km)

```
sar_model = errorsarlm(species_div~temperature, data=subsample,
listw=nb2listw(neighbours_1500km,zero.policy =T), tol.solve = 1e-12,
zero.policy =T)
autocorrelation_sar = correlog(subsample$x, subsample$y, sar_model$residuals,
increment=1000, latlon=T, resamp=100)

## 10  of  100 20  of  100 30  of  100 40  of  100 50  of  100 60  of  100 70
of  100 80  of  100 90  of  100 100  of  100

plot(autocorrelation_sar)
```



The autocorrelation between the closest points look worse here than in the other neighborhood but substantially better than the linear model.

Let's also check the model summary:

```
summary(sar_model)

##
## Call:errorsarlm(formula = species_div ~ temperature, data = subsample,
##     listw = nb2listw(neighbours_1500km, zero.policy = T), zero.policy = T,
##     tol.solve = 1e-12)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -2.456509 -0.248799  0.017434  0.300164  2.639952
##
## Type: error
## Coefficients: (asymptotic standard errors)
##              Estimate Std. Error z value  Pr(>|z|)
## (Intercept) -0.566495   0.393513 -1.4396 0.1499847
## temperature  0.202513   0.056307  3.5966 0.0003224
##
## Lambda: 0.95838, LR test value: 1220.4, p-value: < 2.22e-16
## Asymptotic standard error: 0.011479
##     z-value: 83.491, p-value: < 2.22e-16
## Wald statistic: 6970.8, p-value: < 2.22e-16
##
## Log likelihood: -797.9413 for error model
## ML residual variance (sigma squared): 0.26818, (sigma: 0.51786)
## Number of observations: 1000
## Number of parameters estimated: 4
## AIC: 1603.9, (AIC for lm: 2822.3)
```

Generally one should test a bunch of different configurations of the `knearneigh()` and `dnearneigh()` neighbourhoods, using different numbers of neighbours and different distances respectively. After testing different configurations the best model can be selected using a model selection criterion such as AIC. For that purpose we first define a function that runs through all our desired model configurations. Note: one can also try different values for the `style=` argument, common options are `B`,`C`,`U`,`S`, or `W`. In terms of identifying the effect of predictors `B`,`C`,`U` are identical and only one of these will be used in the examples below:

```
Neighborhood_generator=function(COOR) {
models<<-list(
  nb2listw(knn2nb(knearneigh(COOR,1, longlat = T)),style="W",zero.policy =T),
  nb2listw(knn2nb(knearneigh(COOR,2, longlat = T)),style="W",zero.policy =T),
  nb2listw(knn2nb(knearneigh(COOR,3, longlat = T)),style="W",zero.policy =T),
  nb2listw(knn2nb(knearneigh(COOR,4, longlat = T)),style="W",zero.policy =T),
  nb2listw(knn2nb(knearneigh(COOR,5, longlat = T)),style="W",zero.policy =T),
  nb2listw(knn2nb(knearneigh(COOR,6, longlat = T)),style="W",zero.policy =T),
  nb2listw(knn2nb(knearneigh(COOR,7, longlat = T)),style="W",zero.policy =T),
  nb2listw(knn2nb(knearneigh(COOR,8, longlat = T)),style="W",zero.policy =T),
  nb2listw(knn2nb(knearneigh(COOR,9, longlat = T)),style="W",zero.policy =T),
  nb2listw(knn2nb(knearneigh(COOR,10, longlat = T)),style="W",zero.policy
=T),
  nb2listw(dnearneigh(COOR, 0,250, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,500, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,750, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,1000, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,1250, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,1500, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,2000, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,2500, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,3000, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,3500, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,4000, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,250, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,500, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,750, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,1000, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,1250, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,1500, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,2000, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,2500, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,3000, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,3500, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,4000, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,250, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,500, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,750, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,1000, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,1250, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,1500, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,2000, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,2500, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,3000, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,3500, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,4000, longlat = T),style="S",zero.policy =T)
)
}
```

Now we apply that list of models to our coordinates of the random subsample of our data:

```
neighbourhood_models = Neighborhood_generator(cbind(subsample$x,subsample$y))
```

Calculate the AICc score for each neighborhood model using the `AICc()` function. This is a version of the regular AIC criterion that additionally corrects for small sample size:

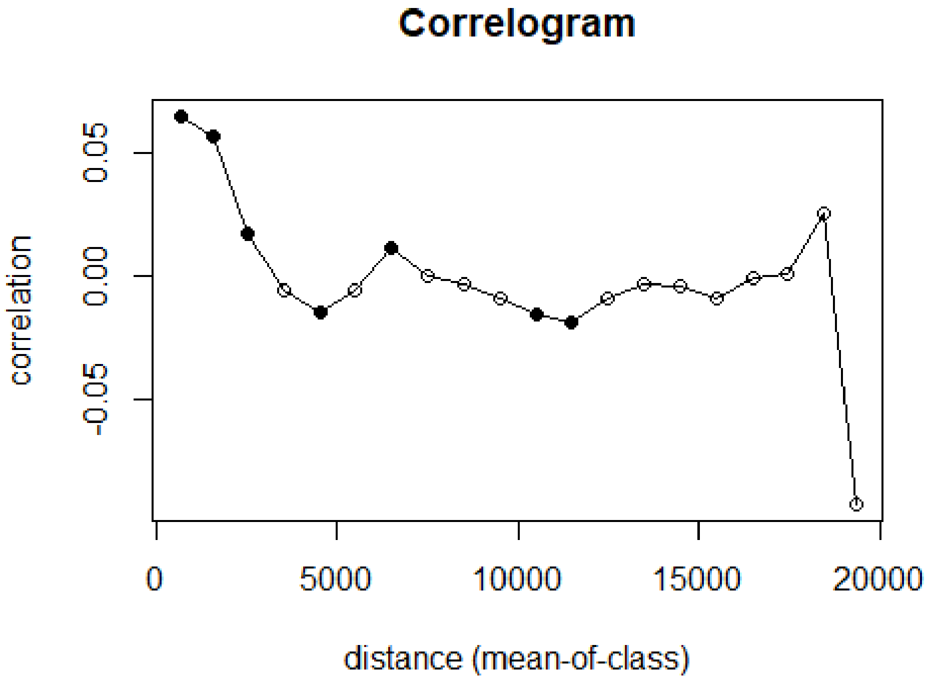
```
library(wiqid)

AIC_LIST=numeric(length(neighbourhood_models))
for (i in 1:length(neighbourhood_models)) {
    AIC_LIST[i]=AICc(errorsarlm(species_div~temperature,
data=subsample,listw=neighbourhood_models[[i]], tol.solve = 1e-12,
zero.policy =T))
}
```

Select the model with the lowest AIC score as the best model:

```
index_best_model = which(AIC_LIST==min(AIC_LIST))
best_neighbour_model = neighbourhood_models[index_best_model]
```

Plot the correlogram:

```
sar_model = errorsarlm(species_div~temperature, data=subsample,
listw=best_neighbour_model[[1]], tol.solve = 1e-12, zero.policy =T)
autocorrelation_sar = correlog(subsample$x, subsample$y, sar_model$residuals,
increment=1000, latlon=T, resamp=100)

## 10  of  100 20  of  100 30  of  100 40  of  100 50  of  100 60  of  100 70
of  100 80  of  100 90  of  100 100  of  100

plot(autocorrelation_sar)
```



This looks better, (even though there is still significant autocorrelation). It seems trivial to see that this figure is better than the linear model but perhaps not that it is better than some of the other neighborhoods. Many researchers prefer to judge neighborhoods by looking at correlograms like this but I find them difficult to compare which is why I would advise to select then based on AICc as we have do in this tutorial.

Let's check the model summary:

```
summary(sar_model)

##
## Call:errorsarlm(formula = species_div ~ temperature, data = subsample,
##     listw = best_neighbour_model[[1]], zero.policy = T, tol.solve = 1e-12)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -2.478424 -0.220046  0.012478  0.203561  1.945463
```

```
##
## Type: error
## Coefficients: (asymptotic standard errors)
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.046210   0.085088 -0.5431  0.58707
## temperature  0.113993   0.047125  2.4190  0.01556
##
## Lambda: 0.86002, LR test value: 1587.3, p-value: < 2.22e-16
## Asymptotic standard error: 0.0098752
##     z-value: 87.09, p-value: < 2.22e-16
## Wald statistic: 7584.6, p-value: < 2.22e-16
##
## Log likelihood: -614.489 for error model
## ML residual variance (sigma squared): 0.14185, (sigma: 0.37663)
## Number of observations: 1000
## Number of parameters estimated: 4
## AIC: 1237, (AIC for lm: 2822.3)
```

Determine model fit

Besides checking the size of the effect of the predictor and the significance of it, you can get a measure of model fit by investigates R2 values. Technically we will use normal R^2 for the linear model and Nagelkerke pseudo R^2 for the SAR models. The math behind the differences is pretty complex but for the curious it is discussed in the wikipedia article about Pseudo R2 values

```
summary(lm_model)

##
## Call:
## lm(formula = species_div ~ temperature, data = subsample)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.4166 -0.5024  0.1893  0.6004  3.1443
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.01629    0.03132  -0.520    0.603
## temperature  0.17275    0.03118   5.541 3.85e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9903 on 998 degrees of freedom
## Multiple R-squared:  0.02985,    Adjusted R-squared:  0.02887
## F-statistic:  30.7 on 1 and 998 DF,  p-value: 3.849e-08

summary(sar_model, Nagelkerke = TRUE)

##
## Call:errorsarlm(formula = species_div ~ temperature, data = subsample,
##     listw = best_neighbour_model[[1]], zero.policy = T, tol.solve = 1e-12)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -2.478424 -0.220046  0.012478  0.203561  1.945463
##
## Type: error
## Coefficients: (asymptotic standard errors)
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.046210   0.085088 -0.5431  0.58707
## temperature  0.113993   0.047125  2.4190  0.01556
##
## Lambda: 0.86002, LR test value: 1587.3, p-value: < 2.22e-16
## Asymptotic standard error: 0.0098752
##     z-value: 87.09, p-value: < 2.22e-16
## Wald statistic: 7584.6, p-value: < 2.22e-16
##
## Log likelihood: -614.489 for error model
## ML residual variance (sigma squared): 0.14185, (sigma: 0.37663)
## Nagelkerke pseudo-R-squared: 0.80163
## Number of observations: 1000
## Number of parameters estimated: 4
## AIC: 1237, (AIC for lm: 2822.3)
```

The R-square value for the SAR model is much higher, which is expected because it contains the spatial information as well and we saw that the data has very strong autocorrelation. Therefore, this value does not tell us much about the predictive power of our actual predictor variable, but only of the whole model. To get an idea of the R-square for only the predictor we can create another model solely containing the neighborhood:

```
sar_model_NULL = errorsarlm(species_div~1, data=subsample,
listw=best_neighbour_model[[1]], tol.solve = 1e-12, zero.policy =T)
summary(sar_model_NULL, Nagelkerke = TRUE)
```

```
##
## Call:
## errorsarlm(formula = species_div ~ 1, data = subsample, listw =
best_neighbour_model[[1]],
##     zero.policy = T, tol.solve = 1e-12)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -2.4535504 -0.2148066  0.0044916  0.2062004  2.0344832
##
## Type: error
## Coefficients: (asymptotic standard errors)
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.046840   0.085801 -0.5459   0.5851
##
## Lambda: 0.86087, LR test value: 1611.8, p-value: < 2.22e-16
## Asymptotic standard error: 0.0098223
##     z-value: 87.644, p-value: < 2.22e-16
## Wald statistic: 7681.6, p-value: < 2.22e-16
##
## Log likelihood: -617.4025 for error model
## ML residual variance (sigma squared): 0.1425, (sigma: 0.37749)
## Nagelkerke pseudo-R-squared: 0.80047
## Number of observations: 1000
## Number of parameters estimated: 3
## AIC: 1240.8, (AIC for lm: 2850.6)
```

We can see that we get a very marginal increase r-squared by adding the predictor. In my case I had 0.80163 for the full model and 0.80047 for the null model. This means that we are only explaining 0.11% more of the data by including the predictor. Your data will be marginally different if you used another seed or no seed at all.

We could also compare models based on AICs

```
AICc(sar_model)

## [1] 1237.018

AICc(sar_model_NULL)

## [1] 1240.829

AICc(lm_model)

## [1] 2822.341
```

This again shows that a spatial model including the predictor is only marginally better than a spatial model without it but are substantially better than a normal linear model.

Finally, a very brief summary of what we found. Our results are telling us that if temperature is increased by one standard deviation, carnivore diversity will on average increase by 0.113993 standard deviations after controlling for spatial autocorrelation. The results without controlling for autocorrelation was 0.17275 instead. In this particular case the effect size was marginally lower when controlling for autocorrelation. This is not always the case. Significance will always be lower when controlling for autocorrelation but effect size can be changes in any random direction.
In other words not controlling for spatial autocorrelation is not biasing your results but it is overestimating your confidence in the conclusions.

Further steps (for the faster students):

- Add multiple predictors (at least 2) in one joined analysis. The general syntax to include multiple predictors into the model is e.g. `glm(species_div~predictor1+predictor2,data=subsample)`, or if you want to include interactions between predictors `glm(species_div~predictor1*predictor2,data=subsample)` (equivalent syntax for the SAR model).

-Try removing all cells with 0, 1 or 2 species from the analyses. These cells are often areas which can sustain more species climatically but are isolated islands with no or very few native species due to isolation rather than climatic reasons.