Stat 413 Estimating the Area of Nations Project

Milagros N. Cortez, Akemi Izuko, Tyler Eggen

05/04/2023

Stat 413 Project

Introduction

In this project we design an algorithm to estimate the area of any country on Earth through Monte-Carlo simulation. The algorithm achieves this by stimulating latitude and longitude points from the surface of the Earth and using a lookup table to match it to the country. Given a country's name as input, the algorithm will report the estimated area of that country, along with a 90% confidence bound around that estimate.

As part of the output, we provide an estimate of the area of the country $[km^2]$, its variance $[km^4]$, and a 90% confidence bound for our estimate of the land area.

Geographic Coordinate Systems

Geographic coordinate systems identify a location on the surface of Earth using two coordinate values - Longitude and Latitude. Longitude is the angular distance from East to West of the Prime meridian plane, which ranges from 0^o to 180^oE or 180^oW. Latitude is the angular distance North or south of the Equatorial plane, which ranges from 0^o to 90^oN or 90^oS at the poles. As a coordinate reference system (CRS) is absolute for which one pair of latitude and longitude values correspond to one place on Earth which is represented as a spherical or ellipsoidal surface[2].

We also define the following which will be used to describe our process:

- Point: pair of coordinates (latitude, longitude) that represent the location of a place on Earth
- Line: path between two or more ordered points. Endpoints of a line are called vertices.
- Polygon: Area bounded by vertices

Monte Carlo Simulation

For this project, we use Monte Carlo simulation sampling with uniform distributions to estimate the area of a country. To achieve this we generate two vectors uniformly at random within a the bounded area of latitude and longitude that captures the country by filling the entries with x iid $Uniform[min(l), max(l)]$ with l being latitude for the first vector and longitude for the second vector. We then calculate the area in km^2 for the bounded area that captures the country, denoted as λ and use the vectors to get the following estimates:

$mean : \hat{\mu} = \lambda \sum_{i=1}^{n} \phi(x_i)$

and

$variance : \hat{\sigma}^2 = \lambda^2 \frac{\hat{\mu}(1-\hat{\mu})}{n-1}$

where $\phi(x) = 1$ if the point falls in the country, and $= 0$ else.

Sinusoidal Projection

Sinusoidal projection, also known as the Sanson-Flamsteed and Mercator-Sanson projection, is a pseudo-cylindrical equal-area projection which has the poles of Earth as points with distorted meridians and continents. This projection is used in the project to convert points with latitude and longitude values into distances in the map which can be used to calculate the area of land that captures the country of interest. This area is used to obtain the estimated area of the country as well as its variance and 90% confidence bounds[3].

To apply the sinusoidal projection, we used the functions:

$x = (\phi)cos(\theta)$

$y = \theta \times (\frac{\theta\pi}{180})$

Where ϕ is the longitude and θ is the latitude of a point.

Algorithm Process

Since completely random sampling across the Earth would require a computationally infeasible number of samples to get a single point in some countries, we split our algorithm procedure into a (1) step that finds a boundary that captures the country, and (2) a sampling stage that estimates the area of the country under the bounded area using Monte Carlo estimation. The first step is split into four parts that refine the bounds of the area that capture the country, called searches, while the second step is a single part called a phase.

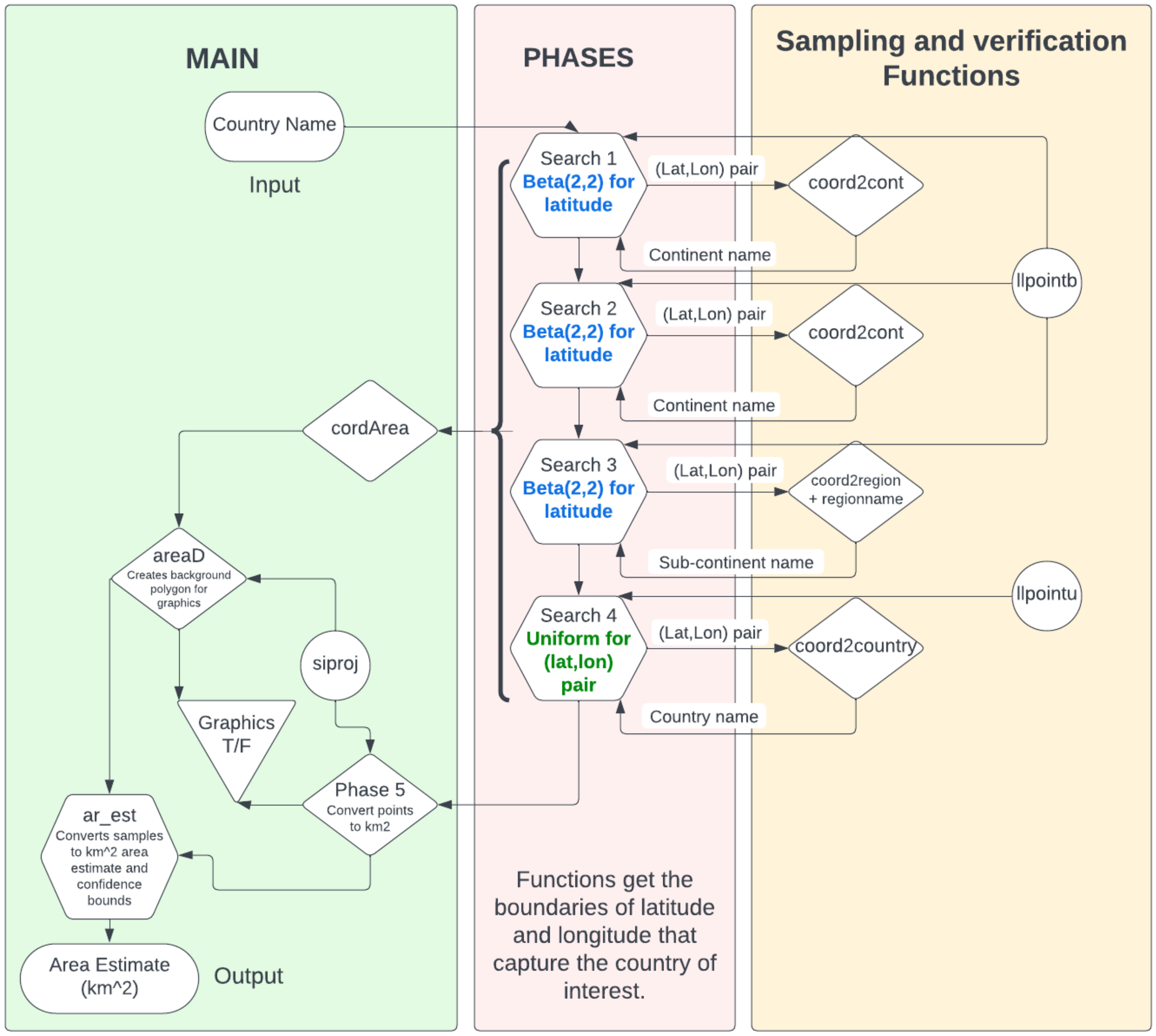
1. First, under search 1, we sample $n = 100$ points of the entire Earth to get an estimate for the bounds that capture the continent containing the target country. We use a scaled Beta(2,2) distribution to sample the latitude from $\mathrm{Beta}(2,2) \cdot 180 - 90$, and a $Uniform(-180, 180)$ to sample the longitude of each point.
2. Search 2 continues attempting to define clearer bounds for the continent containing the country of interest. Using the prior estimate from search 1, we only sample within the latitude and longitude bounds of that estimate. Much like the first step, we sample $n = 100$ points under the same conditions as search 1.

Note: This step importantly differentiates between North and South America, which we found narrowed the region down significantly, enabling us to sample smaller countries like Jamaica.

3. In search 3, we attempt to find bounds for the sub-continent containing our country. An example region is West Africa. Just like search 1 and 2, we sample only within the continental bounds estimated in the previous step, with about 20 degrees of added margin on both sides. Search 3 still samples from a scaled $Beta(2,2)$ on the latitude. Unlike the previous steps, the R code increases its sampling to $n = 300$ points in this stage.
4. Finally for search 4, we search for the bounds in which the country itself is contained, sampling only in this sub-continent. Unlike the previous searches, we use a uniform distribution to sample both the latitude and longitude values, with about 5 degrees of added margin on both sides. Here we sample using $n = 500$ points in the R code.
5. For phase 5 we use sampling from the bounds found in search 4 to get the points used for the area estimate of the country of interest. In the R code, this step uses $n = 3000$ samples to get a much more accurate estimate, a noticeable increase from the previous stages. We continue to sample from a scaled uniform for this stage, just like in search 4.

Using the samples from phase 5, we finally estimate the area of the country in km^2, and return it along with its variance and a 90% confidence bound from the main function.

Flowchart of the R code


MAIN
PHASES
Sampling and verification Functions
Country Name
Input
Search 1 Beta(2,2) for latitude
(Lat,Lon) pair
coord2cont
Continent name
llpointb
Search 2 Beta(2,2) for latitude
(Lat,Lon) pair
coord2cont
Continent name
cordArea
Search 3 Beta(2,2) for latitude
(Lat,Lon) pair
coord2region + regionname
Sub-continent name
areaD Creates background polygon for graphics
llpointu
Search 4 Uniform for (lat,lon) pair
(Lat,Lon) pair
coord2country
Country name
siproj
Graphics T/F
Phase 5 Convert points to km2
ar_est Converts samples to km^2 area estimate and confidence bounds
Area Estimate (km^2)
Output
Functions get the boundaries of latitude and longitude that capture the country of interest.

We can see from the flowchart above how the algorithm is applied to the R code. In the code we have a main function that receives the input and returns the final estimation, variance, confidence bounds and graphics as output, four functions that search the bounds that capture the country of interest using functions that are used to check if the coordinates fall within a country, sub-continent, and continent. These four functions also work along with sampling functions that return latitude and longitude values sampled though uniform distributions or latitude values sampled from a scaled $Beta(2, 2)$ and longitude values sampled from a uniform distribution.

True Values

To test the algorithm we created a code in R which we will test on countries of big, medium, and small area. The countries along with their area in km^2 can be found in the table below:

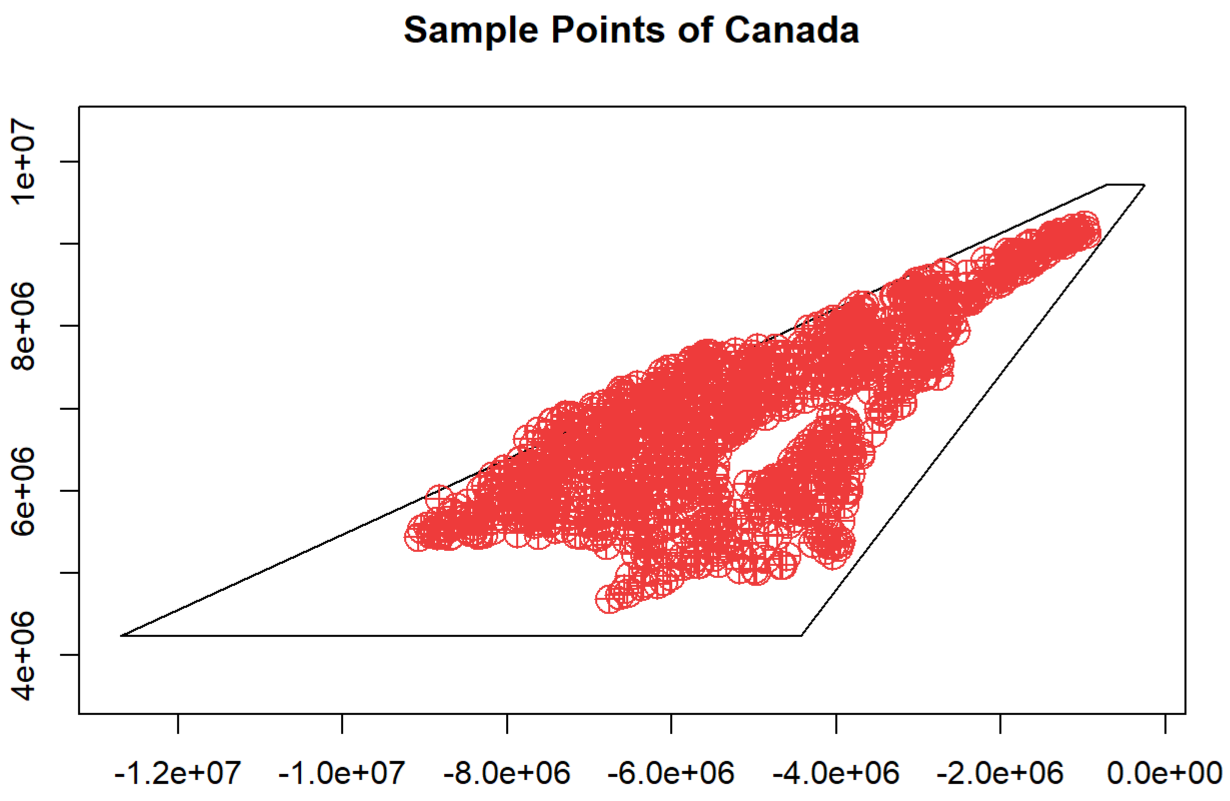
```
# Table of the true area of countries tested
inf = matrix(c(9984670, 9562910, 1141748, 1001450, 110860, 103000, 10452, 10991, 748.6), ncol =
1, byrow = TRUE)
colnames(inf) = c("Area [km^2]")
rownames(inf) = c("Canada", "China", "Colombia","Egypt", "Cuba", "Iceland", "Lebanon", "Jamaic
a", "Singapore")
info = as.table(inf)
knitr::kable(inf, "simple")
```

	Area [km^2]
Canada	9984670.0
China	9562910.0
Colombia	1141748.0
Egypt	1001450.0
Cuba	110860.0
Iceland	103000.0
Lebanon	10452.0
Jamaica	10991.0
Singapore	748.6

(Areas found using google)
The estimates of each country can be found in the following sections:

Estimating Big Countries

```
canada = countryArea("Canada", visualize = T)
```

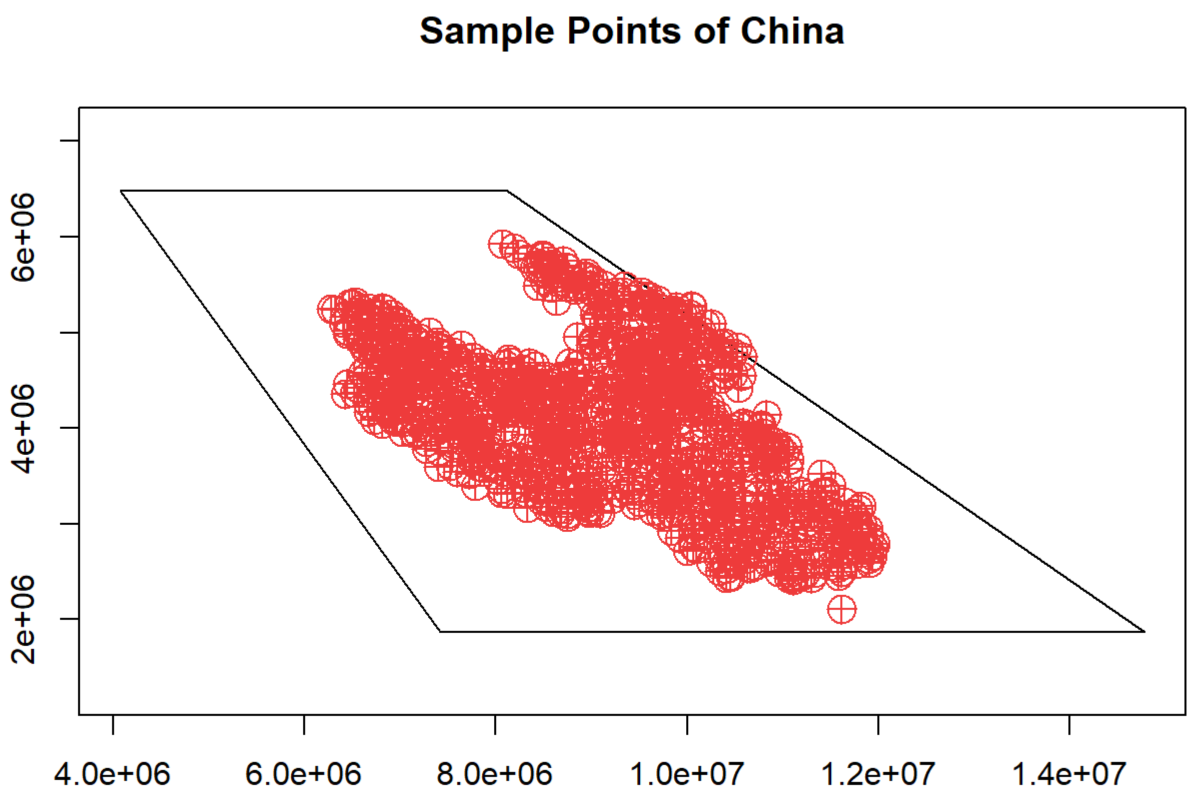


```
canada
```

```
## $mean
## 8694650 [km^2]
##
## $variance
## 44234436711 [km^4]
##
## $ci
## Units: [km^2]
## [1] 8348704 9040595
```

We can see from the output above that the estimated mean area of $8,694,650\ km^2$ and the 90% confidence bound from $8,348,704\ km^2$ to $9,040,595\ km^2$ are lower than the true area of Canada ($9,984,670\ km^2$). Therefore the algorithm underestimates the area of this country.

```
china = countryArea("China", visualize = T)
```



```
china
```

```
## $mean
## 8642355 [km^2]
##
## $variance
## 5.0871e+10 [km^4]
##
## $ci
## Units: [km^2]
## [1] 8271365 9013345
```

We can see from the output above that the estimated mean area of $8,642,355\ km^2$ and the 90% confidence bound from $8,271,365\ km^2$ to $9,013,345\ km^2$ are lower than the true area of China $(9,562,910\ km^2)$. Therefore the algorithm also underestimates the area of this country.

We proceed to do a two sample t-test at the 10% level to see if we can detect a statistically significant difference in the land areas of Canada and China. We set the null and alternative hypothesis to be:

$$H_0: \quad \mu_{Canada} = \mu_{China}; H_a: \quad \mu_{Canada} \neq \mu_{China}$$

The test statistic and p-value of the test can be found below:

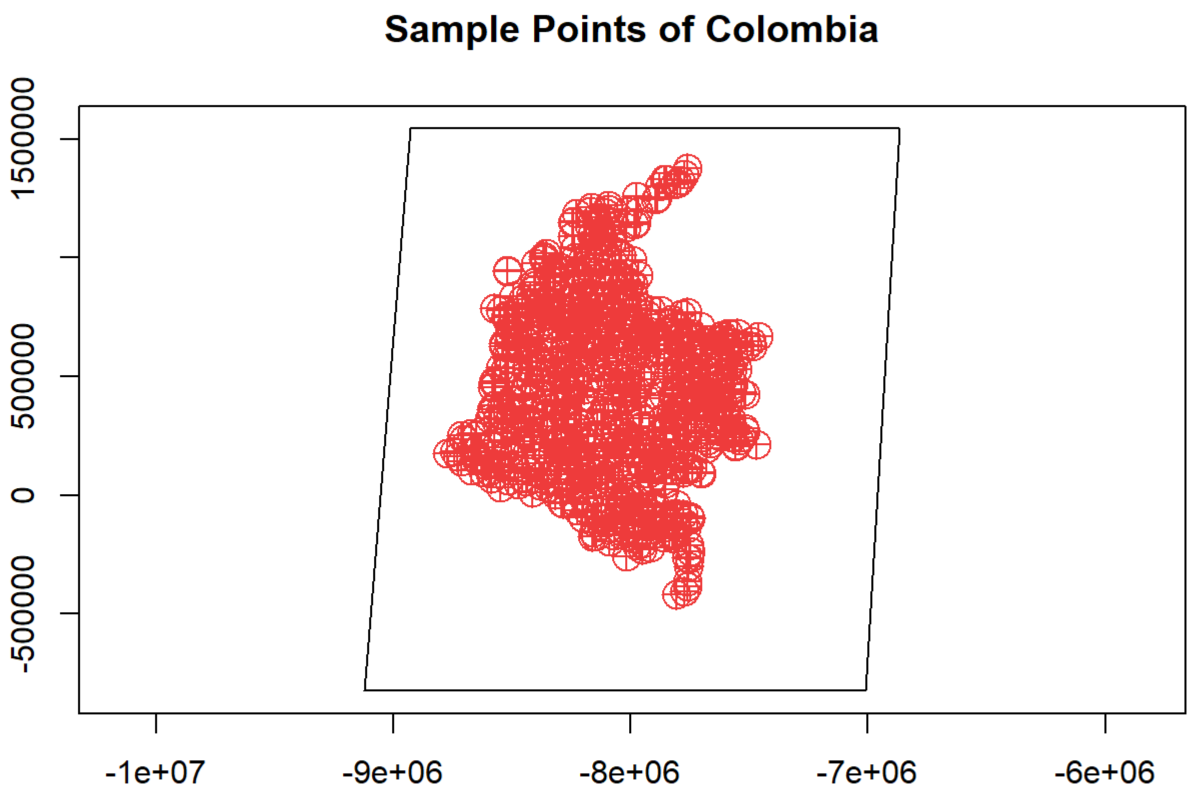
```
tcomparison(canada, china)
```

```
## $tstat
## 9.287877 [1]
##
## $pval
## 2.934089e-20 [1]
```

We can see from the output above that the calculated $p - value < 0.1 = \alpha$ for which we can reject the null hypothesis as the estimated land areas of Canada and China are statistically significantly different.

Estimating Medium Size Countries

```
colombia = countryArea("Colombia", visualize = T)
```

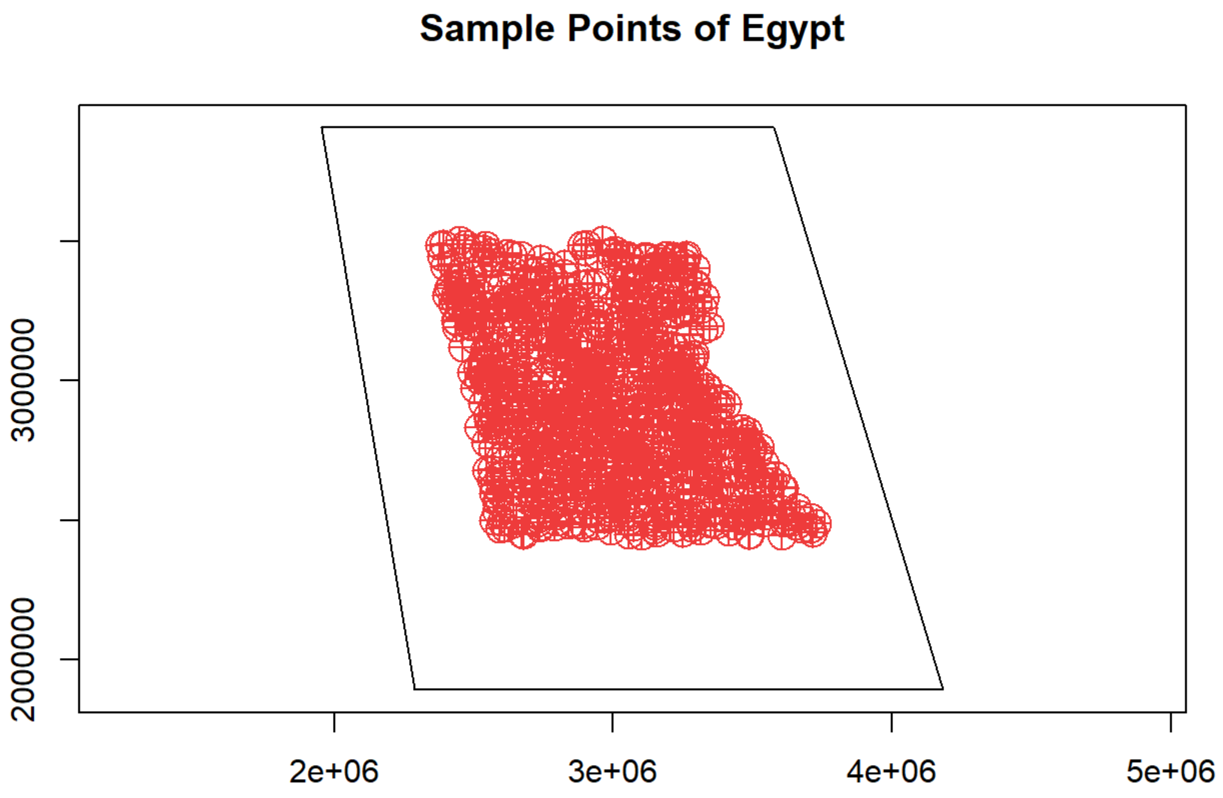


```
colombia
```

```
## $mean
## 1152723 [km^2]
##
## $variance
## 1455804664 [km^4]
##
## $ci
## Units: [km^2]
## [1] 1089963 1215482
```

We can see from the output above that the estimated mean area of $1,152,723\ km^2$ is close to the true area of Colombia ($1,141,748\ km^2$), and the 90% confidence bound from $1,089,963\ km^2$ to $1,215,482\ km^2$ captures the true area. Therefore the algorithm closely estimates the area of this country.

```
egypt = countryArea("Egypt", visualize = T)
```



```
egypt
```

```
## $mean
## 971899.6 [km^2]
##
## $variance
## 834550133 [km^4]
##
## $ci
## Units: [km^2]
## [1]  924382.1 1019417.1
```

We can see from the output above that the estimated mean area of $971,899.6\ km^2$ is slightly smaller but close to the true area of Egypt ($1,001,1450.0\ km^2$), and the 90% confidence bound from $924,382.1\ km^2$ to $1,019,417.1\ km^2$ captures the true area. Therefore the algorithm closely estimates the area of this country.

We proceed to do a two sample t-test at the 10% level to see if we can detect a statistically significant difference in the land areas of Colombia and Egypt. We set the null and alternative hypothesis to be:

$$H_0: \quad \mu_{Colombia} = \mu_{Egypt}; H_a: \quad \mu_{Colombia} \neq \mu_{Egypt}$$

The test statistic and p-value of the test can be found below:

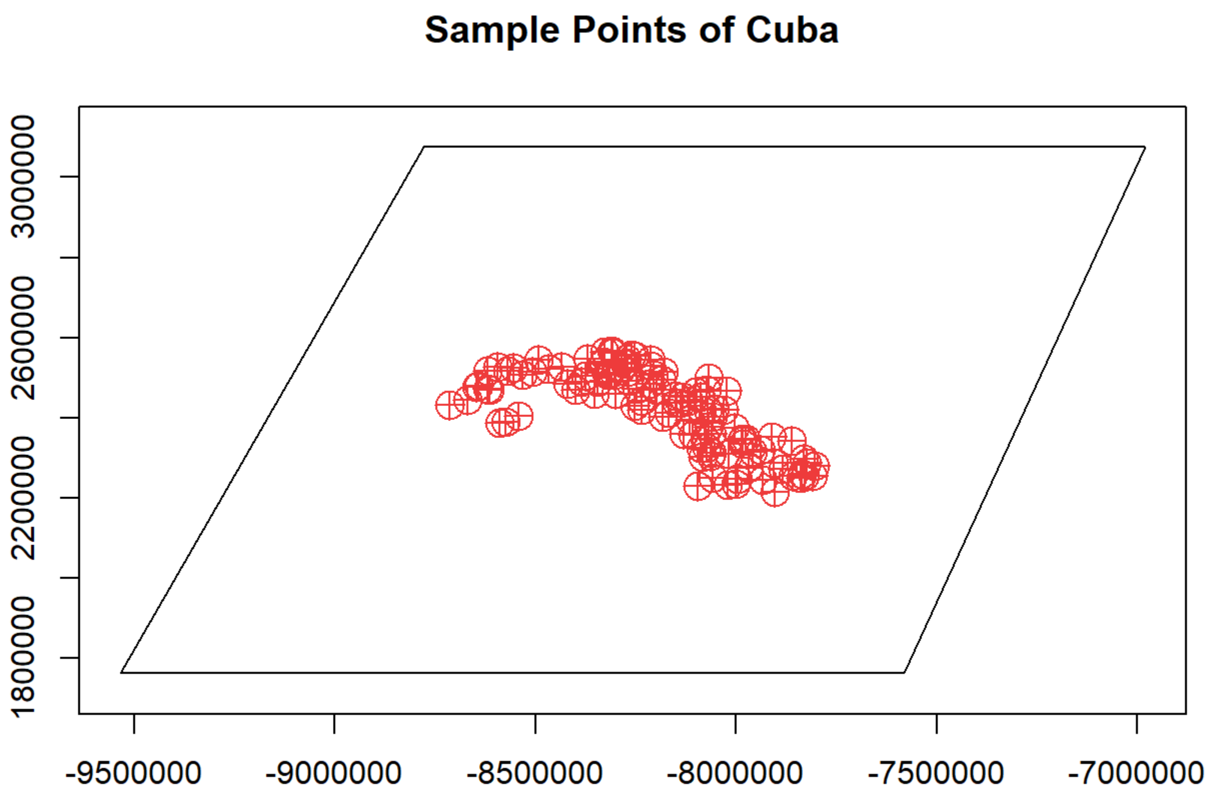
```
tcomparison(colombia, egypt)
```

```
## $tstat
## 206.949 [1]
##
## $pval
## 0 [1]
```

We can see from the output above that the calculated $p-value < 0.1 = \alpha$ for which we can reject the null hypothesis as the estimated land areas of Colombia and Egypt are statistically significantly different.

Estimating Small Countries

```
cuba = countryArea("Cuba", visualize = T)
```

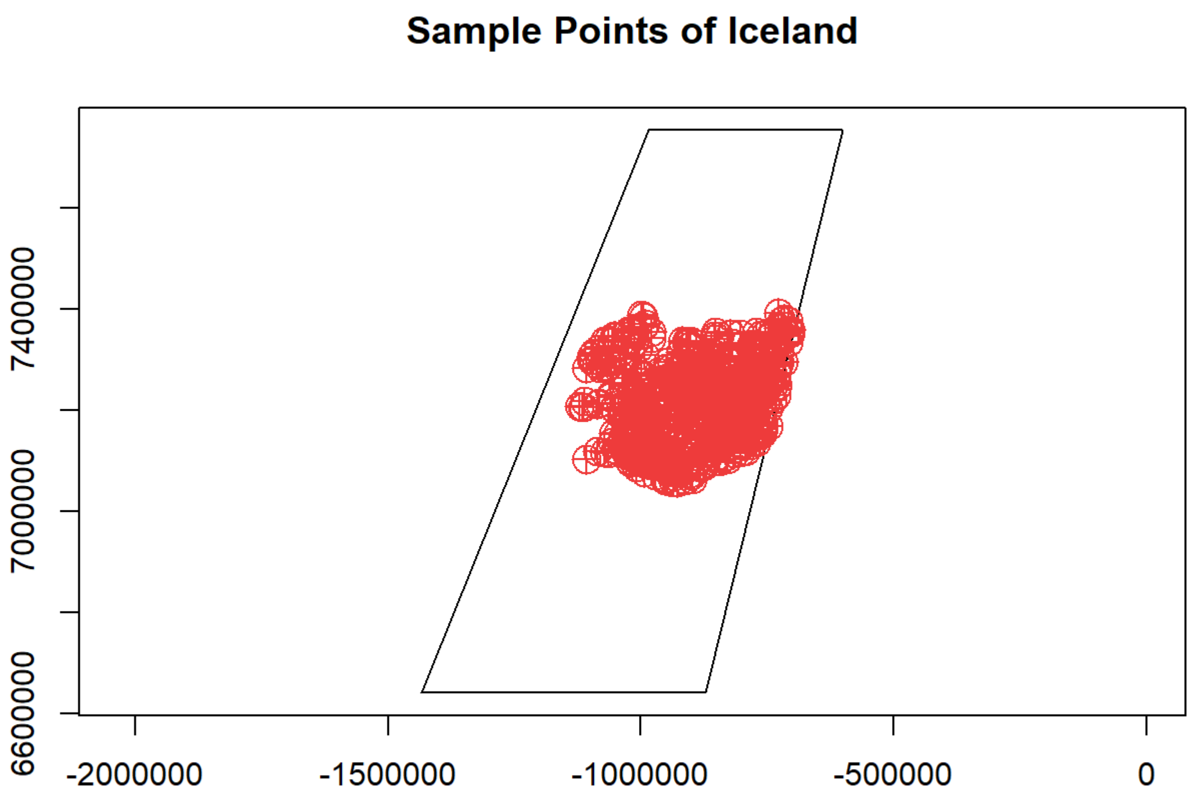


```
cuba
```

```
## $mean
## 91916.87 [km^2]
##
## $variance
## 72642899 [km^4]
##
## $ci
## Units: [km^2]
## [1]  77897.65 105936.09
```

We can see from the output above that the estimated mean area of $91,916.87\ km^2$ and the 90% confidence bound from $77,897.65\ km^2$ to $105,936.09\ km^2$ are lower than the true area of Cuba ($110,860\ km^2$). Therefore the algorithm underestimates the area of this country.

```
iceland = countryArea("Iceland", visualize = T)
```



```
iceland
```

```
## $mean
## 88166.25 [km^2]
##
## $variance
## 12867046 [km^4]
##
## $ci
## Units: [km^2]
## [1] 82266.05 94066.45
```

We can see from the output above that the estimated mean area of $88,166.25\ km^2$ and the 90% confidence bound from $82,266.05\ km^2$ to $94,066.45\ km^2$ are lower than the true area of Iceland ($103,000\ km^2$). Therefore the algorithm also underestimates the area of this country.

We proceed to do a two sample t-test at the 10% level to see if we can detect a statistically significant difference in the land areas of Cuba and Iceland. We set the null and alternative hypothesis to be:

$H_0 : \mu_{Cuba} = \mu_{Iceland};$

$H_a : \mu_{Cuba} \neq \mu_{Iceland}$

The test statistic and p-value of the test can be found below:

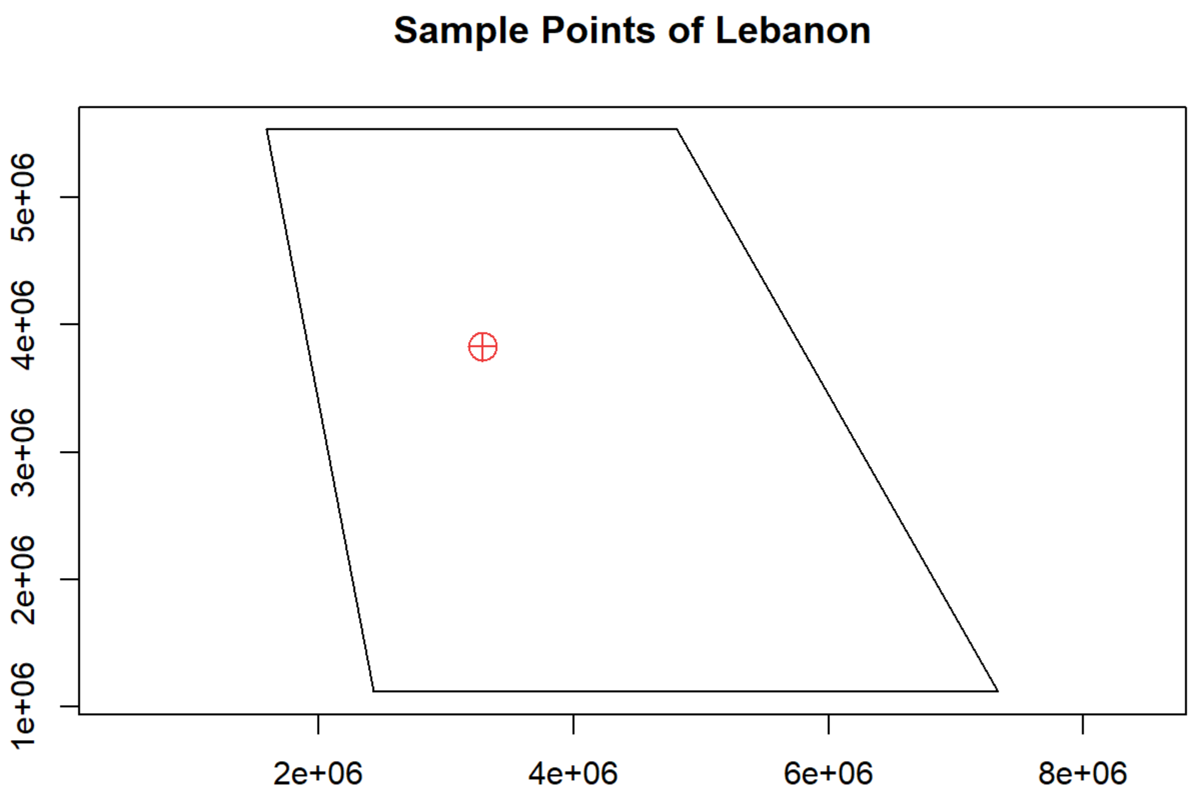
```
tcomparison(cuba, iceland)
```

```
## $tstat
## 22.21546 [1]
##
## $pval
## 2.374777e-101 [1]
```

We can see from the output above that the calculated $p - value < 0.1 = \alpha$ for which we can reject the null hypothesis as the estimated land areas of Cuba and Iceland are statistically significantly different.

Bonus Countries

```
lebanon = countryArea("Lebanon", visualize = T)
```

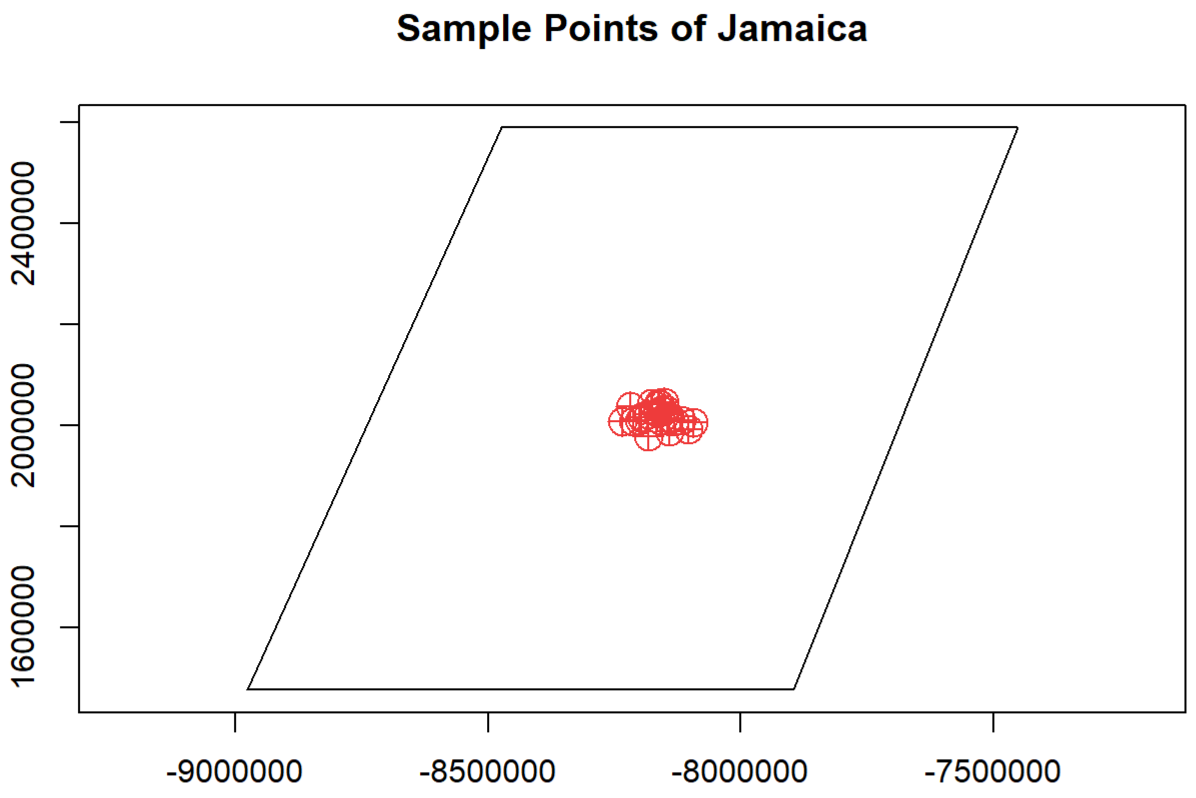


```
lebanon
```

```
## $mean
## 5973.558 [km^2]
##
## $variance
## 35683390 [km^4]
##
## $ci
## Units: [km^2]
## [1] -3852.07 15799.19
```

We can see from the output above that the estimated mean area of $5,973.56\ km^2$ is almost twice as low from the true area of Lebanon ($10,452\ km^2$), but the 90% confidence bound from $-3,852.07\ km^2$ to $15,799.19\ km^2$ captures the true area.

```
jamaica = countryArea("Jamaica", visualize = T)
```

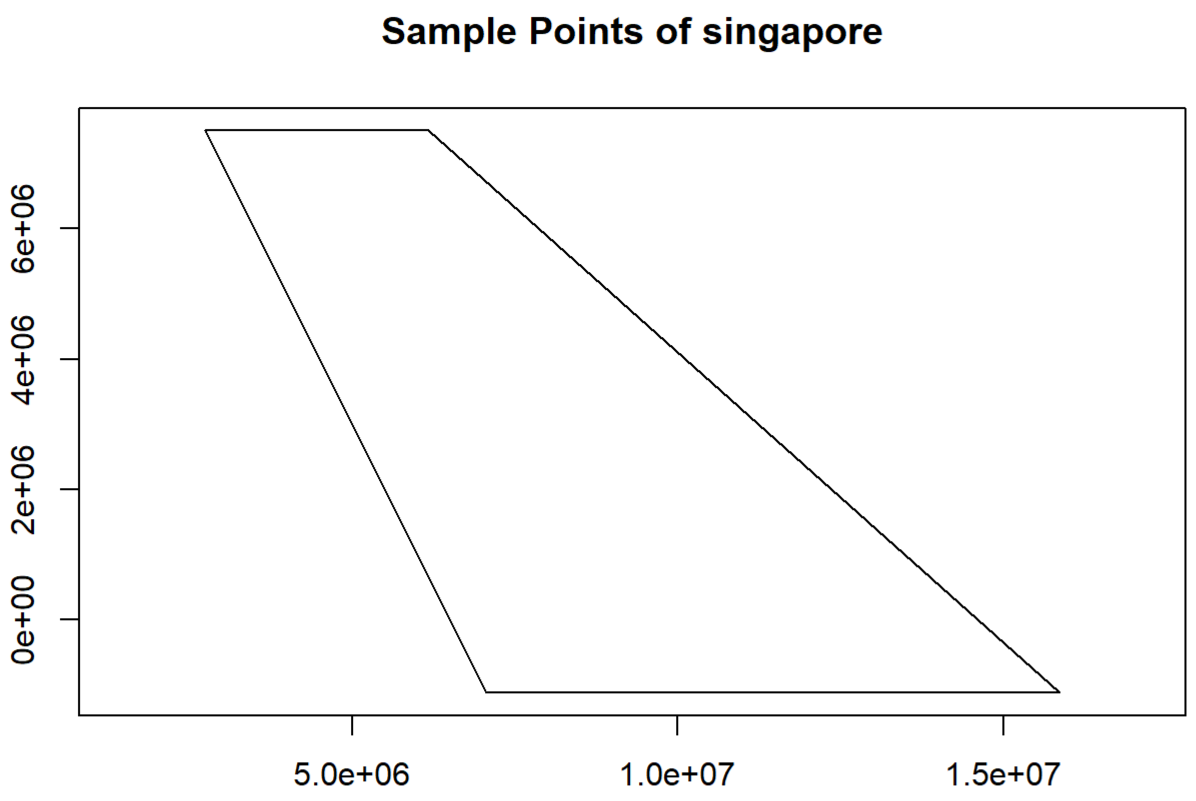


```
jamaica
```

```
## $mean
## 8966.842 [km^2]
##
## $variance
## 3470192 [km^4]
##
## $ci
## Units: [km^2]
## [1]  5902.734 12030.949
```

We can see from the output above that the estimated mean area of $8,966.84\ km^2$ is slightly close to the true area of Jamaica ($10,991\ km^2$), and the 90% confidence bound from $5,902.73\ km^2$ to $12,030.95\ km^2$ captures the true area. We also note that the size of the polygon that captures the country is smaller to the polygon that captures Lebanon which could be a factor as to why this estimate is closer to the thrue value compared to the estimate of Lebanon.

```
singapore = countryArea("singapore", visualize = T)
```



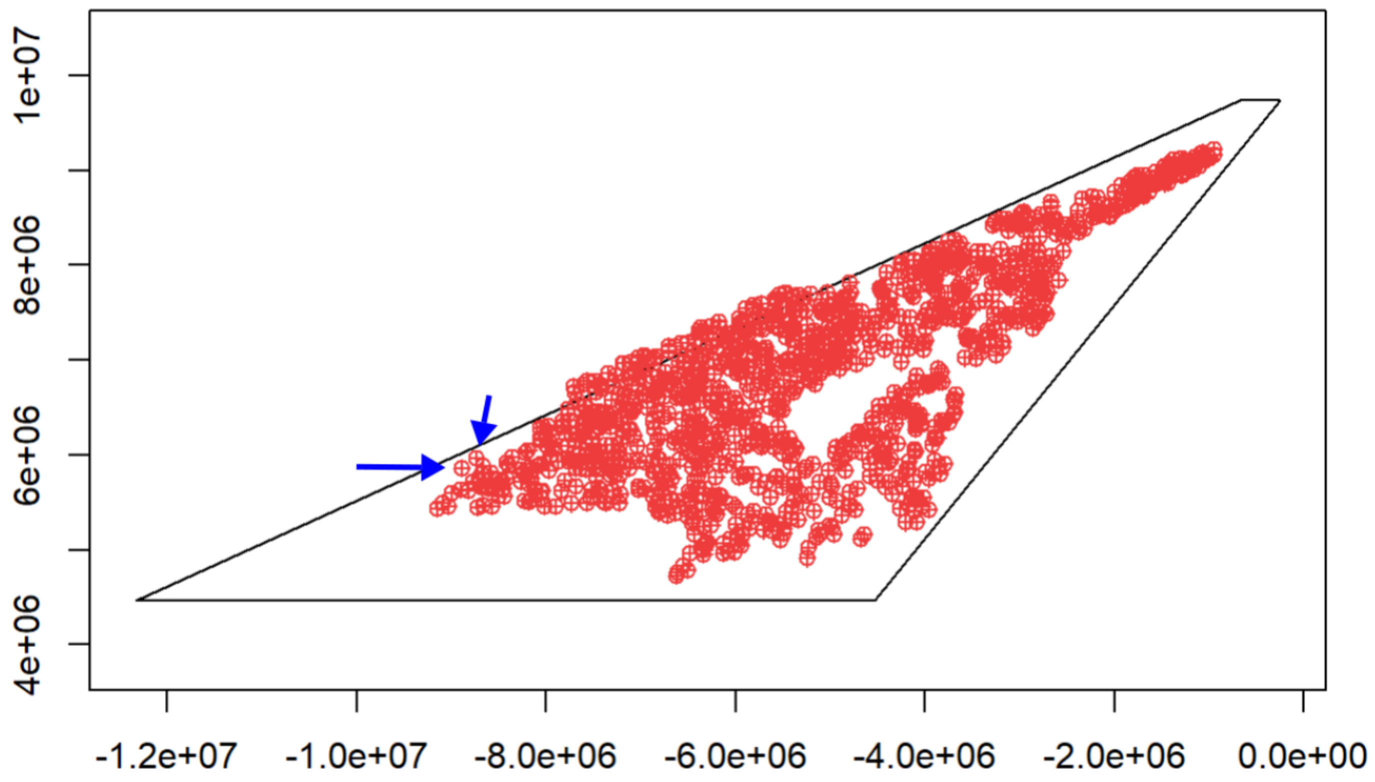
```
singapore
```

```
## $mean
## 0 [km^2]
##
## $variance
## 0 [km^4]
##
## $ci
## Units: [km^2]
## [1] 0 0
```

We can see from the output above that the estimate made by the algorithm was $0\ km^2$ which is not the area of Singapore ($748.6\ km^2$). Looking at the plot above we can see that the polygon covers a large area which would make it hard for a point within a sample to fall under the country. Using an extremely large sample size ($> n = 3000$) would be computationally inefficient and therefore it was not done.

Results

Big countries (Canada & China): These countries were easy to find an estimate, and even when we removed parts of our multi-phase model and simply sampled on the entire Earth we still managed to get an estimate. However, the multi-phase architecture does give a much clearer outline, which makes the plots much nicer, and the estimates much closer to the true value. For example, in the output for Canada, we can see 2 samples on Vancouver Island! The plot of this output can also be found below with blue arrows indicating these sample points:



Even if we were sampling only in Canada, we'd expect to only see about 3/1000 samples land there. If we were sampling the whole Earth, we'd get only 1/15k, well over the runtime we allotted. Given the output above, for Canada, we found an estimated area of $8,694,650\ km^2$, compared to the true value of $9,984,670\ km^2$ with a variance of $44,234,436,711\ km^4$ and a 90% confidence bound of ($8,348,704\ km^2$, $9,040,595\ km^2$); thus our true value did not fall in the 90% confidence bound. Canada's estimated area is underestimated because of our usage of sinusoidal projection. Countries further away from the equator will become more distorted and get predicted with less accuracy than countries near or on the equator. For the estimated area of China, we got $8,642,355\ km^2$, compared to the true area of $9,562,910 km^2$ with a variance of $5.0871 \times 10^10\ km^4$ and a 90% confidence bound of ($8,271,365\ km^2$, $9,013,345\ km^2$). Like Canada, China's area is slightly underestimated. However, it is less so than Canada's since this country is closer to the equator, and thus will be influenced less by sinusoidal projection than Canada and its area estimation will therefore be more accurate. Lastly, when testing to see if there is a statistically significant difference between the estimated areas of China and Canada through a two-sample t-test, we found a t-statistic of 9.287877 and a p-value of 2.934089×10^{-20}. Therefore, at a 10% significance level, we reject the null hypothesis and conclude that there is a statistically significant difference between the estimated areas of China and Canada.

Something to note from the estimates of these countries is that the variances are quite large and there is an underestimation of the area. A factor that possibly contributes to this underestimation is the use of a sinusoidal projection to convert latitude and longitude values into distances.

Medium countries (Colombia & Egypt): These medium-sized countries were still relatively easy to sample, and we ended up getting better estimates for the area than for larger countries. For Colombia, we found an estimated area of $1,152,723\ km^2$ which is close to the true area of $1,141,748\ km^2$, with a variance of $1,455,804,664 km^4$ and a 90% confidence bound of ($1,089,963\ km^2$, $1,215,482\ km^2$). Therefore, the true area is actually captured in the bound. A factor of this outcome could be the use of the sinusoidal projection since Colombia is close to the equator (the equator runs through the lower eastern part of the country). Moreover, the estimated area will be very close since areas close to the equator and prime meridian are the most accurate when using a sinusoidal projection. Likewise, for Egypt, we found an estimated area of $971,899.6\ km^2$. Comparing it to the true value of $1,001,450\ km^2$, we see that our estimation is pretty accurate. Additionally, we also found an estimated variance of $834,550,133\ km^4$ and a 90% confidence bound with limits ($924,382.1\ km^2$, $1,019,417.1\ km^2$). Thus we can conclude that the estimate is in our 90% confidence bound and that our estimate is sufficient. Although not directly on the equator, the estimation for Egypt is quite close to its true value and thus is not affected by sinusoidal projection as much as other countries. Lastly, we performed a two-sample t-test to see if there is a statistically significant difference between the estimated areas of Colombia and Egypt. From this test, we found a t-statistic of 206.949 and a p-value of zero. Therefore, since our p-value is zero, we reject the null hypothesis and conclude that there is a statistically significant difference between the estimated areas of Colombia and Egypt.

Small countries (Cuba & Iceland): Similar to the estimation of big countries, there was an underestimation of the area of the sampled countries. As we get smaller with our countries, both the size of the country, sub-continent, and its relation to the equator affects the quality of our estimate. Looking at Cuba, we found an estimated area of $91,916.87\ km^2$, while its true area is $110,860\ km^2$, with a variance of $72,642,899\ km^4$ and 90% confidence bounds of ($77,897.65\ km^2$, $105,936.09\ km^2$). Looking at the plot output for this country, we can see that the algorithm finds bounds that closely capture the country. Therefore, our true value does not fall within the confidence bound, which has to do mainly with the country's location and size. Similarly, looking at Iceland's prediction, we find an estimated area of $88,166.25\ km^2$ which is lower than the true area is $103,000\ km^2$. Looking at the variance and a 90% confidence bound, we found a variance of $12,867,046\ km^4$ and a 90% confidence bound of ($82,266.05\ km^2$, $94,066.45\ km^2$), respectively. Like with Cuba, the true area is not within our confidence bound, and thus both countries are actually slightly underestimated. Lastly, conducting a two-sample t-test to see if is a statistically significant difference between the estimated areas of Cuba and Icelan, we found a t-statistic of 22.21546 and a p-value of 2.374777×10^{-10}. Therefore, since our p-value is less than our 10% significance level, we reject the null hypothesis and conclude that there is a statistically significant difference between the estimated areas of Cuba and Iceland.

Mini-countries (Jamaica, Lebanon & Singapore): Initially Jamaica proved too small to be captured by a single sample. Using the bounds provided by the continent-searching step, we found we would need about 10k uniform samples to get a single sample in Jamaica. To alleviate this, we introduced another step to search for the sub-continent, which is phase 3 of our algorithm. Doing so allowed us to find some data points in the country of interest; however, it did slow down the algorithm. When looking for an estimate for Jamaica, we found an estimated area of $8,966.842\ km^2$ compared to the true area of $10,991\ km^2$. We also found a variance of $3,470,192\ km^4$ and bounds for a 90% confidence bound of ($5,902.734\ km^2$, $12,030.949\ km^2$). Our true area is captured within our estimated 90% confidence bound. One important note about this size range is that the estimates are not always completely accurate; however, we get a relatively large variance and thus oftentimes the estimate will fall within the bounds. Next, we considered sampling Lebanon, and found an estimated area of $5,973.558\ km^2$ which is almost half of the true area which is $10,452\ km^2$. Looking at the variance and 90% confidence bound, we found a variance of $35,683,390\ km^4$ and 90% confidence bound of ($-3,852.07\ km^2$, $15,799.19\ km^2$). Interestingly, this is the first country that gives us a region that contains negative areas, which obviously cannot happen, despite this, the true value falls within the limits of our bound. In this case, estimating Lebanon struggles for two reasons: the bounds that capture the country found by the searches are larger making it hard for a point to fall into this country on the sampling and estimating process wich results in a large variance. The second reason is that Lebanon is relatively far away from the equator. Thus, it is quite affected by sinusoidal projection.

Unfortunately, Singapore proved too small to sample. In fact, when working initially on the algorithm in Python, geopandas, one of the libraries we planned to use didn't have Singapore on its main dataset "naturalearth_lowres" as it was considered part of Malaysia. Working with R on the algorithm on the other hand, the rworldmap library did contain the country of interest, and it also allowed us to refine our search allowing us to have search 3 in our algorithm [5]. However, despite our sub-continent capturing this country, we estimated it would take 10k samples to land even a single point in Singapore, which was beyond the "reasonable" runtime and sample size to run the code.

Challenges

Our main challenge came from sampling with the use of a sinusoidal projection. This projection increasingly distorts countries far from the equator [3]. Since we initially used uniform sampling for all phases, our estimates consistently underestimated the areas of countries far from the equator. In many cases, the true area would even lie above our 90% confidence bound. To fix this, we considered importance sampling using a shifted $Beta(1/2, 1/2)$ and $Beta(2, 2)$ distribution. Intuitively, it seemed a $Beta(2, 2)$ would do the best job of correcting the distortion caused by the sinusoidal projection, and this proved true as our estimates were close to the true value, yet still underestimated it. Our first run using the $Beta(½, ½)$ underestimated the area of China by 10%, a much worse result than we got with uniform sampling.

Since our algorithm has many layers from our boundary search phase, we tried many different combinations of the choice of distribution for each layer. We found that using a shifted $Beta(2, 2)$ to sample the latitude for the first 3 phases of sampling and then switching back to a uniform for the last 2 phases gave a better estimation result. However, we were still limited by the total number of samples we took, so tiny countries like Singapore and Monaco continued to get zero samples within their borders.

Conclusion

In the end, Monte-Carlo estimation sampling with a uniform distribution was able to achieve estimates close to the true area of the chosen country. For countries closer to the equator, it consistently captured the true area within its 90% confidence bounds. For countries closer to the poles, or those in larger continents, the true area occasionally falls above the 90% confidence bounds given by our algorithm.

An improvement to this code, as with any Monte-Carlo algorithm, would be to find ways to increase the number of points sampled while maintaining or increasing the efficiency of the code. Implementing our algorithm in R, which has a single-threaded model, combined with slow for-loop iteration significantly limited the number of samples we could achieve in a reasonable amount of time. However, simply switching to another language would be difficult, due to the lack of good geo-coding libraries available as in the case of Python.

Almost every other geo-coding library operates using web requests for each point. This prevents the need to load a massive super-high resolution map of the world, though it's also immensely slow, often being slower than our R code depending on the network. The alternative is loading the map locally. Most local maps found were either too low resolution like the case of geopandas "naturalearth_lowres" dataset, or far too massive, like Openstreetmap's dataset. An ideal approach would be stripping down Openstreetmap's dataset to just country polygons, though this process was beyond our capabilities for this project.

Despite appearing like a simple random sampling algorithm, we had to layer our search for latitude and longitude bounds that would capture a country of interest before applying Monte Carlo to estimate its area. In turn, this also increased the stability of our estimate and reduced the runtime. Using $Beta(2, 2)$ to get the latitude values for the bounds in earlier phases helped to mitigate the unevenness caused by the sinusoidal projection. Even with our algorithm's capability for estimating countries with an area over $10,000km^2$, a classical flood-fill algorithm remains a far stronger choice, particularly compared to our final phase of Monte Carlo sampling.

References

[1] Martin-Alarcon, D. (2019). The geopandas cookbook. Daniel Martin-Alarcon. Retrieved April 4, 2023, from https://www.martinalarcon.org/2018-12-31-d-geopandas/ (https://www.martinalarcon.org/2018-12-31-d-geopandas/)

[2] Lovelace, R., Nowosad, J., & Muenchow, J. (1970, January 1). Chapter 2 geographic data in R: Geocomputation with R. Chapter 2 Geographic data in R | Geocomputation with R. Retrieved April 4, 2023, from https://r.geocompx.org/spatial-class.html (https://r.geocompx.org/spatial-class.html)

[3] Wikimedia Foundation. (2023, April 2). Sinusoidal projection. Wikipedia. Retrieved April 4, 2023, from https://en.wikipedia.org/wiki/Sinusoidal_projection (https://en.wikipedia.org/wiki/Sinusoidal_projection)

[4] Wikimedia Foundation. (2023, March 24). Geographic coordinate system. Wikipedia. Retrieved April 4, 2023, from https://en.wikipedia.org/wiki/Geographic_coordinate_system#/media/File:FedStats_Lat_long.svg (https://en.wikipedia.org/wiki/Geographic_coordinate_system#/media/File:FedStats_Lat_long.svg)

[5] South, A. (2022, October 14). Package ‘rworldmap’ - cran.r-project.org. Package ‘rworldmap’. Retrieved April 5, 2023, from https://cran.r-project.org/web/packages/rworldmap/rworldmap.pdf (https://cran.r-project.org/web/packages/rworldmap/rworldmap.pdf)

[6] Comprehensive R Archive Network (CRAN). (2023, January 19). Classes and methods for spatial data [R package sp version 1.6-0]. The Comprehensive R Archive Network. Retrieved April 4, 2023, from https://cran.r-project.org/web/packages/sp/index.html (https://cran.r-project.org/web/packages/sp/index.html)

Code of Estimation

Main functions to identify a set of coordinates to a location on Earth:

```
set.seed(2023)
library(sp)# package that provides spatial data in R1
```

```
## Warning: package 'sp' was built under R version 4.1.3
```

```
library(rworldmap) #has functions that match coordinates to locations on the map
```

```
## Warning: package 'rworldmap' was built under R version 4.1.3
```

```
library(ggplot2) #graphing purposes I guess? (Haven't done any graphing yet)
```

```
## Warning: package 'ggplot2' was built under R version 4.1.3
```

```
library(countrycode) #has a function that matches country to continent
```

```
## Warning: package 'countrycode' was built under R version 4.1.3
```

```
library(sf)
```

```
## Warning: package 'sf' was built under R version 4.1.3
```

```
library(units)
```

```
## Warning: package 'units' was built under R version 4.1.3
```

```r
# Function that matches coordinates to country
coord2country = function(llong){
  countriesSP = getMap(resolution = "low")
  sPoints = SpatialPoints(llong, proj4string = CRS(proj4string(countriesSP)))
  indices = over(sPoints, countriesSP)
  indices$ADMIN
}
# Function that matches coordinates to region
coord2region = function(llong){
  countriesSP = getMap(resolution = "low")
  sPoints = SpatialPoints(llong, proj4string = CRS(proj4string(countriesSP)))
  indices = over(sPoints, countriesSP)
  indices$Stern
}
regionname = function(country){
  datastern = country2Region(regionType = "Stern")
  region = c()
  for (i in 1:13){
    sternname = names(datastern)[i]
    for (j in 1:length(datastern[[i]])){
        if (grepl(country, datastern[[i]][j], ignore.case = T)==T){
          region = c(region, sternname)
        }
    }
  }
  return(unique(region))
}
# Function that matches coordinates to continent
coord2cont = function(llong){
  countriesSP = getMap(resolution = "low")
  sPoints = SpatialPoints(llong, proj4string = CRS(proj4string(countriesSP)))
  indices = over(sPoints, countriesSP)
  indices$REGION
}

# Adjustable Latitude Longitude data frame of 1 set of coordinates used in search 4 as well as p
hase 5. Used Uniform distribution.
llpointu = function(laL, laU, loL, loU){
  df = data.frame(
  long = runif(1, loL, loU),
  lat = runif(1, laL, laU))
  return(df)
}
# Adjustable Latitude Longitude data frame of 1 set of coordinates used in search 1-3. Uses adju
sted Beta(2,2) and Uniform(LoL, LoU)
llpointb = function(laL, laU, loL, loU){
  df = data.frame(
  long = runif(1, loL, loU),
  lat = (laU-laL)*rbeta(1, 2, 2)+laL)
  return(df)
}
```

```
# Sinusoidal projection. Converts lat-long coordinates to distances in the map
siproj = function(lat, long){
  earthrad = 6371009 #in meters
  latitude_dist = pi*earthrad/180
  latr = lat*pi/180
  x = (long)*cos(latr)*latitude_dist #
  y = lat*latitude_dist #
  return(c(x, y))
}

# Search 1: Continent Find
# Initial search of the continent where country can be found.
# Returns bounds that roughly capture the continent and in turn the country of interest.
search1 = function(continent, country){
  # number of replications
  n = 100
  # Set initial bounds (this is all of the lat-lon bounds of Earth)
  laL = -90
  laU = 90
  loL = -180
  loU = 180
  # list to store latitude and longitude values
  long = c()
  lat = c()
  for (i in 1:n) {
    points = llpointb(laL, laU, loL, loU)
    # while condition prevents <NA> points from passing to the if else conditions below
    while (is.na(coord2cont(points)) == T) {
      points = llpointb(laL, laU, loL, loU)
    }
    # matching coordinates to continent
    llcontinent = coord2cont(points)

    # conditions to match llcontinent to the continent of country input
    if (llcontinent == continent){
      long = c(long, points$long)
      lat = c(lat, points$lat)
      }
    else{
      # llcontinent and continent have different terms for America and Oceania
      if (grepl("America", continent, ignore.case = T) & grepl("America", llcontinent, ignore.ca
se = T) == T){
        long = c(long, points$long)
        lat = c(lat, points$lat)
      }
      if (grepl("Oceania", continent, ignore.case = T) & grepl("Australia", llcontinent, ignore.
case = T) == T){
        long = c(long, points$long)
        lat = c(lat, points$lat)
      }
      else{
        next
      }
    }
  }
  # Returning smaller bounds that "roughly" capture the continent of interest
  return(list(
    laL = min(lat),
    laU = max(lat),
    loL = min(long),
    loU = max(long)
  )
  )
}

# Search 2: Continent Search
# Secondary search of the continent where country can be found.
#Returns bounds that roughly capture the continent and in turn the country of interest.
search2 = function(sr1, continent, country){
  # number of replications
  n = 100
  # Widening the bounds before refining
  if (sr1$laL - 20< -90){
    laL = sr1$laL + (-90-sr1$laL)/2
  }
  else{
    laL = sr1$laL - 20
  }
  if (sr1$laU+20 > 90){
    laU = sr1$laU+ (90-sr1$laL)/2
  }
  else{
    laU = sr1$laU + 20
  }
  if (sr1$loL-20 < -180){
    loL = sr1$loL + (-180-(sr1$loL))/2
  }
  else{
    loL = sr1$loL - 20
  }
  if(sr1$loU+20 > 180){
    loU = sr1$loU + (180-(sr1$loL))/2
  }
  else{
    loU = sr1$loU + 20
  }
  # List to store latitude and longitude
  long = c()
  lat = c()
  cont = c()
  # Similar conditions as search 1
  for (i in 1:n) {
    points = llpointb(laL, laU, loL, loU)
    while (is.na(coord2cont(points)) == T) {
      points = llpointb(laL, laU, loL, loU)
    }

    llcontinent = coord2cont(points)
    if (llcontinent == continent){
      long = c(long, points$long)
      lat = c(lat, points$lat)
      cont = c(cont, llcontinent)
      }
    else{
      # Refine search for countries in N/S/C America as well as Greenland (special case)
      if (grepl("America", continent, ignore.case = T) & ("Europe"== llcontinent) & (country=="G
reenland")){
        long = c(long, points$long)
        lat = c(lat, points$lat)
        cont = c(cont, "North")
      }
      if (grepl("America", continent, ignore.case = T) & grepl("America", llcontinent, ignore.ca
se = T) & (country!="Greenland") == T){
        # Given that a country is not Greenland, and is in N or S America. If the country is eit
her Canada or the US we store the point under "North", else "south".
        if ((grepl("North", llcontinent, ignore.case = T) == T) & (country =="Canada")){
          long = c(long, points$long)
          lat = c(lat, points$lat)
          cont = c(cont, "North")
        }
        if ((grepl("North", llcontinent, ignore.case = T)& grepl("United", country, ignore.case
= T) == T)){
          long = c(long, points$long)
          lat = c(lat, points$lat)
          cont = c(cont, "North")
        }
        if ((grepl("South", llcontinent, ignore.case = T)&(country!="Canada")== T)&(grepl("Unite
d", country, ignore.case = T) == FALSE)){
          long = c(long, points$long)
          lat = c(lat, points$lat)
          cont = c(cont, "South")
        }
      }
      # Oceania/Australia case
      if (grepl("Oceania", continent, ignore.case = T) & grepl("Australia", llcontinent, ignore.
case = T) == T){
        long = c(long, points$long)
        lat = c(lat, points$lat)
      }
      else{
        next
      }
    }
  }
  laL = min(lat)
  laU = max(lat)
```

```r
  loL = min(long)
  loU = max(long)
  # correct upper latitude for countries in South America
  if (cont[1]=="South"){
    laU = laU - 5
  }

  # Returns more refined bounds that will be used for country search
  return(list(
    laL = laL,
    laU = laU,
    loL = loL,
    loU = loU
  )
  )
}

# Search 3: Region Search
# Initial search of the country within continent bounds.
# Returns bounds that roughly capture the country of interest within within a region in the cont
inent.
search3 = function(sr2, country){
  # higher number of replications
  n = 300
  # same widening of bounds as in search 2
  if (sr2$laL - 20< -90){
    laL = sr2$laL + (-90-sr2$laL)/2
  }
  else{
    laL = sr2$laL - 20
  }
  if (sr2$laU+20 > 90){
    laU = sr2$laU+ (90-sr2$laL)/2
  }
  else{
    laU = sr2$laU + 20
  }
  if (sr2$loL-20 < -180){
    loL = sr2$loL + (-180-(sr2$loL))/2
  }
  else{
    loL = sr2$loL - 20
  }
  if(sr2$loU+20 > 180){
    loU = sr2$loU + (180-(sr2$loL))/2
  }
  else{
    loU = sr2$loU + 20
  }
  long = c()
  lat = c()
  for (i in 1:n) {
    points = llpointb(laL, laU, loL, loU)
    while (is.na(coord2region(points)) == T) {
      points = llpointb(laL, laU, loL, loU)
    }
    # matches llregion to country region, else it goes to the next replicaiton
    llregion = coord2region(points)
    region = regionname(country)
    if ((llregion %in% region)==T){
      long = c(long, points$long)
      lat = c(lat, points$lat)
      }
    else{
      next
    }
  }

  # setting the bound values
  laL = min(lat)
  laU = max(lat)
  loL = min(long)
  loU = max(long)
  # Apply an adjustment to make the boundaries slightly wider or better fitted to the initial bo
undaries if any of the bound values exceeds the initial set bounds at the start of search 1
  if (laL - 10< -90){
    laL = laL + (-90-laL)/2
  }
  else{
    laL = laL - 10
  }
  if (laL < -90){
      laL = -90
  }
  if (laU+20 > 90){
    laU = laU+ (90-laL)/2
  }
  else{
    laU = laU + 20
  }
  if (laU > 90){
      laU = 90
  }
  if (loL-20 < -180){
    loL = loL + (-180-(loL))/2
  }
  else{
    loL = loL - 20
  }
  if(loL < -180){
    loL = -180
  }
  if(loU+10 > 180){
    loU = loU + (180-(loL))/2
  }
  else{
    loU = loU + 10
  }
  if (loU > 180){
    loU = 180
  }
  # returning the lat-long bounds that will be used in phase 4 to estimate the area.
  return(list(
    laL = laL,
    laU = laU,
    loL = loL,
    loU = loU  )
  )
}

# Search 4: Country Search
# Refined search of the country within region bounds.
# Returns bounds that roughly capture the country of interest in the region.
search4 = function(s3, country){
  n = 500
  # widening of bounds done in search 3
  laL = s3$laL
  laU = s3$laU
  loL = s3$loL
  loU = s3$loU
  long = c()
  lat =  c()
  for (i in 1:n){
    points  = llpointu(laL, laU, loL, loU)
    while (is.na(coord2country(points)) == T) {
      points = llpointu(laL, laU, loL, loU)
    }
    # matches llcountry to country, else it goes to the next replication
    llcountry = coord2country(points)
    if (grepl(country, llcountry, ignore.case = T) == T){
      long = c(long, points$long)
      lat = c(lat, points$lat)
    }
    else{
      next
    }
  }

  if(length(lat)==0){
    # similar adjustment as in search 3
    laL = laL + 5
    laU = laU - 5
    loL = loL + 5
    loU = loU - 5
    return(list(
      laL = laL,
      laU = laU,
      loL = loL,
      loU = loU
      )
    )
  }
  else{
    laL = min(lat)
    laU = max(lat)
    loL = min(long)
    loU = max(long)
    # similar adjustment as in search 3
    if (laL - 5< -90){
      laL = laL + (-90-laL)/2
    }
    else{
      laL = laL - 5
    }
    if (laL < -90){
        laL = -90
    }
    if (laU+5 > 90){
      laU = laU+ (90-laL)/2
    }
    else{
      laU = laU + 5
    }
    if (laU > 90){
        laU = 90
    }
    if (loL-5 < -180){
      loL = loL + (-180-(loL))/2
    }
    else{
      loL = loL - 5
    }
    if(loL < -180){
      loL = -180
    }
    if(loU+5 > 180){
      loU = loU + (180-(loL))/2
    }
    else{
      loU = loU + 5
    }
    if (loU > 180){
      loU = 180
    }
    # returning the lat-long bounds that will be used in phase 5 to estimate the area.
    return(list(
      laL = laL,
      laU = laU,
```

```r
      loL = loL,
      loU = loU  )
    )
  }

}

cordArea = function(name){
  # Reverse Search: Get the continent and match it to a bound of latitudes and longitudes, then
use that to find the country and return a bound of latitudes and longitudes
  country  = name
  continent = countrycode(sourcevar = country, origin = "country.name", destination = "continen
t")
  s1 = search1(continent, country)
  s2 = search2(s1, continent, country)
  s3 = search3(s2, country)
  s4 = search4(s3, country)
  return(s4)
}

areaD = function(sr4){
  # Function that gets the area bounded by the latitude and longitude range found by coord area.
  # Use of sunusoidal projection.
  p1 = siproj(sr4$laU,sr4$loL)
  p2 = siproj(sr4$laL,sr4$loL)
  p3 = siproj(sr4$laL,sr4$loU)
  p4 = siproj(sr4$laU,sr4$loU)
  lat = c(p1[2], p2[2], p3[2] ,p4[2]+10, p1[2])
  long = c(p1[1], p2[1], p3[1] ,p4[1], p1[1])
  # polygon of the transformed lat-long bounds found in search 3
  polygon = st_sfc(st_polygon(list(cbind(long, lat)))) %>% st_set_crs(32615)
  conv = set_units(st_area(polygon), km^2) #converts the area from square meters to square kilo
meters
  return(conv)
}

phase5 = function(sr4, country){
  # function that does MC simulation to actually find the area of the country
  # all lat-long values are stored
  n = 3000
  long = c()
  lat = c()
  bin = c() # binary indicator
  for (i in 1:n) {
    points = llpointu(sr4$laL, sr4$laU, sr4$loL, sr4$loU)
    if (is.na(coord2country(points)) == T) {
      transform = siproj(points$lat, points$long)
      long = c(long, transform[1])
      lat = c(lat, transform[2])
      bin = c(bin, 0)
      next
    }
    # matches llcountry to country, else it goes to the next replicaiton
    else{
      llcountry = coord2country(points) # matching sampled lat-long points to a country within t
he bounds of search 4
      # binary indicator. 1 inidcates that the point sampled falls in the country, 0 else.
      if (grepl(country, llcountry, ignore.case = T) == T){
        transform = siproj(points$lat, points$long)
        long = c(long, transform[1])
        lat = c(lat, transform[2])
        bin = c(bin, 1)
      }
      else{
        transform = siproj(points$lat, points$long)
        long = c(long, transform[1])
        lat = c(lat, transform[2])
        bin = c(bin, 0)
        next
        }
    }
  }
  # Setting data frame with latitude, longitude, and the binary indicator as columns
  df = data.frame(long, lat, bin)
  return(df)
}

# Function that gives the estimated area, variance, and confidence bounds
ar_est = function(area, df, alpha){
  nrows = nrow(df)# sample size
  points = sum(df$bin)# number of points that fall in the country of interest
  fraction = points/nrows #proportion of points that fall in the country of interest
  estAr = area*fraction # Estimated area
  estV = (area^2)*fraction*(1-fraction)/(nrows-1) # estimated variance
  normCI = c(estAr-qnorm(1-alpha/2)*sqrt(estV), estAr+qnorm(1-alpha/2)*sqrt(estV)) # confidence
bounds (normal)
  return(list(
    mean = estAr,
    variance = estV,
    ci = normCI
  ))
}
# function that gives the graphics when visualize = T. Shows the polygon and sampled points that
fall in the country of interest.
graphics = function(coordinates, df, name){
  p1 = siproj(coordinates$laU,coordinates$loL)
  p2 = siproj(coordinates$laL,coordinates$loL)
  p3 = siproj(coordinates$laL,coordinates$loU)
  p4 = siproj(coordinates$laU,coordinates$loU)
  lat = c(p1[2], p2[2], p3[2] ,p4[2]+10, p1[2])
  long = c(p1[1], p2[1], p3[1] ,p4[1], p1[1])
  polygon = st_sfc(st_polygon(list(cbind(long, lat)))) %>% st_set_crs(32615)
  onedf = df[df$bin == 1,]
  plot(polygon, axes = T, main = paste("Sample Points of", name, sep = " "))
  points(x = onedf$long, y = onedf$lat, pch = 10, col = "brown2", cex = 2)
}

# Main function. Returns estimated area, variance, and confidence bounds set at 90%
countryArea =  function(name, visualize = F){
  coordinates = cordArea(name) # lat-long bounds from search 3

  area = areaD(coordinates) # calculating area of polygon
  sample = phase5(coordinates, name)
  estimate = ar_est(area, sample, alpha = 0.10) # estimates for country

  if (visualize == T){
    graphics(coordinates, sample, name)
  }
  return(estimate)
}
```

Code for Comparison

This code is for a two sample t-test to detect a statistically significant difference in the land areas of the selected pairs of countries under big, medium and small categories

```
tcomparison = function(country1, country2){
 mean1 = country1$mean
 mean2 = country2$mean
 tmean = mean1-mean2
 v1 = country1$variance
 v2 = country2$variance
 n=3000
 sd = sqrt((v1/n)+(v2/n))
 tstat = tmean/sd
 pval = 2*pt(tstat, df = n-1, lower = FALSE)
 return(list(
    tstat = tstat,
    pval = pval
  ))
}
```