Estimating the Area of Nations Revision

Milagros N. Cortez

2023-06-20

Introduction

This project is a revision of a previous project (https://milic01.github.io/Stat-413-Project) for Stat 413 at the University of Alberta. The project consists on designing an algorithm to estimate the area of any country on Earth using Monte-Carlo simulation. The algorithm simulates the latitude and longitude points from the surface of the Earth, and uses a lookup table to match the simulated coordinates to the country of interest. 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 of interest in $[km^2]$, its variance $[km^4]$, and a 90% confidence bound for our estimate of the land area.

Notes About the Previous Version

In this revision we note that in the previous version we used a sinusoidal projection also known as the Sanson-Flamsteed and Mercer-Sanson projection which consists of a pseudocylindrical equal-area projection that has the poles of Earth as points with distorted meridians and continents. The conversion was used to convert points with latitude and longitude values into distances in the map which were used to calculate the area of land that captured the country of interest. The functions used for the sinusoidal projection were:

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

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

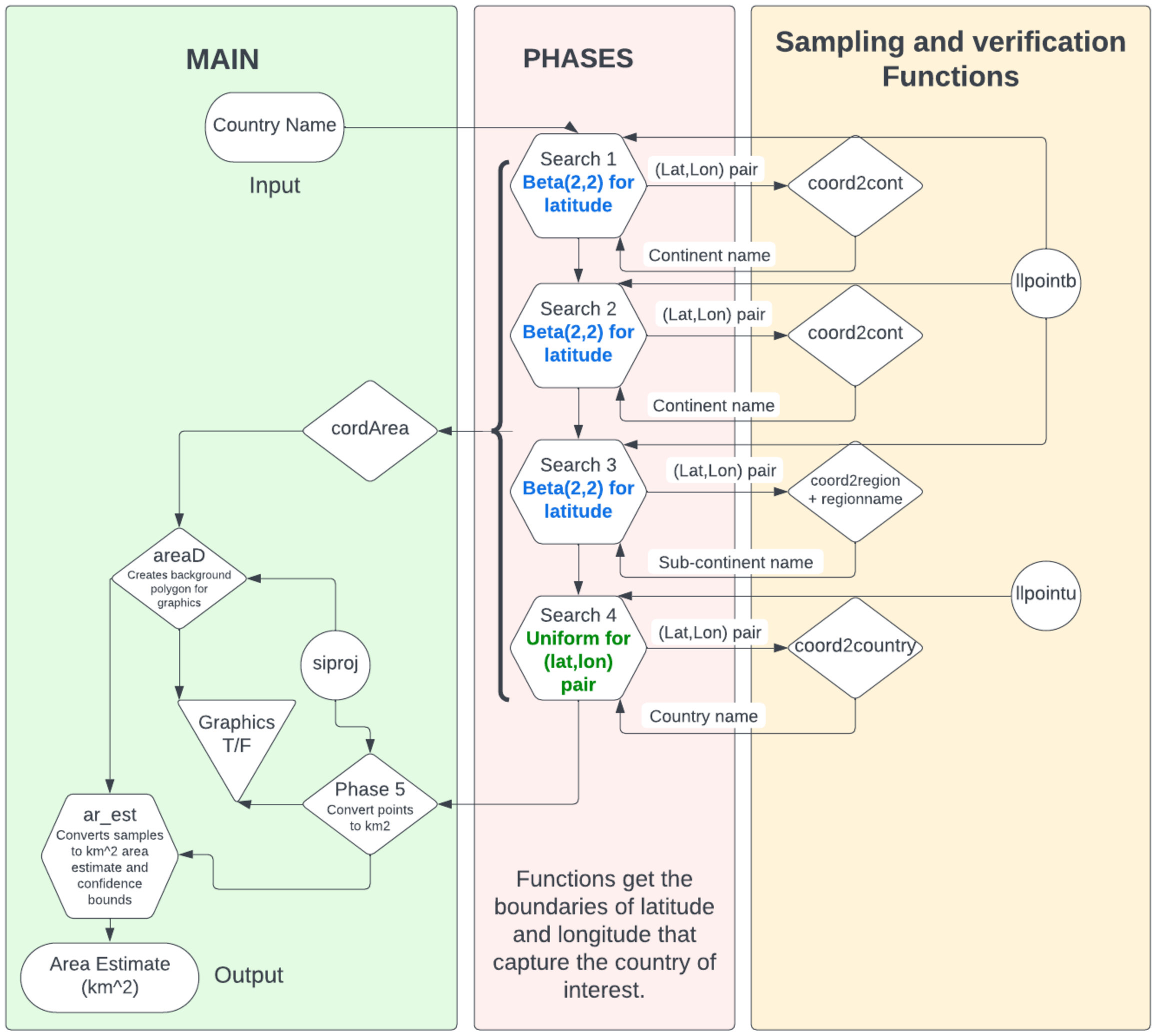
where ϕ is the longitude and θ is the latitude of a point. Using this projection, it was found that the algorithm underestimated the area of countries with a large area like Canada and China and countries with a small area like Cuba and Iceland. It was also hard for the algorithm to estimate the area of smaller countries like Lebanon and Singapore. The underestimation of some countries can be due to the projection increasingly distorting countries far from the equator. Below we can see a table of known areas retrieved from google of each country tested and the results from the algorithm:

	True Area [km^2]	Estimated Mean Area [km^2]	Variance [km^4]	LB [km^2]	UB [km^2]
Canada	9984670.0	8694650.00	44234436711	8348704.00	9040595.00
China	9562910.0	8642355.00	50871000000	8271365.00	9013345.00
Colombia	1141748.0	1152723.00	1455804664	1089963.00	1215482.00
Egypt	1001450.0	971899.60	834550133	924382.10	1019417.10
Cuba	110860.0	91916.87	72642899	77897.65	105936.09
Iceland	103000.0	88166.25	12867046	82266.05	94066.45
Lebanon	10452.0	9573.56	35683390	-3852.07	15799.19
Jamaica	10991.0	8966.84	3470192	5902.73	12030.95
Singapore	748.6	0.00	0	0.00	0.00

In this algorithm, something that worked well was the use of a boundary search phase which gave a much clearer outline of the country's location to get an estimate close to the true value. We also noticed that a scaled $Beta(2, 2)$ to sample latitude for the first three search phases, and a uniform distribution for the last two phases gave a better estimation result.

Original Algorithm Process

Below is a flowchart of how the original algorithm is applied to the R code:



To capture the country of interest regardless of size, we split the original algorithm into (1) a step that finds a boundary that captures the country of interest, 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 thee 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 $Beta(2, 2) * 180 - 90$, 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.

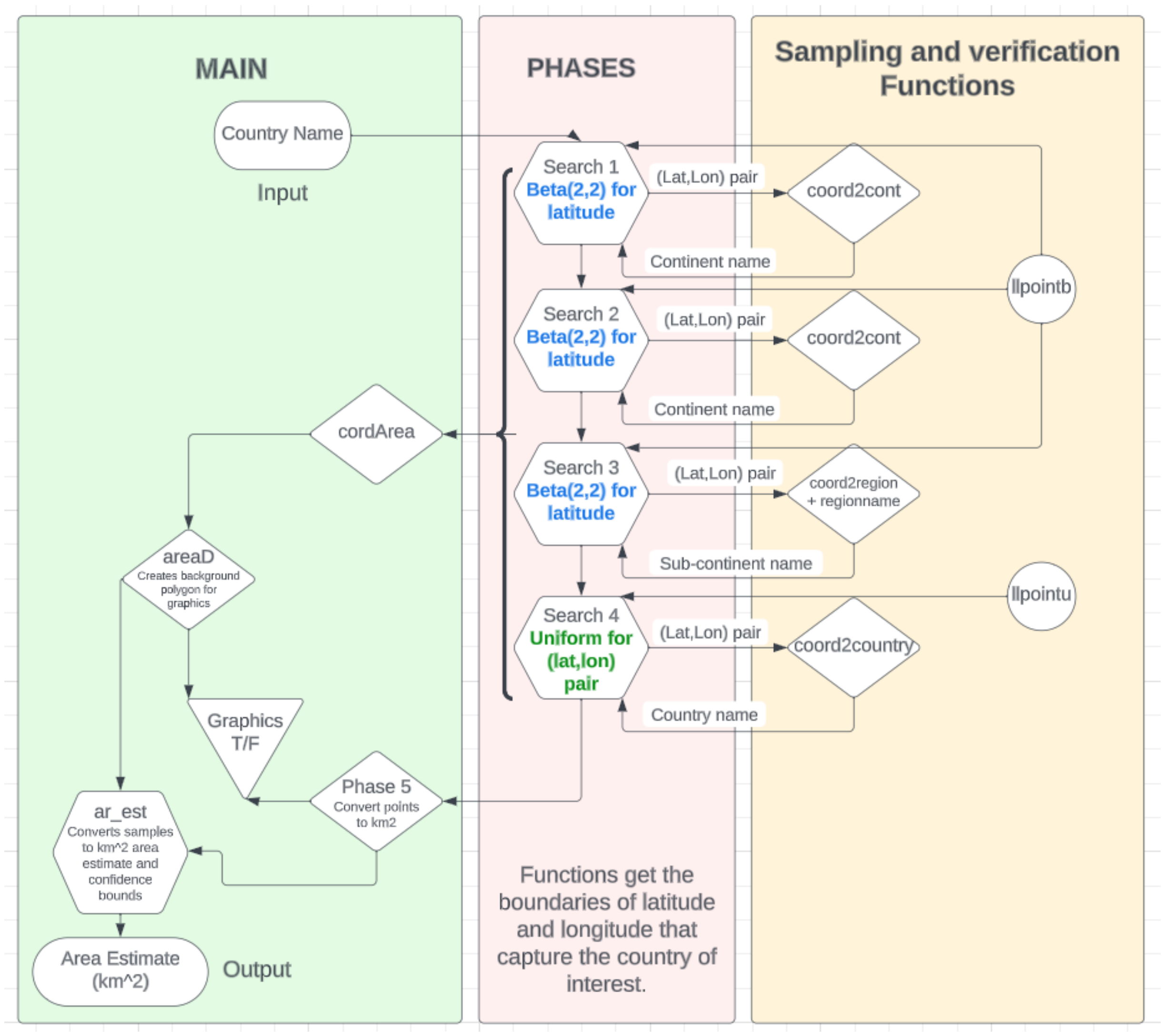
4. Finally for search 4, we search for the bounds in which the country itself is contained, sampling only in this sub-continent. Unlike the previous searches, we use a uniform distribution to sample both the latitude and longitude values, with about 5 degrees of added margin on both sides. Here we sample using $n = 1000$ 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 $n = 4000$ 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 90% confidence bound.

For this revised version we use a different method by taking out the siproj function which used the sinusoidal projection to convert latitude and longitude coordinates into distances, and instead use volume integration directly in function areaD to find the area of a spherical rectangle bound by latitude and longitude coordinates.

Below is the algorithm with the new changes:


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
Graphics T/F
Phase 5 Convert points to km2
ar_est Converts samples to km^2 area estimate and confidence bounds
Functions get the boundaries of latitude and longitude that capture the country of interest.
Area Estimate (km^2)
Output

flowchrt2

Calculating the Volume of a Spherical Rectangle

as part of the revised version we will use volume integration to find the area of a spherical rectangle bound by latitude and longitude coordinates. We note the functions of spherical coordinates are:

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

$y = rcos(\phi)sin(\theta)$

$z = rsin(\phi)$

where r is the radius, ϕ is the latitude, and θ is the longitude. The radius of Earth is $6,371\ km$. To calculate the volume we have to first find the determinant of the Jacobian of the equations above:

$$det \begin{bmatrix} \frac{\delta x}{\delta r} & \frac{\delta x}{\delta \phi} & \frac{\delta x}{\delta \theta} \\ \frac{\delta y}{\delta r} & \frac{\delta y}{\delta \phi} & \frac{\delta y}{\delta \theta} \\ \frac{\delta z}{\delta r} & \frac{\delta z}{\delta \phi} & \frac{\delta z}{\delta \theta} \end{bmatrix}$$

$$= det \begin{bmatrix} cos(\phi)cos(\theta) & -rsin(\phi)cos(\theta) & -rcos(\phi)sin(\theta) \\ \cos(\phi)sin(\theta) & -rsin(\phi)sin(\theta) & rcos(\phi)cos(theta) \\ sin(\phi) & rcos(\phi) & 0 \end{bmatrix}$$

$$= cos(\phi)cos(\theta)\ det \begin{bmatrix} -rsin(\phi)sin(\theta) & rcos(\phi)cos(theta) \\ rcos(\phi) & 0 \end{bmatrix} + rsin(\phi)cos(\theta)\ det \begin{bmatrix} \cos(\phi)sin(\theta) & rcos(\phi)cos(theta) \\ sin(\phi) & 0 \end{bmatrix} - rcos(\phi)sin(\theta)\ det \Bigg[cos$$

$= R^2cos(\theta)$

$= |J|$

We use the result above to get the volume of a spherical rectangle determined by latitude and longitude bounds (ϕ_l, ϕ_u) and (θ_l, θ_u) through integration:

$$area = \int_{\theta_l}^{\theta_u} \int_{\phi_l}^{\phi_u} |J| \; d\phi d\theta$$

$$= \int_{\theta_l}^{\theta_u} \int_{\phi_l}^{\phi_u} R^2 cos(\theta) \; d\phi d\theta$$

$$= R^2 \int_{\theta_l}^{\theta_u} \int_{\phi_l}^{\phi_u} cos(\theta) \; d\phi d\theta$$

$$= R^2 [\phi_u - \phi_l] \int_{\theta_l}^{\theta_u} cos(\theta) \; d\theta$$

$$= R^2 [\phi_u - \phi_l][sin(\theta_u) - sin(\theta_l)]$$

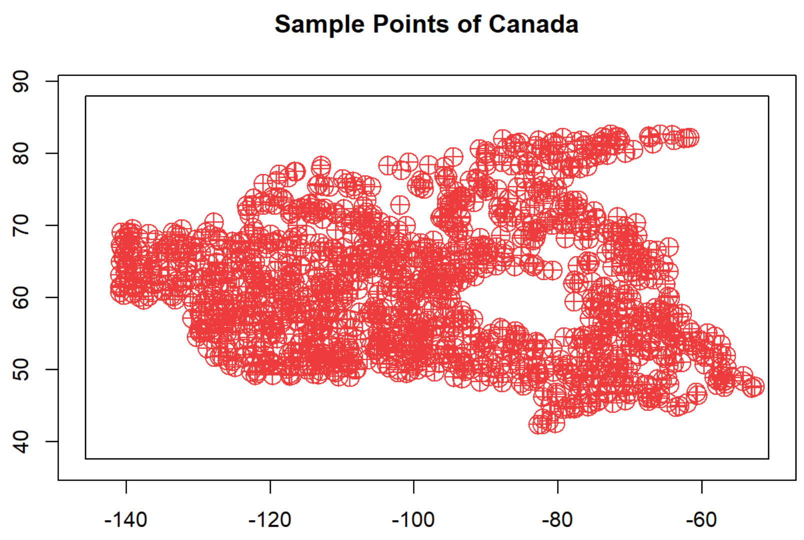
as the radius of Earth is $6,371 \; km$.

$$= 6371^2 [\phi_u - \phi_l][sin(\theta_u) - sin(\theta_l)]$$

We use the last equation to find the area enclosed by the latitude and longitude found in phase 4 which is done by the function areaD of the algorithm.

Estimating Big Countries

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

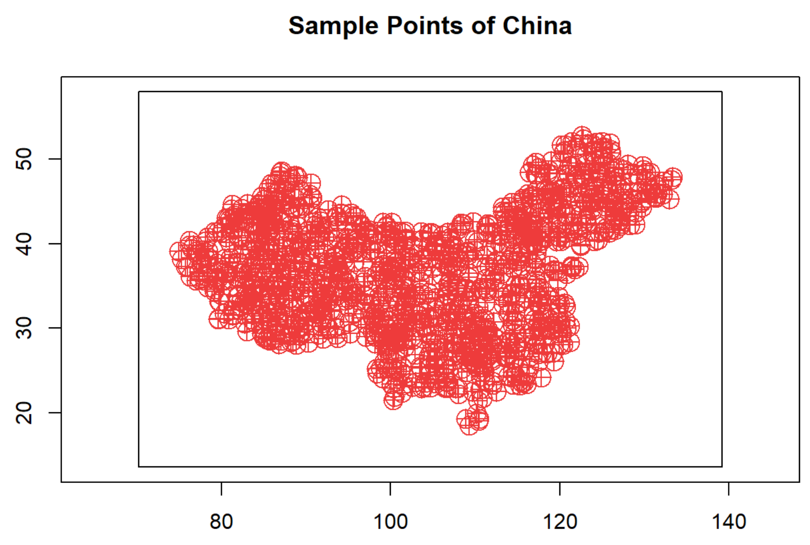


```
canada
```

```
## $mean
## 9272999 [km^2]
##
## $variance
## 39281956563 [km^4]
##
## $ci
## Units: [km^2]
## [1] 8946994 9599003
```

We can see from the output above that the estimated mean area of $9,272,999 \; km^2$ and the 90% confidence bound of $8,946,994 \; km^2$ to $9,599,003 \; km^2$ are lower than the true area of Canada ($9,984,670 \; km^2$). However, this estimate is closer to the true value compared to the estimate of the previous algorithm which estimated a mean area of $8,694,650 \; km^2$ and a 90% confidence bound of $8,348,704 \; km^2$ to $9,040,595 \; km^2$. Another thing to note is that the graphics are less distorted than the graphics of the previous version.

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



```
china
```

```
## $mean
## 8823161 [km^2]
##
## $variance
## 46690810396 [km^4]
##
## $ci
## Units: [km^2]
## [1] 8467740 9178582
```

We can see from the output above that the estimated mean area of $8,823,161\ km^2$ and the 90% confidence bound of $8,467,740\ km^2$ to $9,178,582\ km^2$ are lower than the true area of China $(9,562,910\ km^2)$. However, this estimate is closer to the true value compared to the estimate of the previous algorithm which estimated a mean area of $8,642,355\ km^2$ and a 90% confidence bound of $8,271,365\ km^2$ to $9,013,345\ km^2$. 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 a detect statistically significant difference in the land areas of Canada and China. We set the null and alternative hypothesis:

$H_0 : \mu_{Canada} = \mu_{China}$

$H_a : \mu_{Canada} \neq \mu_{China}$

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

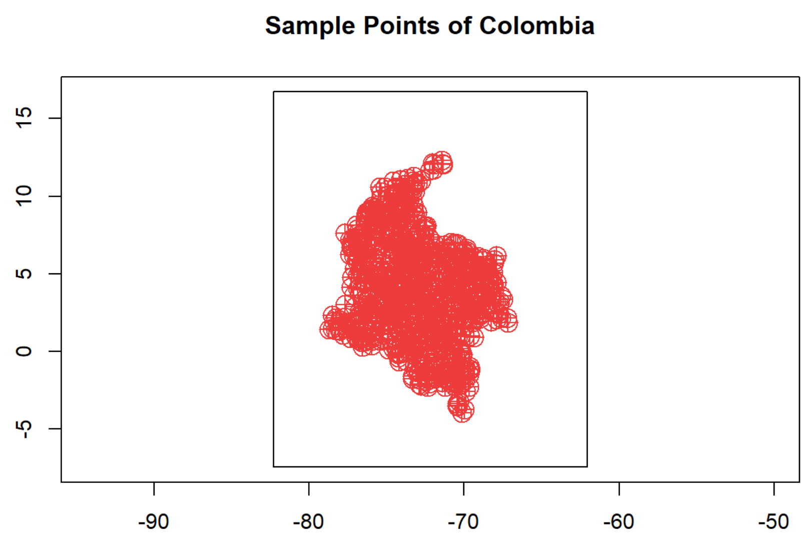
```
tcomparison(canada, china)
```

```
## $tstat
## 84.03036 [1]
##
## $pval
## 0 [1]
```

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

Estimating Medium Size Countries

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

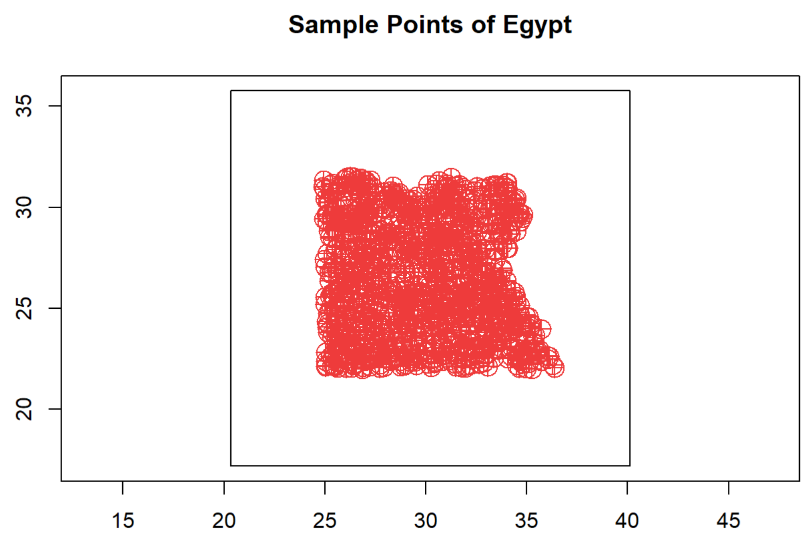


```
colombia
```

```
## $mean
## 1150909 [km^2]
##
## $variance
## 1393929275 [km^4]
##
## $ci
## Units: [km^2]
## [1] 1089498 1212320
```

We can see from the output above that the estimated mean area of $1,150,909\ km^2$ is close to the true area of Colombia ($1,141,748\ km^2$) and the 90% confidence bound of $1,089,498\ km^2$ to $1,212,320\ km^2$ captures the true area.

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



```
egypt
```

```
## $mean
## 998253.5 [km^2]
##
## $variance
## 760697979 [km^4]
##
## $ci
## Units: [km^2]
## [1]  952887.2 1043619.8
```

We can see from the output above that the estimated mean area of $998,253.5\ km^2$ is close to the true area of Egypt ($1,001,450\ km^2$) and the 90% confidence bound of $952,887.2\ km^2$ to $1,043,619.8\ km^2$ captures the true area.

We can see that just like the previous algorithm, this algorithm also closely estimates the area of medium sized countries. We proceed to do a two-sample t-test at the 10% level to see if we can a detect statistically significant difference in the land areas of Colombia and Egypt. We set the null and alternative hypothesis:

$H_0 : \mu_{Colombia} = \mu_{Egypt}$

$H_a : \mu_{Colombia} \neq \mu_{Egypt}$

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

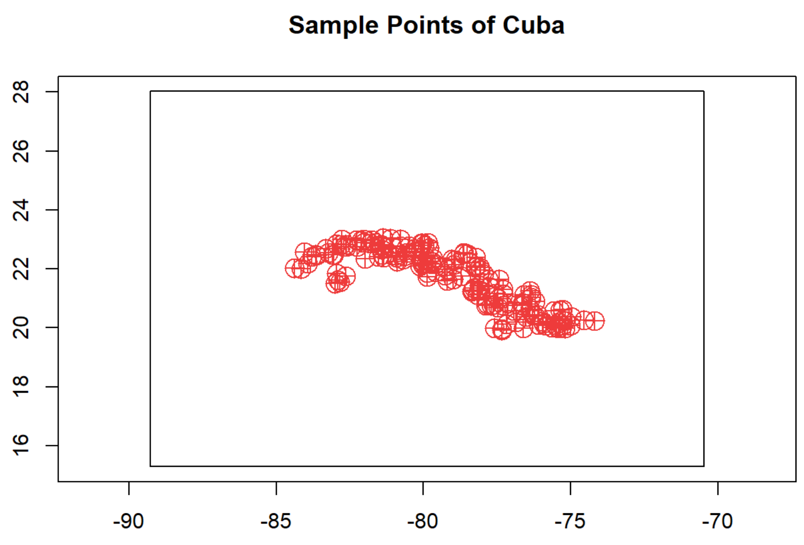
```
tcomparison(colombia, egypt)
```

```
## $tstat
## 180.1305 [1]
##
## $pval
## 0 [1]
```

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

Estimating Small Countries

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

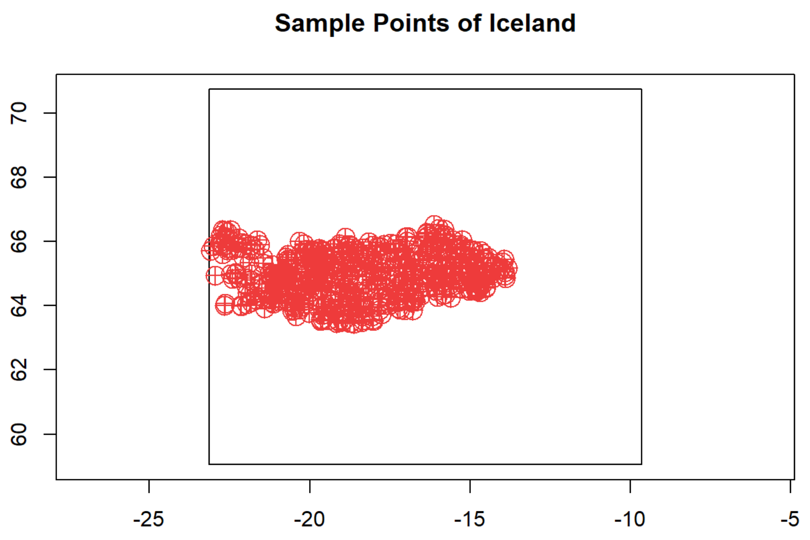


```
cuba
```

```
## $mean
## 103690.6 [km^2]
##
## $variance
## 68532768 [km^4]
##
## $ci
## Units: [km^2]
## [1]  90073.75 117307.43
```

We can see from the output above that the estimated mean area of $103,690.6\ km^2$ is close to the true area of Cuba ($110,860\ km^2$) and the 90% confidence bound of $90,073.75\ km^2$ to $117,307.43\ km^2$ captures the true area.

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



```
iceland
```

```
## $mean
## 103642.9 [km^2]
##
## $variance
## 18717336 [km^4]
##
## $ci
## Units: [km^2]
## [1]  96526.71 110759.15
```

We can see from the output above that the estimated mean area of $103,642.9\ km^2$ is close to the true area of Iceland ($103,000\ km^2$) and the 90% confidence bound of $96,526.71\ km^2$ to $110,759.15\ km^2$ captures the true area.

We can see that this algorithm closely estimates the area of small sized countries compared to the previous algorithm. We proceed to do a two-sample t-test at the 10% level to see if we can a detect statistically significant difference in the land areas of Cuba and Iceland. We set the null and alternative hypothesis:

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

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

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

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

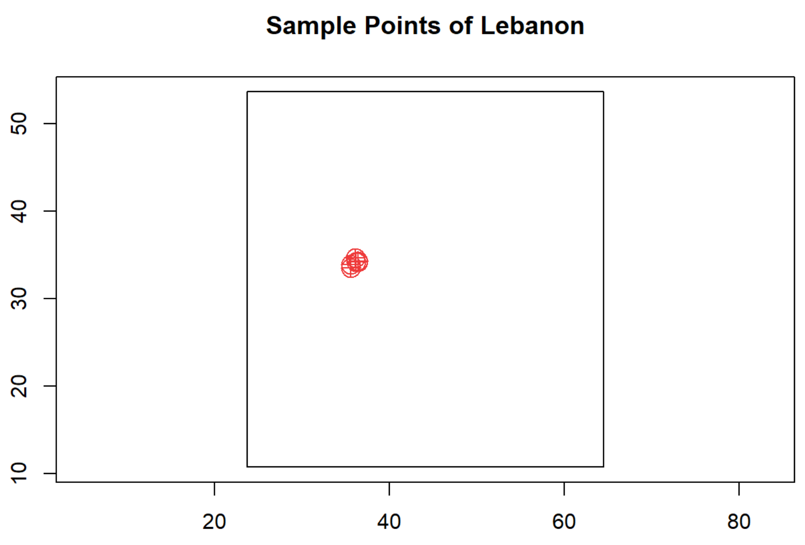
```
## $tstat
## 0.2794889 [1]
##
## $pval
## 0.7798889 [1]
```

We can see that the $p - value > 0.1 = \alpha$ for which we fail to reject the null hypothesis as the estimated land area of Iceland and Cuba are not statistically significantly different. This result is different from the two-sample t-test performed in the previous project for the first algorithm.

Bonus Countries

In this section we find the estimate and confidence bound of three smaller countries which we tested in the first project:

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

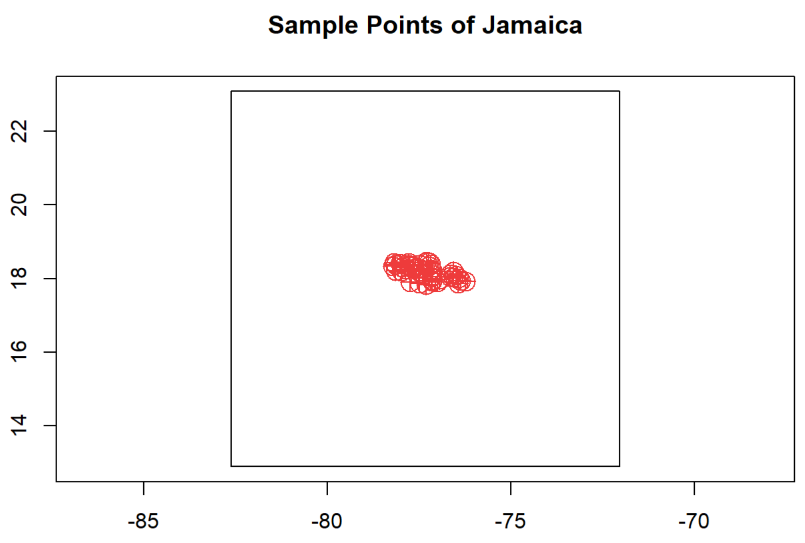


```
lebanon
```

```
## $mean
## 26839.72 [km^2]
##
## $variance
## 119911617 [km^4]
##
## $ci
## Units: [km^2]
## [1]  8827.885 44851.549
```

We note that increasing the sample size in phase 5 to $n = 4,000$ decreases the variance, and also improves the estimation of smaller countries. In this case we can see that the estimated mean area of $26,839.72\ km^2$ is slightly higher to the true area of Lebanon $(10,452\ km^2)$ and the 90% confidence bound of $8,827.885\ km^2$ to $44,851.549\ km^2$ captures the true area.

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

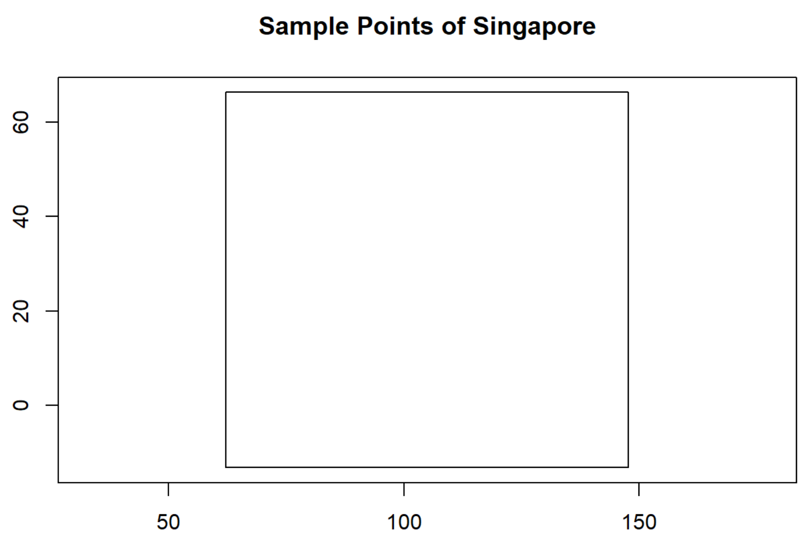


```
jamaica
```

```
## $mean
## 13324.61 [km^2]
##
## $variance
## 4183925 [km^4]
##
## $ci
## Units: [km^2]
## [1]  9960.117 16689.096
```

We can see that the estimated mean area of $13,324.61\ km^2$ is slightly higher to the true area of Jamaica $(10,991\ km^2)$ and the 90% confidence bound of $9,960.117\ km^2$ to $16,689.096\ km^2$ captures the true area.

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



```
singapore
```

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

For Singapore we can see that just like the previous project, the algorithm still cannot estimate its area. Looking at the graph we can see that the bounds of latitude and longitude are still too big to capture the country. Using a larger sample size would be computationally inefficient, but there might be another method to capture the country in smaller bounds to get an estimate.

Conclusion

Like the previous project, Monte-Carlo estimation sampling with a uniform distribution was able to achieve estimates close to the true area of a chosen country. Using volume integration instead of using a sinusoidal projection on the latitude and longitude bounds improved the estimates. In the previous project, the algorithm underestimated the area of Canada, China, Iceland, Cuba, and Lebanon and overestimated the area of Jamaica. The algorithm with the new changes still underestimates slightly the area of Canada and China, but it closely estimates the true area of Colombia, Egypt, Iceland, Cuba, Jamaica, and Lebanon. The 90% confidence bounds also capture the true area of these countries and all bounds are positive. For both versions of the algorithm, we note that Singapore cannot be estimated.

An improvement to the code would be finding ways to increase the number of points sampled while maintaining the efficiency of the code. Slow-loop iteration used in the code of the algorithm limited the number of samples we could achieve. Another improvement could be adding an extra layer to get smaller bounds that capture small countries like Singapore.

Code

```r
#Project R code Milagros N. Cortez

set.seed(2023)
library(sp)# package that provides spatial data in R1
library(rworldmap) #has functions that match coordinates to locations on the map
library(ggplot2) #graphing purposes I guess? (Haven't done any graphing yet)
library(countrycode) #has a function that matches country to continent
library(sf)
library(units)# function to add units

# 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
}
# Function to match country to region
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. 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 adjusted Beta(2,2) and Uniform(l
oL, 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)
}

# 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.case = 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=="Greenland")){
        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.case = T) & (country!="Greenla
nd") == T){
        # Given that a country is not Greenland, and is in N or S America. If the country is either Canada or the US we stor
e 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("United", 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)
  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 continent.
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 boundaries 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){
```

```r
    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 = 1000
  # 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,
      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 = "continent")
  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 coordArea.
  earthrad = 6371 # earth's radius in km
  area = (earthrad^2)*(sr4$loU-sr4$loL)*(pi/180)*(sin(sr4$laU*(pi/180))-sin(sr4$laL*(pi/180))) # area of a spherical rectang
le using latitude and longitude in radians
  conv = set_units(area, 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 = 4000
  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) {
      long = c(long, points$long)
      lat = c(lat, points$lat)
      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 the 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){
        long = c(long, points$long)
        lat = c(lat, points$lat)
        bin = c(bin, 1)
      }
      else{
        long = c(long, points$long)
        lat = c(lat, points$lat)
        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$bin)# number of points that fall in the country of interest
  fraction = points/nrows #proportion of points that fall in the country of interest
  estAr = area*fraction # Estimated area
  estV = (area^2)*fraction*(1-fraction)/(nrows-1) # estimated variance
  normCI = c(estAr-qnorm(1-alpha/2)*sqrt(estV), estAr+qnorm(1-alpha/2)*sqrt(estV)) # confidence bounds (normal)
  return(list(
    mean = estAr,
    variance = estV,
    ci = normCI
  ))
}
# function that gives the graphics when visualize = T. Shows the polygon and sampled points that fall in the country of inte
rest.
graphics = function(coordinates, df, name){
  p1 = c(coordinates$loL,coordinates$laU)
  p2 = c(coordinates$loL,coordinates$laL)
  p3 = c(coordinates$loU,coordinates$laL)
  p4 = c(coordinates$loU,coordinates$laU)
  lat = c(p1[2], p2[2], p3[2] ,p4[2], p1[2])
  long = c(p1[1], p2[1], p3[1] ,p4[1], p1[1])
  polygon = st_sfc(st_polygon(list(cbind(long, lat)))) %>% st_set_crs(32615)
  onedf = df[df$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)
}

# two sample t-test function
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
  ))
}
```