The aim of this project is to build a Metropolis-Hastings algorithm to identify our position on a map using the position resection method.
Position resection is a method for determining an unknown geographic position by measuring bearing angles with respect to known positions. Therefore, a point with unknown position is occupied and sightings are taken for three known points. In our case, we are somewhere south-west of Milton Keynes, England, as depicted in Figure 1. The three following points and their bearings have been chosen:

- Bletchley Park in Bletchley (longitude = -0.7449436, latitude = 51.99919) with a bearing of -179.95.

- Towcester Centre for Leisure in Towcester (longitude = -1.0074497, latitude = 52.1318862) with a bearing of -64.86.

- Cranfield University in Cranfield (longitude = -0.6300633, latitude = 52.0694635) with a bearing of 64.04.

As we can observe in Figure 1, we have plotted the reference points chosen along with their bearing lines and we have obtained a triangle. Our position should be situated within the depicted triangle.

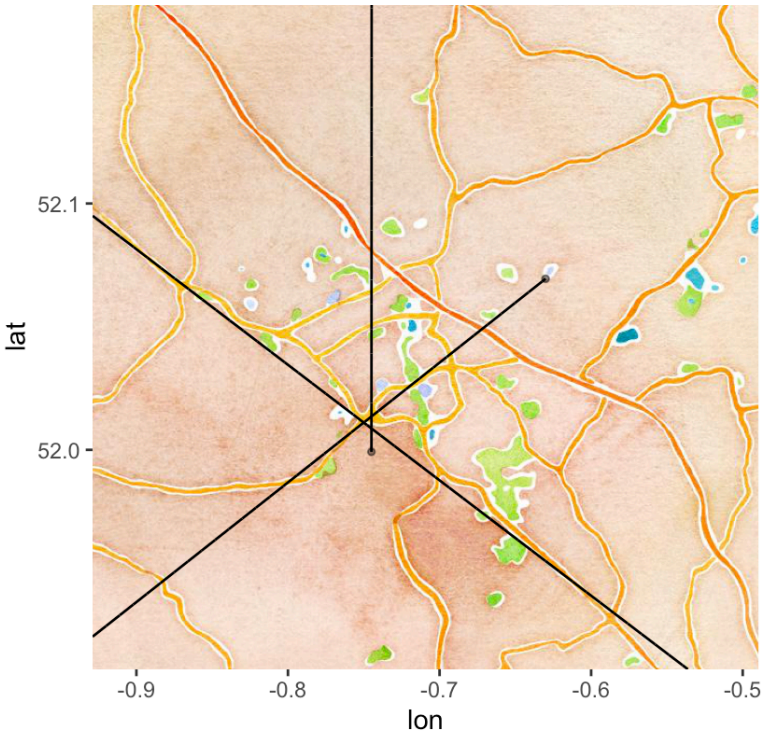

Figure 1. Depicts the triangle obtained by plotting the bearing lines on the map.

The aim of the project is to be precise about our position, and we want to study the uncertainty of our result. Therefore, we want to obtain the complete distribution of our plausible locations on the map. In order to do so, we will need build a statistical model in order to describe the setting of our problem. Therefore, λ will be the longitude and ϕ will be the latitude for our position, and they will both be treated as random variables. Furthermore, we will also infer our variance $\underline{\sigma}^2$ alongside them. Our three bearings will be angles α, β and γ, and they will be our observed data. They have been measured by using the function *bearing* from the package *geosphere.* Also, we will assume that our measurements are unbiased, and so the mean of errors is equal to zero. We don't know the variance of these error, hence why we infer $\underline{\sigma}^2$ too.

We are also given a formula to calculate the bearing angle between two points P1 and P2, considering that we know their coordinates, as follows:

$$B(P1, P2) = 360 + \arctan\ (\phi 2 - \phi 1 \,/\, \lambda 2 - \lambda 1) \cdot 180\ \pi \cdot \text{modulo } 360$$

In the above formula $\lambda 1$ is the longitude and $\phi 1$ is the latitude for P1, whereas $\lambda 2$ and $\phi 2$ are the latitude and longitude for P2 respectively.

In order to identify our position we will have to work within the Bayesian framework, which uses probabilities to quantify belief, and these beliefs are updated when observing new evidence.

Since scientific hypotheses are quantified with parametric statistical models, they impose likelihoods of the form $p(D|\theta)$, where D is the data we have observed - in our case the bearings α, β and γ - and θ is the parameter of the model that answers the question of interest. In our problem the parameters that answer the question of interest are λ, ϕ and σ^2. The likelihood defines the probability of observing our data α, β and γ given that the model parameters are fixed at certain values.
Our initial belief can be described with a distribution of model parameters called the prior distribution. In our case the priors are $p(\lambda)$, $p(\phi)$ and $p(\sigma^2)$.
After observing our evidence α, β and γ, we will update our belief about λ, ϕ and σ^2 to the posterior distribution $p(\lambda, \phi, \sigma^2 \mid \alpha, \beta, \gamma)$. The way we update our belief is by using Bayes' Theorem:

$$p(\lambda, \phi, \sigma^2 \mid \alpha, \beta, \gamma) = p(\alpha, \beta, \gamma \mid \lambda, \phi, \sigma^2) \times p(\lambda) \times p(\phi) \times p(\sigma^2) / p(\alpha, \beta, \gamma) =$$
$$p(\alpha, \beta, \gamma \mid \lambda, \phi, \sigma^2) \times p(\lambda) \times p(\phi) \times p(\sigma^2) / \iiint p(\alpha, \beta, \gamma \mid \lambda, \phi, \sigma^2) \times p(\lambda) \times p(\phi) \times p(\sigma^2)\, d\lambda d\phi d\sigma$$

The most tricky part is evaluating the integral in the denominator of Bayes' Theorem. Since it cannot be evaluated analytically, we will resort to the Metropolis-Hastings algorithm, that will allow us to sample directly from the posterior.

<u>Defining our problem</u>

Now, since the bearings α, β and γ of the three locations were measured independently, we can use the following likelihood:

$$\begin{aligned} p(\alpha, \beta, \gamma|\lambda, \phi, \sigma 2) = & N\ (\alpha|B((\lambda, \phi), (\lambda_1, \phi_1)), \sigma^2) \times \\ & N\ (\beta|B((\lambda, \phi), (\lambda_2, \phi_2)), \sigma^2) \times \\ & N\ (\gamma|B((\lambda, \phi), (\lambda_3, \phi_3)), \sigma^2) \\ = & 1 \sqrt{2\pi\sigma^2} \exp(-(\alpha - B((\lambda, \phi), (\lambda_1, \phi_1)))^2 / 2\sigma^2) \times \\ & 1 \sqrt{2\pi\sigma^2} \exp(-(\beta - B((\lambda, \phi), (\lambda_2, \phi_2)))^2 / 2\sigma^2) \times \\ & 1 \sqrt{2\pi\sigma^2} \exp(-(\gamma - B((\lambda, \phi), (\lambda_3, \phi_3)))^2 / 2\sigma^2) \end{aligned}$$

We can see that our likelihood is the product of three Gaussian distributions, our angles α, β and γ are the observed data, and we are interested in the probability of observing them under the assumption of B(P1, P2) which is our μ and σ^2, which is our variance.
Furthermore, (λ_1, ϕ_1) = (-0.7449436, 51.99919) and it's Bletchey Park in Bletchley; (λ_2, ϕ_2) = (-1.0074497, 52.1318862) and it is Towcester Centre for Leisure in Towcester; (λ_3, ϕ_3) = (-0.6300633, 52.0694635) and it is Cranfield University in Cranfield.

Inference using wide uniform priors

In this part of the project we will not introduce a lot of information as part of our priors. We will define our priors on λ, ϕ and σ^2 by assuming wide uniform priors for λ and ϕ, and an exponential prior for σ^2:

$\lambda \sim$ Uniform(−1.2, -0.2)

$\phi \sim$ Uniform(51.5, 52.5)

$\sigma \sim$ Exponential(20)

Since we have no reason to justify dependency between λ, ϕ and σ^2, we will keep these priors independent, therefore our prior will be defined in the following way:

$p(\lambda) \times p(\phi) \times p(\sigma^2) = 1/(-0.2 - (-1.2)) \times 1/(52.5 - 51.5) \times (20 \times \exp(-20x))$

Considering the information outlined above and knowing the formula *posterior = likelihood x prior*, the posterior $p(\lambda, \phi, \sigma 2 \mid \alpha, \beta, \gamma)$ will look in the following way down to proportionality:

$$p(\lambda, \phi, \sigma 2 \mid \alpha, \beta, \gamma) \propto 1 \sqrt{2\pi\sigma^2} \exp\left(- (\alpha - B((\lambda, \phi), (\lambda_1, \phi_1))) ^2 / 2\sigma^2 \right) \times$$
$$1 \sqrt{2\pi\sigma^2} \exp\left(- (\beta - B((\lambda, \phi), (\lambda_2, \phi_2))) ^2 / 2\sigma^2 \right) \times$$
$$1 \sqrt{2\pi\sigma^2} \exp\left(- (\gamma - B((\lambda, \phi), (\lambda_3, \phi_3))) ^2 / 2\sigma^2 \right) \times$$
$$1/(-0.2 - (-1.2)) \times 1/(52.5 - 51.5) \times (20 \times \exp(-20x))$$

Since the posterior cannot be evaluated analytically, we will rely on the Metropolis-Hastings algorithm to perform the evaluation and then we will sample from the stationary distribution.

Metropolis - Hastings Algorithm

The Metropolis-Hastings algorithm falls into the class of Markov Chain Monte Carlo methods. They are a class of algorithms used from sampling from complex probability distributions where the integral in the denominator cannot be evaluated analytically. In practice, an ensemble of Markov chains is developed, starting from arbitrary points. These chains are stochastic processes, which move around randomly looking for steps to be a reasonably high contribution to the integral to move into next, assigning them higher probabilities. The Metropolis-Hastings algorithm constructs a Markov chain that converges to a stationary distribution and ensures that this stationary distribution is our posterior $p(\lambda, \phi, \sigma^2 \mid \alpha, \beta, \gamma)$.

Next, we apply the Metropolis-Hastings algorithm to our problem. We will work with the logarithm of probabilities, therefore alpha has been set to min(0,r), where r is the acceptance ratio. We will use the wide uniform priors outlined above.

At first attempt, I used the formula given in the project for calculating the bearing angle between two points - B(P1, P2) - and incorporated it in the log-likelihood. The algorithm was initialised from the intersection of the first and second bearing lines, instead of initialising it randomly from the prior.

Findings

I have used three separate chains, and in each chain we draw 4000 samples for longitude, latitude and variance and each of them is added one by one until the Gelman-Rubin convergence criterion is achieved. This will indicate the end of the burn-in period.

For the proposal distribution, I have used multivariate normal with the following variance-covariance matrix:

```
      [,1]  [,2]  [,3]
[1,] 1e-08 0e+00 0e+00
[2,] 0e+00 1e-08 0e+00
[3,] 0e+00 0e+00 1e-04
```

After a few iterations, with acceptance rate being 55.34%, we have arrived to the state of the chains depicted in Figure 2.

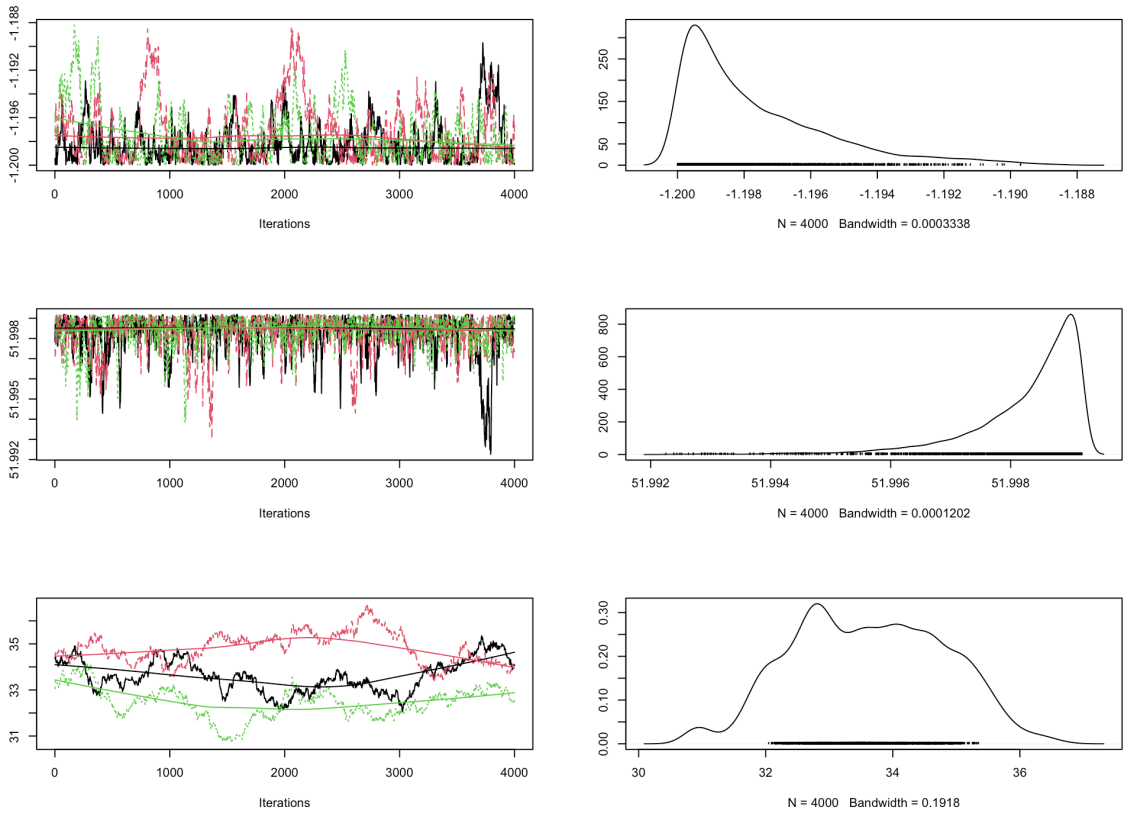

Figure 2. State of the chains from the first run of the Metropolis-Hastings algorithm.

We can see from the plots that the algorithm hasn't yet converged, but is is somewhat close for the longitude (first plot) and the latitude (second plot). The variance (third traceplot) is still very unstable, we can observe a trend for all three chains.

I have also used the Gelman-Rubin diagnostic for convergence, and it confirms the fact that the chains are yet to converge.

As a solution, I have taken the covariance of the samples collected in the first run of the algorithm and used it as the variance-covariance matrix in the proposal distribution. I initialised the algorithm near the mode of the posterior, at the intersection of the first two bearing lines. Once I performed this, I achieved convergence, from the first run of the Metropolis-Hastings algorithm, according to the Gelman-Rubin diagnostic, with an acceptance rate of 45.04 %. The acceptance rate is suitable for our multi-parameter model. The state of the chains and the distribution of the possible values of the parameters are depicted in Figure 3.

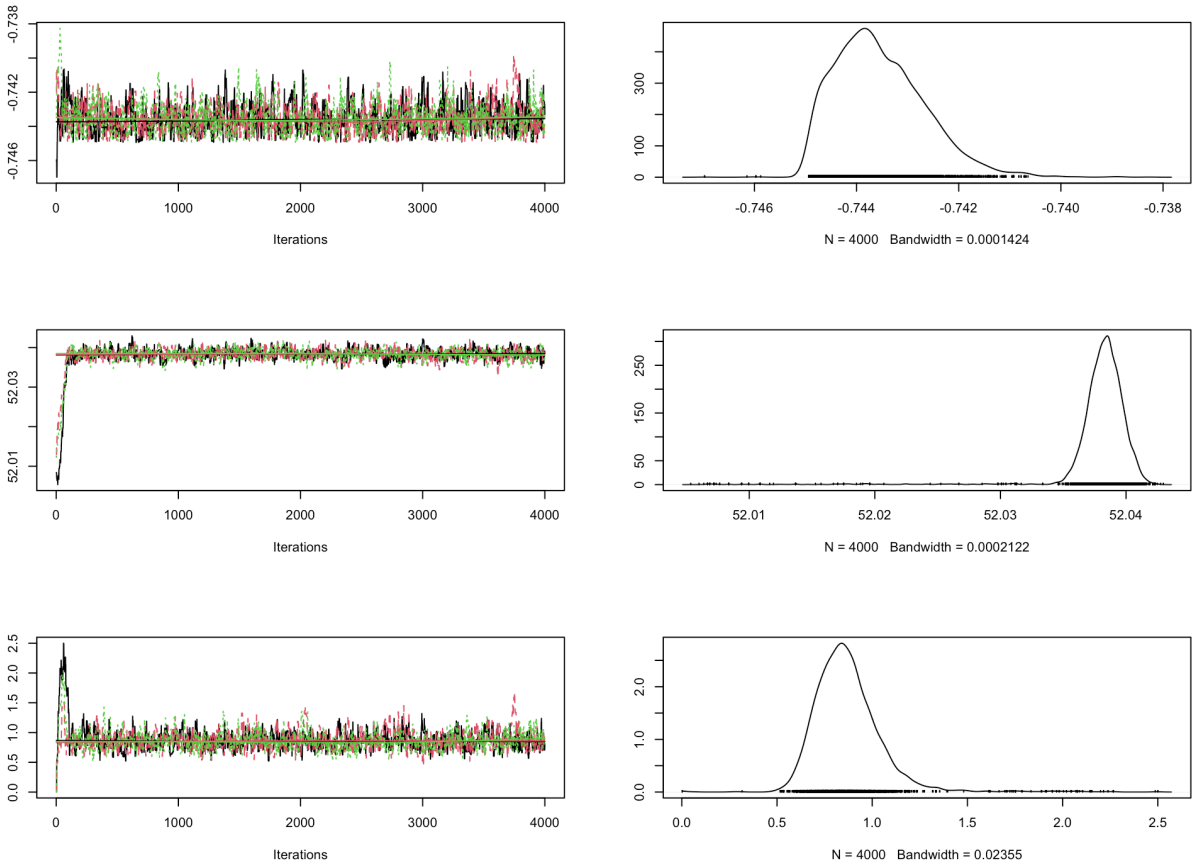

Figure 3. The Metropolis-Hastings algorithm has achieved convergence.

We can see that the trace plots appear to become stationary after 200 iterations or so. Now that convergence has been achieved, we begin to collect the final sample from the posterior distribution. Once again, we have three chains, and this time we collect 10000 samples for each parameter. The algorithm is initialised from the last samples collected previously when we achieved convergence. The acceptance rate for the final sample is 28.19%, which, once again, it is suitable for data.

The means for our parameters present in the following way: for our longitude, λ, the mean is 51.99824, for our latitude, ϕ, the mean is -1.1976 and the mean for the variance σ^2 is 33.81294. What we want to do now is visualise our posterior on the map, as depicted in Figure 4. As we can see, the posterior is shown somewhere in Croughton, which is approximately 21 miles from the triangle we obtained by plotting the bearing lines. This is not a great result, so I considered ways to improve it, such as tinkering with the priors and

making them more narrow for the values of λ and ϕ, however this did not work as expected.

The next step I took to improve the results was to change the log-ikelihood. Instead of using the formula given in the project description to calculate the bearings, I used the function *bearing* from package *geosphere.*

Once again I ran the

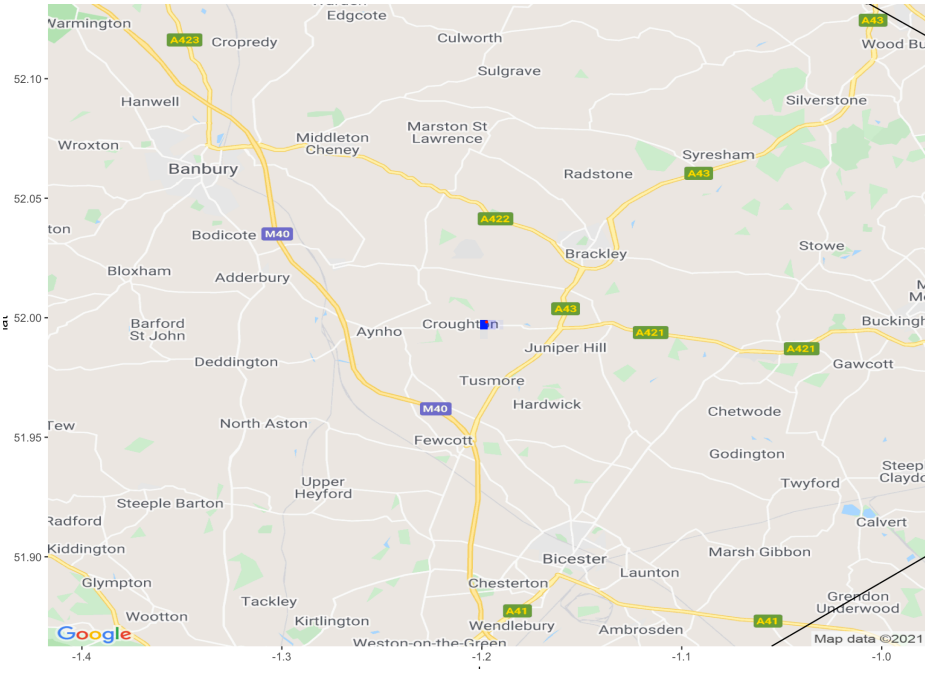

Figure 4. Our posterior shown on the map.

Metropolis-Hastings algorithm a few times in the same setup as before and I used the covariance of the samples as the variance-covariance matrix in the multivariate normal proposal distribution. The chains achieved convergence very quickly, from the first run, with a 43.3% acceptance rate. Figure 5 shows the plots of the chains, where we can see

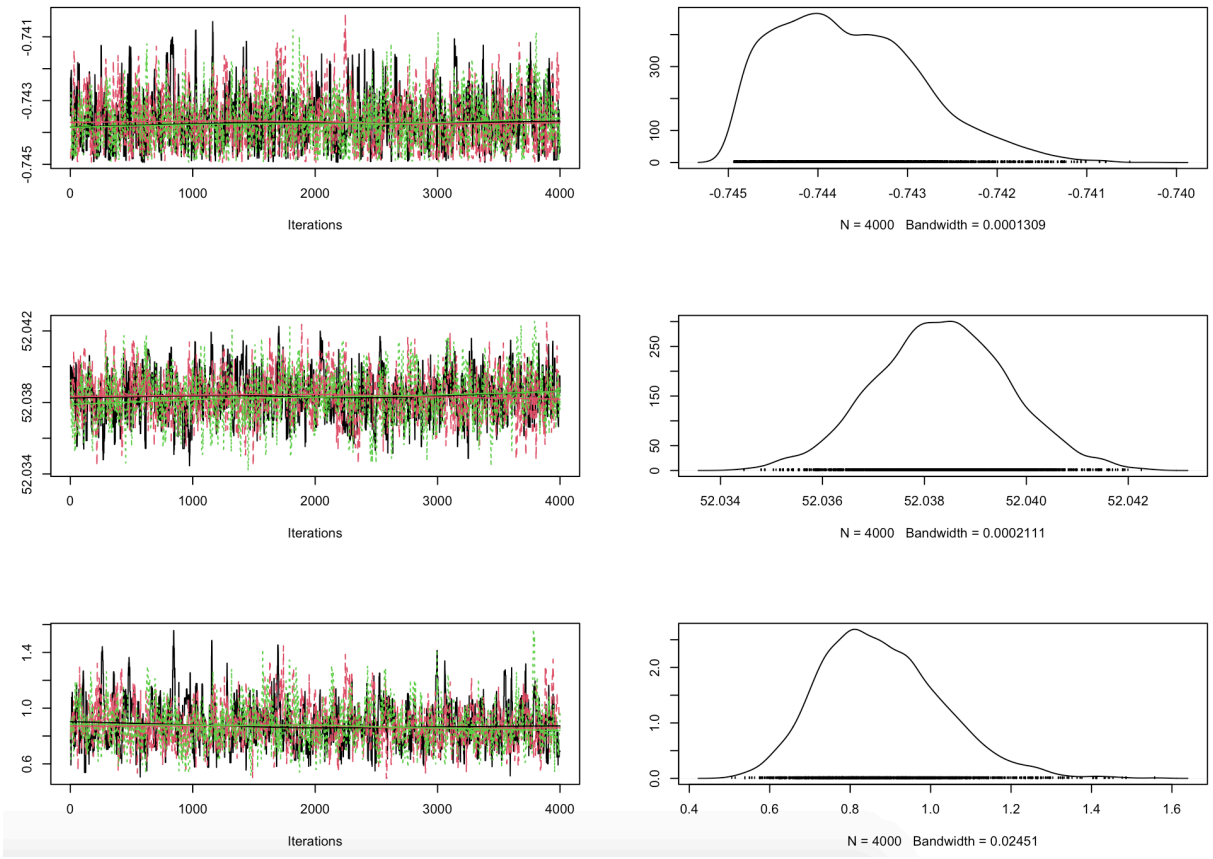

Figure 5. Shows that the chains have achieved convergence

that they have indeed achieved convergence, along the distribution of possible values for the parameters.
After I achieved convergence, I once again obtained the final sample from the posterior distribution. I used three chains and obtained 10000 samples for each parameter. The acceptance rate was 43.42%, and the mean for the longitude was 52.03838, for latitude the mean was -0.7435957 and for the variance it was 0.8774624, which is very different from the results we obtained previously, especially in the case of variance, which is much lower.

Next, I plotted the posterior on the map, which can be seen in Figure 6.

The location of the posterior is still outside the triangle, but much closer, only off by 3 miles, compared to 21, which was the case before.

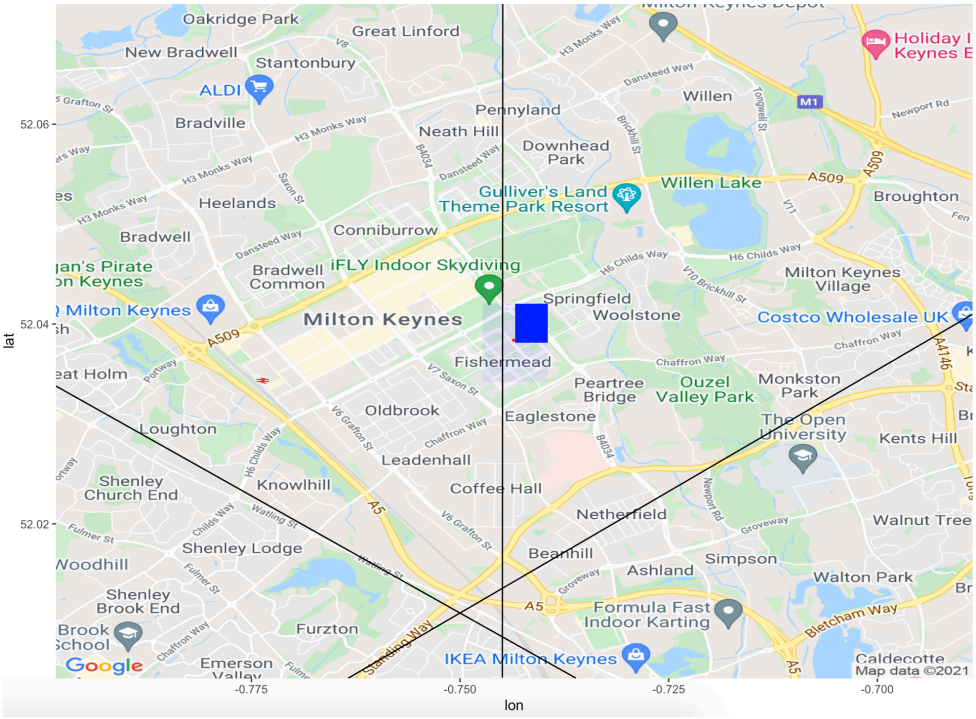

Figure 6. Posterior plotted on the map

Unfortunately, the results are not as expected in the project description, where it states that the location should be somewhere around the upper corner of the triangle. I have tried to replicate those results numerous times, but this is the best posterior distribution I have been able to achieve.

With that being said, we move forwards to the next part of the project, in which we assume proximity to a road as part of our prior.

Inference assuming proximity to a road

For the second part of the project we assume that we are standing on a road, and we introduce this information as part of our prior distribution. We therefore have the following priors:

$p(\lambda, \phi) \propto \mathrm{N}(\rho(\lambda, \phi)|0, 6^2)$

$\sigma^2 \sim \text{Exponential}(20)$

In this case $\rho(\lambda, \phi)$ is the shortest distance to the middle of the nearest road, and it follows a normal distribution with mean zero and standard deviation of 6. We maintain the same exponential prior for our variance, σ^2. We will keep these two priors independent, therefore our prior will look in the following way:

$p(\lambda, \phi) \times p(\sigma^2) = 1 \sqrt{2\pi x 6^2} \exp\left(- \left(\rho(\lambda, \phi) - 0 \right)^2 / 2x6^2 \right) \times (20 \times \exp(-20x))$

Then we have our posterior:

$p(\lambda, \phi, \sigma 2 \mid \alpha, \beta, \gamma) = p(\alpha, \beta, \gamma \mid \lambda, \phi, \sigma 2) \times p(\lambda, \phi) \times p(\sigma^2) / p(\alpha, \beta, \gamma)$

$= p(\alpha, \beta, \gamma \mid \lambda, \phi, \sigma 2) \times p(\lambda, \phi) \times p(\sigma^2) / \iiint p(\alpha, \beta, \gamma \mid \lambda, \phi, \sigma^2) \times p(\lambda, \phi) \times p(\sigma^2) \, d\lambda d\phi d\sigma$

Findings

Next, we apply the Metropolis-Hastings algorithm to our problem. I have once again used three chains, and have drawn 4000 samples for each parameter in each chain and samples are added in one by one until the Gelman-Rubin criterion is satisfied. The algorithm was initialised from the intersection of the first two bearing lines. I have used the Gelman-Rubin criterion in order to monitor convergence. For the proposal distribution I have used the multivariate normal, with the variance-covariance matrix given in the project description.
After a very slow calculation, I obtained the results depicted in Figure 7. Going by the Gelman-Rubin criterion, the chain had achieved convergence, however upon visual inspection of the plots of the chain I disagree. It does not look like the plots have reached a stationary distribution for λ and ϕ, as there appears to be a trend. On the other hand, the plot of the variance chains seemed to have achieved convergence. Acceptance rate for these chains was 21%, right at the limit.

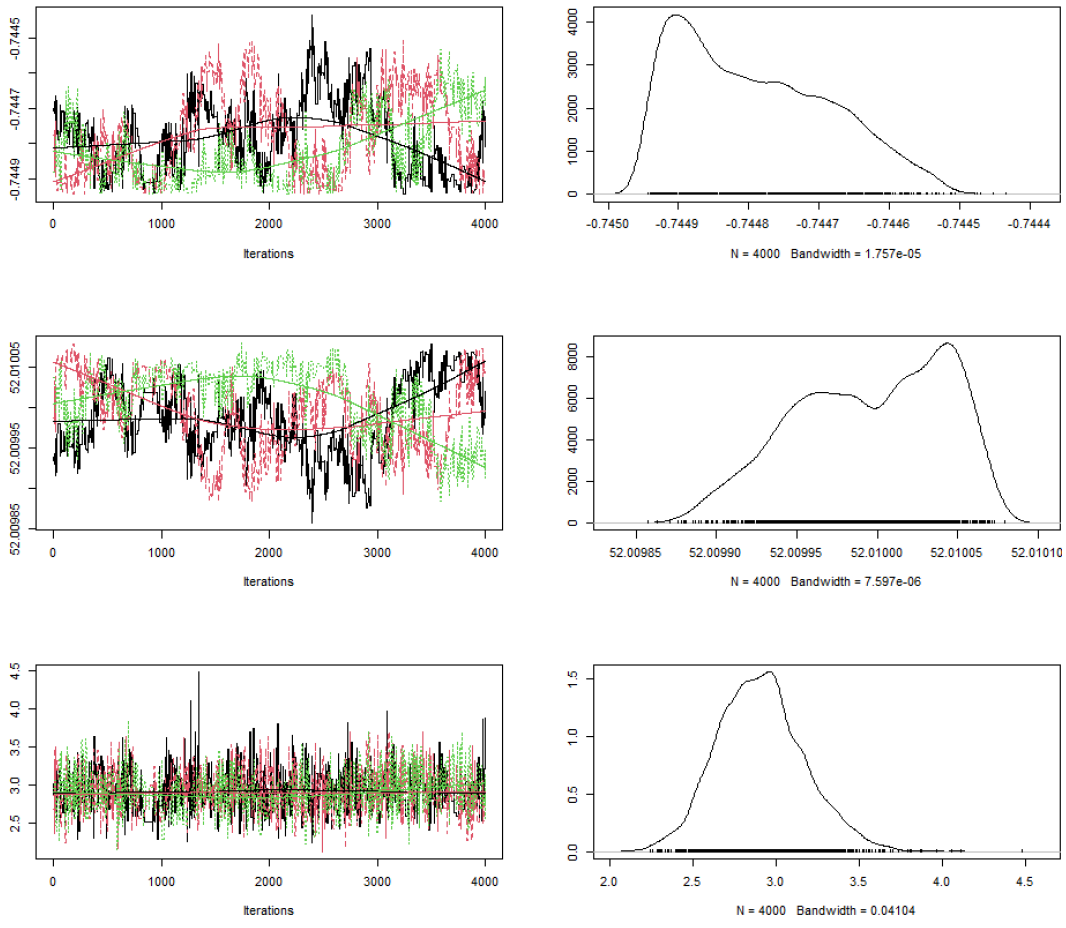

Figure 7. The chains for longitude and latitude have not achieved convergence

Next, I have calculated the covariance of these samples and used it as the variance-covariance matrix of the proposal distribution. I ran the algorithm again, and it achieved convergence very quickly according to the Gelman-Rubin diagnostic. Figure 8 shows that all the chains have achieved convergence for all three parameters and it depicts their distributions. Furthermore, the acceptance rate for the chains was 51.48%.

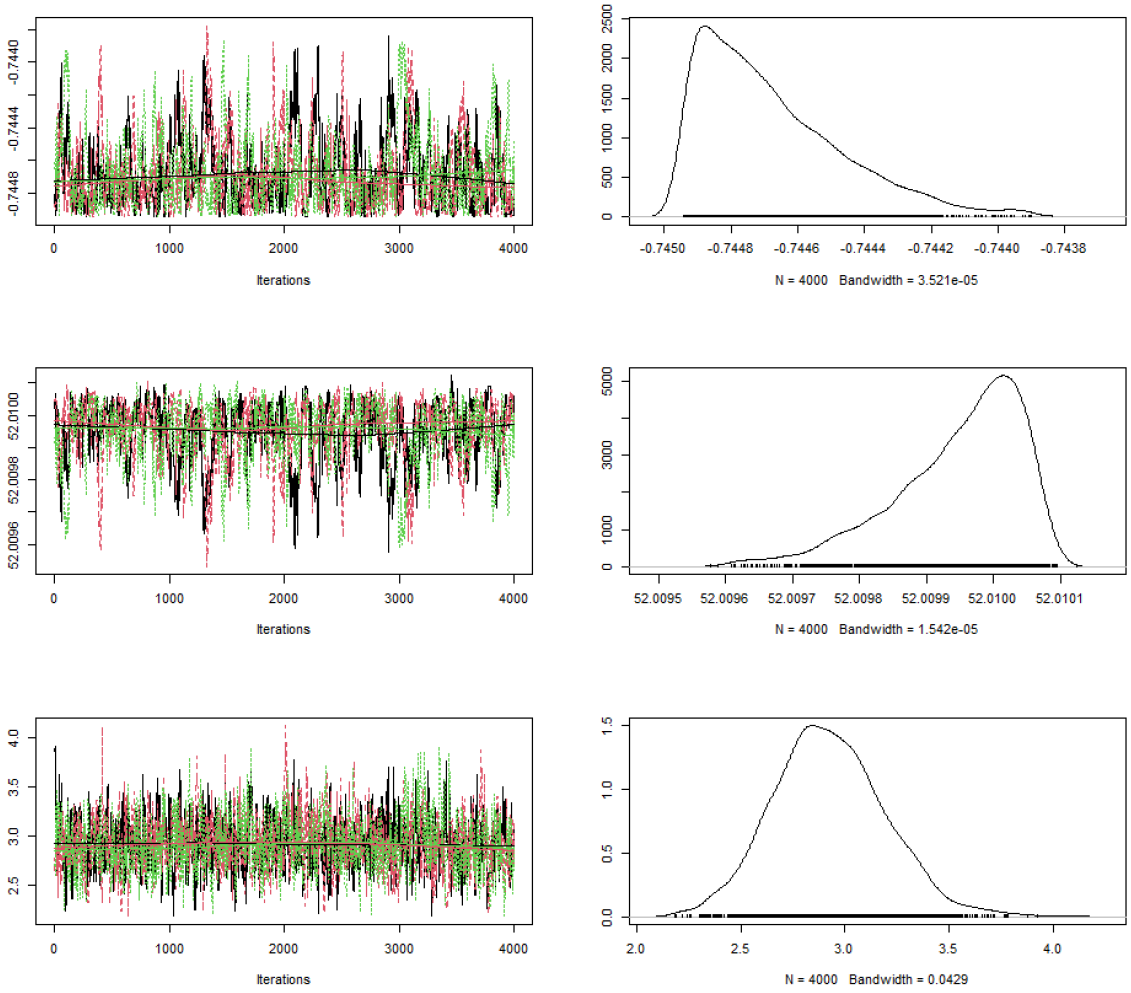

Figure 8. Shows the chains at convergence.

Next, after achieving convergence, we want to collect our final sample. We will have three chains, and we will collect 10000 samples for each parameter. The algorithm will be initialised from the last samples drawn from the stationary distribution. Once the final sample is collected, we can see that the acceptance rate is 51.1%. The mean for the longitude is -0.74468 and for latitude it is 52.00995, whereas the mean for variance is 2.94.

Now, we want to see our posterior plotted on a map, as depicted in Figure 9. We can see the red dot that represents our location is right outside the triangle, on Watling Street. However, the distribution of possible locations covers the entire triangle and some of the are outside it.

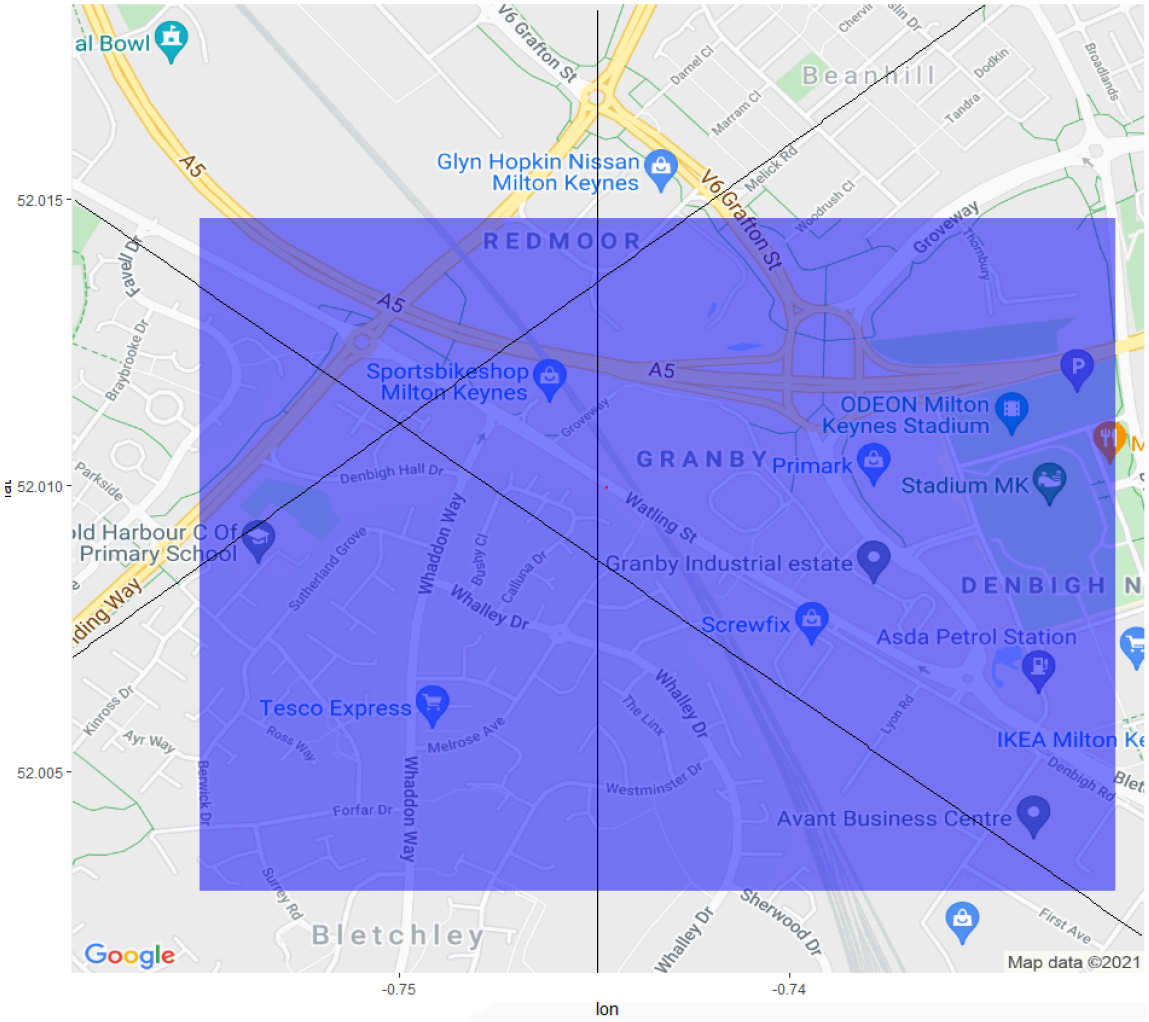

Figure 9. Posterior distribution of our location plotted on the map.

Conclusion

In this project I have used the Metropolis-Hastings algorithm and have experimented with two different priors in order to identify our position on the map with the position resection method.

First, I used a wide uniform prior for the longitude and latitude and an exponential prior for the variance. The likelihood was the product of three normal distributions, and contained our data, the bearing angles I measured. I used two different formulas for calculating the bearing angles, once the formula given in the project, and the second time the function *bearing* from the package *geosphere,* and obtained different results. The first time, the posterior location was 21 miles away from the triangle, which was highly inaccurate. The second time, the posterior was located 3 miles away from the triangle, which is an improvement. However, it is disappointing that neither of the attempts located our position with more accuracy. I suspect that the lack of information provided as part of the priors might have played a role in this.

For the second part of the project, I have used a prior that assumed that we were standing on a road, and calculated the distance to the middle of the nearest road. The results were more accurate in this case, however what concerns me is that the posterior covers such a wide area which much expands outside the triangle. The exact coordinates

were accurate, they gave the exact position on a road near the triangle, which is satisfying.

In conclusion, the prior that assumed a position on the road offered more information and therefore, the posterior was more accurate in identifying our position on the map.

Appendix

```
require(osmar)
require(geosphere)
require(ggmap)
require(mvtnorm)
require(coda)
require(reshape2)
require(MASS)
require(ggplot2)
require(osmdata)
require(magrittr)
require(sf)
```

Part 1

```
register_google(key="AIzaSyAnGAw53RJtWLiJ76HKtnxarZK2VR4GL5s",write=TRUE)
map <- get_map(c(-0.70956,52.04623),zoom=11,maptype="road")
ggmap(map)

landmarks<-data.frame(lon=c(-0.7449436,-1.0074497, -0.6300633),lat=c(51.99919,
52.1318862,52.0694635))
alpha <- bearing(c(-0.70956,52.04623), c(landmarks[1,1], landmarks[2,1] ))
beta <- bearing(c(-0.70956,52.04623), c(landmarks[2,1], landmarks[2,2] ))
gamma <- bearing(c(-0.70956,52.04623), c(landmarks[3,1], landmarks[2,3] ))

d <- seq(0,3,0.0001) # Length of the line
line1 <- data.frame(lon=landmarks[1,1] + d*sin(alpha*pi/180+pi),
                    lat=landmarks[1,2] + d*cos(alpha*pi/180+pi))
line2 <- data.frame(lon=landmarks[2,1] + d*sin(beta*pi/180+pi),
                    lat=landmarks[2,2] + d*cos(beta*pi/180+pi))
line3 <- data.frame(lon=landmarks[3,1] + d*sin(gamma*pi/180+pi),
                    lat=landmarks[3,2] + d*cos(gamma*pi/180+pi))

map <- get_map(c(-0.70956,52.04623),zoom=11,maptype="watercolor")
mapPlot <- ggmap(map)+
  geom_point(aes(x = lon, y = lat), size = 1, data = landmarks, alpha = .5) +
  geom_line(aes(x=lon,y=lat),data=line1) +
  geom_line(aes(x=lon,y=lat),data=line2) +
  geom_line(aes(x=lon,y=lat),data=line3)
mapPlot

intersectBearings <- function(p1,b1,p2,b2) {
  x1 <- p1[1]
```

```
  x2 <- p1[1] + 0.1*sin(b1*pi/180)
  x3 <- p2[1]
  x4 <- p2[1] + 0.1*sin(b2*pi/180)
  y1 <- p1[2]
  y2 <- p1[2] + 0.1*cos(b1*pi/180)
  y3 <- p2[2]
  y4 <- p2[2] + 0.1*cos(b2*pi/180)
  x <- ((x1*y2-y1*x2)*(x3-x4)-(x1-x2)*(x3*y4-y3*x4))/((x1-x2)*(y3-y4)-(y1-y2)*(x3-x4))
  y <- ((x1*y2-y1*x2)*(y3-y4)-(y1-y2)*(x3*y4-y3*x4))/((x1-x2)*(y3-y4)-(y1-y2)*(x3-x4))
  return(as.numeric(c(x,y)))
}


intersectBearings(landmarks[1,],alpha,landmarks[2,],beta)

intersection <- intersectBearings(landmarks[1,],alpha,landmarks[2,],beta)

bearings <- function(lambda1, phi1, lambda2, phi2) {
  (360 + atan2(phi2 - phi1, lambda2 - lambda1) * 180/pi) %% 360
}

#likelihood <- function(theta, alpha, beta, gamma) {
  #if (theta[3] <= 0){
   # return (log(0))
 # }
 # else {
  #  return (dnorm(alpha, bearings(theta[1], theta[2],landmarks[1,1], landmarks[1,2]),
#sqrt(theta[3]), log=T) +
 #             dnorm(beta, bearings(theta[1], theta[2],landmarks[2,1], landmarks[2,2]),
#sqrt(theta[3]), log=T) +
 #             dnorm(gamma, bearings(theta[1], theta[2],landmarks[3,1], landmarks[3,2]),
#sqrt(theta[3]), log=T))
  }
}

likelihood <- function(theta, alpha, beta, gamma) {
  if (theta[3] <= 0){
    return (log(0))
  }
  else {
    return (dnorm(alpha, bearing(c(theta[1], theta[2]), c(landmarks[1,1], landmarks[1,2])),
sqrt(theta[3]), log=T) +
              dnorm(beta, bearing(c(theta[1], theta[2]), c(landmarks[2,1], landmarks[2,2])),
sqrt(theta[3]), log=T) +
              dnorm(gamma, bearing(c(theta[1], theta[2]), c(landmarks[3,1], landmarks[3,2])),
sqrt(theta[3]), log=T))
  }
}

prior <- function(theta) {
  sum(dunif(theta[1], -1.2, -0.2, log = T) +
        dunif(theta[2], 51.5, 52.5, log=T) +
```

```
      dexp(theta[3], 20, log=T))
}


draws <- array(0,dim=c(4000,3,3))
draws[4000,1,] <- runif(3,intersection[1] - 0.01, intersection[1] + 0.01)
draws[4000,2,] <- runif(3,intersection[2] - 0.01, intersection[2] + 0.01)
draws[4000,3,] <- rexp(3,20)

prop.cov <- c(1e-8,1e-8,1e-4)*diag(3)

converged <- FALSE
while (!converged) {
  draws[1,,] <- draws[4000,,]
  accepted <- 1
  for (step in 2:4000) {
    for (chain in 1:3) {
      proposed <- rmvnorm(1,draws[step-1,,chain],prop.cov)
      r <- likelihood(proposed, alpha, beta, gamma) + prior(proposed) -
        (likelihood(draws[step-1,,chain], alpha, beta, gamma ))-prior(draws[step-1,,chain])
      a <- min(0,r)
      u <- runif(1)
      if (log(u) < a) {
        draws[step,,chain] <- proposed
        accepted <- accepted + 1
      } else {
        draws[step,,chain] <- draws[step-1,,chain]
      }
    }
  }
  print(sprintf("Acceptance rate: %f",accepted/120))
  chainlist <- mcmc.list(Chain1=mcmc(draws[,,1]),
                         Chain2=mcmc(draws[,,2]),
                         Chain3=mcmc(draws[,,3]))
  converged <- all((gelman.diag(chainlist)$psrf[,2])<1.2)
  if (!converged) {
    if (accepted/120 <= 20) {
      prop.cov <- prop.cov * 0.8
    }
    if (accepted/120 >= 60) {
      prop.cov <- prop.cov * 1.5
    }
  }
  plot(chainlist) # This plots current state of the chains
  Sys.sleep(0.1)
}

s1 <- rbind(draws[,,1], draws[,,2], draws[,,3])
vc <- cov(s1)
vc

draws <- array(0,dim=c(4000,3,3))
```

```
draws[4000,1,] <- runif(3,intersection[1] - 0.01, intersection[1] + 0.01)
draws[4000,2,] <- runif(3,intersection[2] - 0.01, intersection[2] + 0.01)
draws[4000,3,] <- rexp(3,20)

converged <- FALSE
while (!converged) {
  draws[1,,] <- draws[4000,,]
  accepted <- 1
  for (step in 2:4000) {
    for (chain in 1:3) {
      proposed <- rmvnorm(1,draws[step-1,,chain],vc)
      r <- likelihood(proposed, alpha, beta, gamma) + prior(proposed) -
        (likelihood(draws[step-1,,chain], alpha, beta, gamma ))-prior(draws[step-1,,chain])
      a <- min(0,r)
      u <- runif(1)
      if (log(u) < a) {
        draws[step,,chain] <- proposed
        accepted <- accepted + 1
      } else {
        draws[step,,chain] <- draws[step-1,,chain]
      }
    }
  }
  print(sprintf("Acceptance rate: %f",accepted/120))
  chainlist <- mcmc.list(Chain1=mcmc(draws[,,1]),
                         Chain2=mcmc(draws[,,2]),
                         Chain3=mcmc(draws[,,3]))
  converged <- all((gelman.diag(chainlist)$psrf[,2])<1.2)
  if (!converged) {
    if (accepted/120 <= 20) {
      vc <- vc * 0.8
    }
    if (accepted/120 >= 60) {
      vc <- vc * 1.2
    }
  }
  plot(chainlist) # This plots current state of the chains
  Sys.sleep(0.1)
}

#since convergence is achieved we begin collecting the final sample from the posterior

sample <- array(0,dim=c(10000,3,3))
sample[1,,] <- draws[4000,,]
accepted <- 1

for (step in 2:10000) {
  for (chain in 1:3) {
    proposed <- rmvnorm(1,sample[step-1,,chain],vc)
    r <- likelihood(proposed, alpha, beta, gamma) + prior(proposed) -
      (likelihood(sample[step-1,,chain], alpha, beta, gamma ))-prior(sample[step-1,,chain])
    a <- min(0,r)
```

```
    u <- runif(1)
    if (log(u) < a) {
      sample[step,,chain] <- proposed
      accepted <- accepted + 1
    } else {
      sample[step,,chain] <- sample[step-1,,chain]
    }
  }
}
print(sprintf("Acceptance rate: %f",accepted/300))

#distance to a road

mean(sample[,2,])
mean(sample[,1,])
mean(sample[,3,])

D <- kde2d(as.vector(sample[,1,]),as.vector(sample[,2,]),
           h=c(sd(sample[,1,]),sd(sample[,2,])),
           n=1024,
           lims=c(-1.5, 2.5,51, 55)) # Enough to cover map
z <- melt(D$z)
z$Var1<-D$x[z$Var1]
z$Var2<-D$y[z$Var2]
map <- get_map(c(mean(sample[,1,]),mean(sample[,2,])),zoom=13,maptype="road")
mapPoints <- ggmap(map)+
  geom_point(aes(x = lon, y = lat), size = 1, data = landmarks, alpha = .5) +
  geom_raster(data=z,aes(x=Var1,y=Var2,fill=value))+
  guides(fill=FALSE,alpha=FALSE)+
  scale_fill_gradientn(colours=c("#0000FF00","#0000FFFF"))+
  coord_cartesian() +
  geom_line(aes(x=lon,y=lat),data=line1) +
  geom_line(aes(x=lon,y=lat),data=line2) +
  geom_line(aes(x=lon,y=lat),data=line3)+
  geom_point(aes(x=lon,y=lat),
             data=data.frame(lon=mean(sample[,1,]),lat=mean(sample[,2,])),
             size=0.5,colour="#FF0000")
mapPoints
```

Part 2

```
register_google(key="AIzaSyAnGAw53RJtWLiJ76HKtnxarZK2VR4GL5s",write=TRUE)
map <- get_map(c(-0.70956,52.04623),zoom=11,maptype="road")
ggmap(map)

landmarks<-data.frame(lon=c(-0.997343,-1.9108337, 1.2672059),lat=c(51.4502136,
52.4773136,52.6154779))
alpha <- bearing(c(-0.70956,52.04623), c(-0.997343, 51.4502136 ))
beta <- bearing(c(-0.70956,52.04623), c(-1.9108337, 52.4773136 ))
gamma <- bearing(c(-0.70956,52.04623), c(1.2672059, 52.6154779 ))
```

```
d <- seq(0,3,0.0001) # Length of the line
line1 <- data.frame(lon=landmarks[1,1] + d*sin(alpha*pi/180+pi),
                    lat=landmarks[1,2] + d*cos(alpha*pi/180+pi))
line2 <- data.frame(lon=landmarks[2,1] + d*sin(beta*pi/180+pi),
                    lat=landmarks[2,2] + d*cos(beta*pi/180+pi))
line3 <- data.frame(lon=landmarks[3,1] + d*sin(gamma*pi/180+pi),
                    lat=landmarks[3,2] + d*cos(gamma*pi/180+pi))

map <- get_map(c(-0.70956,52.04623),zoom=7,maptype="watercolor")
mapPlot <- ggmap(map)+
  geom_point(aes(x = lon, y = lat), size = 1, data = landmarks, alpha = .5) +
  geom_line(aes(x=lon,y=lat),data=line1) +
  geom_line(aes(x=lon,y=lat),data=line2) +
  geom_line(aes(x=lon,y=lat),data=line3)
mapPlot

intersectBearings <- function(p1,b1,p2,b2) {
  x1 <- p1[1]
  x2 <- p1[1] + 0.1*sin(b1*pi/180)
  x3 <- p2[1]
  x4 <- p2[1] + 0.1*sin(b2*pi/180)
  y1 <- p1[2]
  y2 <- p1[2] + 0.1*cos(b1*pi/180)
  y3 <- p2[2]
  y4 <- p2[2] + 0.1*cos(b2*pi/180)
  x <- ((x1*y2-y1*x2)*(x3-x4)-(x1-x2)*(x3*y4-y3*x4))/((x1-x2)*(y3-y4)-(y1-y2)*(x3-x4))
  y <- ((x1*y2-y1*x2)*(y3-y4)-(y1-y2)*(x3*y4-y3*x4))/((x1-x2)*(y3-y4)-(y1-y2)*(x3-x4))
  return(as.numeric(c(x,y)))
}

intersectBearings(landmarks[1,],alpha,landmarks[2,],beta)

intersection <- intersectBearings(landmarks[1,],alpha,landmarks[2,],beta)

intersection <- intersectBearings(landmarks[1,],alpha,landmarks[2,],beta)
roads.box <- center_bbox(center_lon=intersection[1],
                         center_lat=intersection[2],
                         width=150,
                         height=150)
api <- osmsource_api(url="https://api.openstreetmap.org/api/0.6/")
q = opq(st_bbox(roads.box)) %>%
  add_osm_feature(key = "highway")
res = osmdata_sf(q = q)
hw_lines <- st_coordinates(res$osm_lines)[,1:2]

distanceToRoad <- function(par) {
  distance <- dist2Line(c(par[1],par[2]), hw_lines)
  proper.distance <- distance[1] * 3.28084
  return(proper.distance)
}
```

```
bearings <- function(lambda1, phi1, lambda2, phi2) {
  (360 + atan2(phi2 - phi1, lambda2 - lambda1) * 180/pi) %% 360
}

likelihood <- function(theta, alpha, beta, gamma) {
  if (theta[3] <= 0){
    return (log(0))
  }
  else {
    return (dnorm(alpha, bearings(theta[1], theta[2],landmarks[1,1], landmarks[1,2]),
sqrt(theta[3]), log=T) +
              dnorm(beta, bearings(theta[1], theta[2],landmarks[2,1], landmarks[2,2]),
sqrt(theta[3]), log=T) +
              dnorm(gamma, bearings(theta[1], theta[2],landmarks[3,1], landmarks[3,2]),
sqrt(theta[3]), log=T))
  }
}


prior <- function(theta) {
  sum(dnorm(distanceToRoad(theta), 0, 6, log=T) +
        dexp(theta[3], 20, log=T))
}

draws <- array(0,dim=c(4000,3,3))
draws[4000,1,] <- runif(3,intersection[1] - 0.01, intersection[1] + 0.01)
draws[4000,2,] <- runif(3,intersection[2] - 0.01, intersection[2] + 0.01)
draws[4000,3,] <- rexp(3,20)

prop.cov <- c(1e-8,1e-8,1e-4)*diag(3)

converged <- FALSE
while (!converged) {
  draws[1,,] <- draws[4000,,]
  accepted <- 1
  for (step in 2:4000) {
    for (chain in 1:3) {
      proposed <- rmvnorm(1,draws[step-1,,chain],prop.cov)
      r <- likelihood(proposed, alpha, beta, gamma) + prior(proposed) -
        (likelihood(draws[step-1,,chain], alpha, beta, gamma ))-prior(draws[step-1,,chain])
      a <- min(0,r)
      u <- runif(1)
      if (log(u) < a) {
        draws[step,,chain] <- proposed
        accepted <- accepted + 1
      } else {
        draws[step,,chain] <- draws[step-1,,chain]
      }
    }
  }
  print(sprintf("Acceptance rate: %f",accepted/120))
  chainlist <- mcmc.list(Chain1=mcmc(draws[,,1]),
```

```
                    Chain2=mcmc(draws[,,2]),
                    Chain3=mcmc(draws[,,3]))
  converged <- all((gelman.diag(chainlist)$psrf[,2])<1.2)
  if (!converged) {
    if (accepted/120 <= 20) {
      prop.cov <- prop.cov * 0.8
    }
    if (accepted/120 >= 60) {
      prop.cov <- prop.cov * 1.2
    }
  }
  plot(chainlist) # This plots current state of the chains
  Sys.sleep(0.1)
}

s1 <- rbind(draws[,,1], draws[,,2], draws[,,3])
vc <- cov(s1)
vc

converged <- FALSE
while (!converged) {
  draws[1,,] <- draws[4000,,]
  accepted <- 1
  for (step in 2:4000) {
    for (chain in 1:3) {
      proposed <- rmvnorm(1,draws[step-1,,chain],vc)
      r <- likelihood(proposed, alpha, beta, gamma) + prior(proposed) -
        (likelihood(draws[step-1,,chain], alpha, beta, gamma ))-prior(draws[step-1,,chain])
      a <- min(0,r)
      u <- runif(1)
      if (log(u) < a) {
        draws[step,,chain] <- proposed
        accepted <- accepted + 1
      } else {
        draws[step,,chain] <- draws[step-1,,chain]
      }
    }
  }
  print(sprintf("Acceptance rate: %f",accepted/120))
  chainlist <- mcmc.list(Chain1=mcmc(draws[,,1]),
                    Chain2=mcmc(draws[,,2]),
                    Chain3=mcmc(draws[,,3]))
  converged <- all((gelman.diag(chainlist)$psrf[,2])<1.2)
  if (!converged) {
    if (accepted/120 <= 20) {
      vc <- vc * 0.8
    }
    if (accepted/120 >= 60) {
      vc <- vc * 1.2
    }
  }
  plot(chainlist) # This plots current state of the chains
```

```
  Sys.sleep(0.1)
}

#since convergence is achieved we begin collecting the final sample from the posterior

sample <- array(0,dim=c(10000,3,3))
sample[1,,] <- draws[4000,,]


for (step in 2:10000) {
  for (chain in 1:3) {
    proposed <- rmvnorm(1,sample[step-1,,chain],vc)
    r <- likelihood(proposed, alpha, beta, gamma) + prior(proposed) -
      (likelihood(sample[step-1,,chain], alpha, beta, gamma ))-prior(sample[step-1,,chain])
    a <- min(0,r)
    u <- runif(1)
    if (log(u) < a) {
      sample[step,,chain] <- proposed
    } else {
      sample[step,,chain] <- sample[step-1,,chain]
    }
  }
}

#distance to a road

mean(sample[,1,])
mean(sample[,2,])

#plots
D <- kde2d(as.vector(sample[,1,]),as.vector(sample[,2,]),
           h=c(sd(sample[,1,]),sd(sample[,2,])),
           n=1024,
           lims=c(-1.5, 2.5,51, 55)) # Enough to cover map
z <- melt(D$z)
z$Var1<-D$x[z$Var1]
z$Var2<-D$y[z$Var2]
map <- get_map(c(mean(sample[,1,]),mean(sample[,2,])),zoom=15,maptype="road")
mapPoints <- ggmap(map)+
  geom_point(aes(x = lon, y = lat), size = 1, data = landmarks, alpha = .5) +
  geom_raster(data=z,aes(x=Var1,y=Var2,fill=value))+
  guides(fill=FALSE,alpha=FALSE)+
  scale_fill_gradientn(colours=c("#0000FF00","#0000FFFF"))+
  coord_cartesian() +
  geom_line(aes(x=lon,y=lat),data=line1) +
  geom_line(aes(x=lon,y=lat),data=line2) +
  geom_line(aes(x=lon,y=lat),data=line3)+
  geom_point(aes(x=lon,y=lat),
             data=data.frame(lon=mean(sample[,1,]),lat=mean(sample[,2,])),
             size=0.5,colour="#FF0000")
mapPoints
```