MATH 611 HW8

Jeff Gould

10/19/2020

1 - Let $X(t)$ be a Markov chain on the state space $\{F, C\}$ (F - fair, C - cheating). Suppose that $X(t)$ changes state with probability $\alpha = .05$ regardless of its current state. Let $Y(t)$ be a r.v. with values from $\{1, 2, 3, 4, 5, 6\}$. ($Y(t)$ corresponds to the t-th role of a die). If $X(t) = F$ then $Y(t)$ is uniformly distributed on $\{1, 2, 3, 4, 5, 6\}$ (a fair die). If $X(t) = C$ then $P(Y(t) = 6) = 1/50$ and all values for $Y(t)$ are equally likely. Assume $X(0) = F$ where π is the stationary distribution of $X(t)$.

i) Write a function, SampleCasino(T) that samples $X(t)$, $Y(t)$ for $t \leq T$

```
SampleCasino <- function(bigT){

  X_t <- c("Fair", rep(NA, bigT))

  for (j in 2:(bigT+1)) {
    if(X_t[j-1] == "Fair"){
      X_t[j] = sample(c("Fair", "Cheat"), 1, prob = c(0.95, 0.05))
    }else{
      X_t[j] = sample(c("Fair", "Cheat"), 1, prob = c(0.05, 0.95))
    }

  }

  Y_t <- sapply(X_t, function(x){
    if(x == "Fair"){
      return(sample(1:6, 1))
    }else{
      return(sample(1:6, 1, prob = c(49/250, 49/250, 49/250, 49/250, 49/250, 1/50)))
    }
  })

  return(data.frame(t = c(0:bigT),
                    X = X_t,
                    Y = Y_t))

}
```

ii)

```
set.seed(1234)
testSample <- SampleCasino(200)
```

$$P(X(0) = i_0, X(1) = i_1, \dots X(T) = i_T | Y(0) = j_0, Y(1) = j_1, \dots Y(T) = j_T) =$$
$$\frac{P(X(0) = i_0, X(1) = i_1 \dots X(T) = i_T, Y(0) = j_0, Y(1) = j_1, \dots Y(T) = j_T)}{P(Y(0) = j_0, Y(1) = j_1 \dots Y(T) = j_T)} =$$

$$\frac{1}{\alpha} M_{1i_1} M_{i_1 i_2} \dots M_{i_{T-1} i_T} g_1(j_0) g_{i_1}(j_1) \dots g_{i_T}(j_T)$$

$$\phi = \begin{pmatrix} 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 49/250 & 49/250 & 49/250 & 49/250 & 49/250 & 1/50 \end{pmatrix}$$

$$M = \begin{pmatrix} 0.95 & 0.05 \\ 0.05 & 0.95 \end{pmatrix}$$

$$\alpha = P(Y(0) = j_0, Y(1) = j_1, Y(2) = j_2, \dots, Y(T) = j_t) = \frac{1}{6} \sum_{j_1 \in \{F,C\}} \sum_{j_2 \in \{F,C\}} \cdots \sum_{j_T \in \{F,C\}} g_{j_1}(Y(1)) g_{j_2}(Y(2)) \dots g_{j_T}(Y(T))$$

c

Let Ω be the state space of Z. Then Ω consists of all permutations of length T of $\{Fair, Cheat\}$, with the excpetion that $\omega_0 = Fair$

d

$P(\omega) = \frac{1}{\alpha} M_{1i} M_{i_1 i_2} \dots M_{i_{T-1} i_T} g_1(j_o) g_{i_1}(j_1) \dots g_{i_T}(j_T)$

Suppose we select a sequence of 5 rolls to flip and create ω', ie $\omega_{i_k}\omega_{i_{k+1}}\omega_{i_{k+2}}\omega_{i_{k+3}}\omega_{i_{k+4}}$ are switched from their current state to the opposite

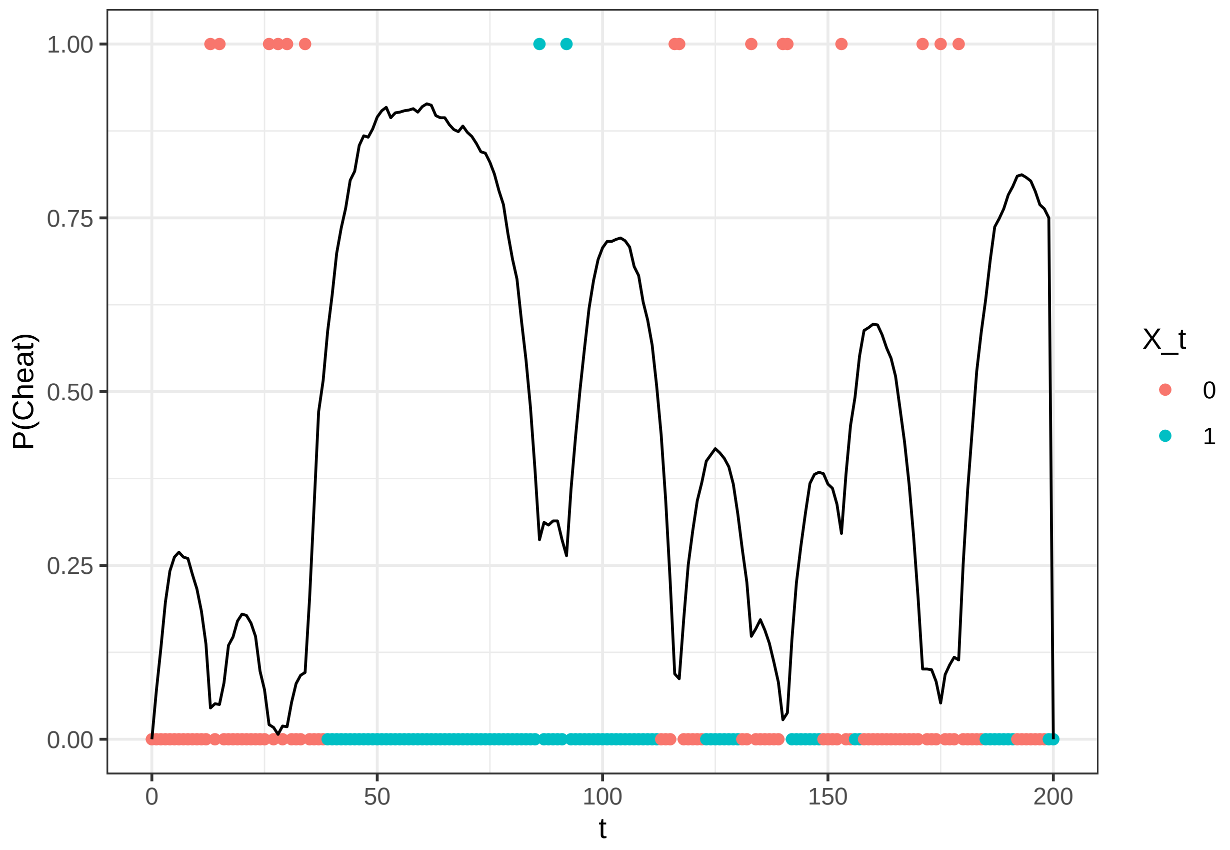
$$\frac{P(\omega')}{P(\omega)} = \frac{1/\alpha M_{1\omega'_i} M_{\omega'_{i_1}\omega'_{i_2}} \dots M_{\omega i'_{T-1} \omega i_T'} g_1(j_0) g_{\omega'_{i_1}}(j_1) \dots g_{\omega'_{i_T}}(j_T)}{1/\alpha M_{1\omega_i} M_{\omega_{i_1}\omega_{i_2}} \dots M_{\omega i_{T-1} \omega i_T} g_1(j_0) g_{\omega_{i_1}}(j_1) \dots g_{\omega_{i_T}}(j_T)} =$$

$$\frac{M_{\omega'_{k-1}\omega'_k} M_{\omega'_k\omega'_{k+1}} M_{\omega'_{k+1}\omega'_{k+2}} M_{\omega'_{k+2}\omega'_{k+3}} M_{\omega'_{k+3}\omega'_{k+4}} M_{\omega'_{k+4}\omega'_{k+5}} g_{\omega'_k}(j_k) g_{\omega'_{k+1}}(j_{k+1}) g_{\omega'_{k+2}}(j_{k+2}) g_{\omega'_{k+3}}(j_{k+3}) g_{\omega'_{k+4}}(j_{k+4})}{M_{\omega_{k-1}\omega_k} M_{\omega_k\omega_{k+1}} M_{\omega_{k+1}\omega_{k+2}} M_{\omega_{k+2}\omega_{k+3}} M_{\omega_{k+3}\omega_{k+4}} M_{\omega_{k+4}\omega_{k+5}} g_{\omega_k}(j_k) g_{\omega_{k+1}}(j_{k+1}) g_{\omega_{k+2}}(j_{k+2}) g_{\omega_{k+3}}(j_{k+3}) g_{\omega_{k+4}}(j_{k+4})}$$

Also, $\frac{R(\omega'\omega)}{R(\omega\omega')} = 1$, as $R(\omega'\omega) = R(\omega\omega') = 0.05^5$ (in this example, below we ended up using 10)

```r
### Initialize omega as all fair die
omega <- rep("Fair", nrow(testSample))

### probabiliry matrix for each state space
phi <- matrix(c(rep(1/6, 6), rep(49/250, 5), 1/50), byrow = T, nrow = 2)

rownames(phi) = c("Fair", "Cheat")

### Transition probabilty matrix
M <- matrix(c(0.95, 0.05, 0.05, 0.95), nrow = 2)
rownames(M) = c("Fair", "Cheat")
colnames(M) = c("Fair", "Cheat")

K <- 9
```

```r
s <- 1
iterations <- 5000000
Omega <- matrix(NA, nrow = iterations / 5000, ncol = nrow(testSample))

while(s <= iterations){

  ### sample starting point to flip sequence
  k <- sample(2:(length(omega)), 1)
  omega_prime <- omega
  for(kk in k:(k+K)){
    if(kk > 200){break}

    if(omega[kk] == "Fair"){
      omega_prime[kk] = "Cheat"
    }else{
      omega_prime[kk] = "Fair"
    }

  }

  end_index <- min(k+K, 200)
  g_prime <- sapply(k:end_index, function(x)phi[omega_prime[x], testSample$Y[x]])
  g <- sapply(k:end_index, function(x)phi[omega[x], testSample$Y[x]])

  end_index2 <- min(k+K+1, 200)
  M_omega_prime <- sapply(k:(end_index2), function(x){M[omega_prime[x-1], omega_prime[x]]})
  M_omega <- sapply(k:(end_index2), function(x){M[omega[x-1], omega[x]]})


  MH_ratio <- prod(M_omega_prime) * prod(g_prime) / (prod(M_omega) * prod(g))
  u <- runif(1)

  if(u <= MH_ratio){
    omega <- omega_prime
  }

  if(s %% 5000 == 0){Omega[s/5000, ] = omega}

  s <- s + 1
}


results <- data.frame(p_cheating = colMeans(Omega == "Cheat"),
                      Y_t = testSample$Y,
                      X_t = as.numeric(testSample$X == "Cheat"),
                      t = seq(0, 200))

results %>%
  mutate(is_6 = ifelse(Y_t == 6, 1, 0),
         X_t = as.factor(X_t)) %>%
  ggplot(aes(x = t)) +
  geom_point(aes(y = is_6, color = X_t)) +
  theme_bw() +
  geom_line(aes(y =  p_cheating)) +
  labs( y = "P(Cheat)")
```



Here the line represents the probabilty that the casino is cheating, the points indicate whether or not a 6 was rolled (1-5 is plotted at 0, 6 is plotted at 1), and the color of the point indicates whether or not the casino was cheating.

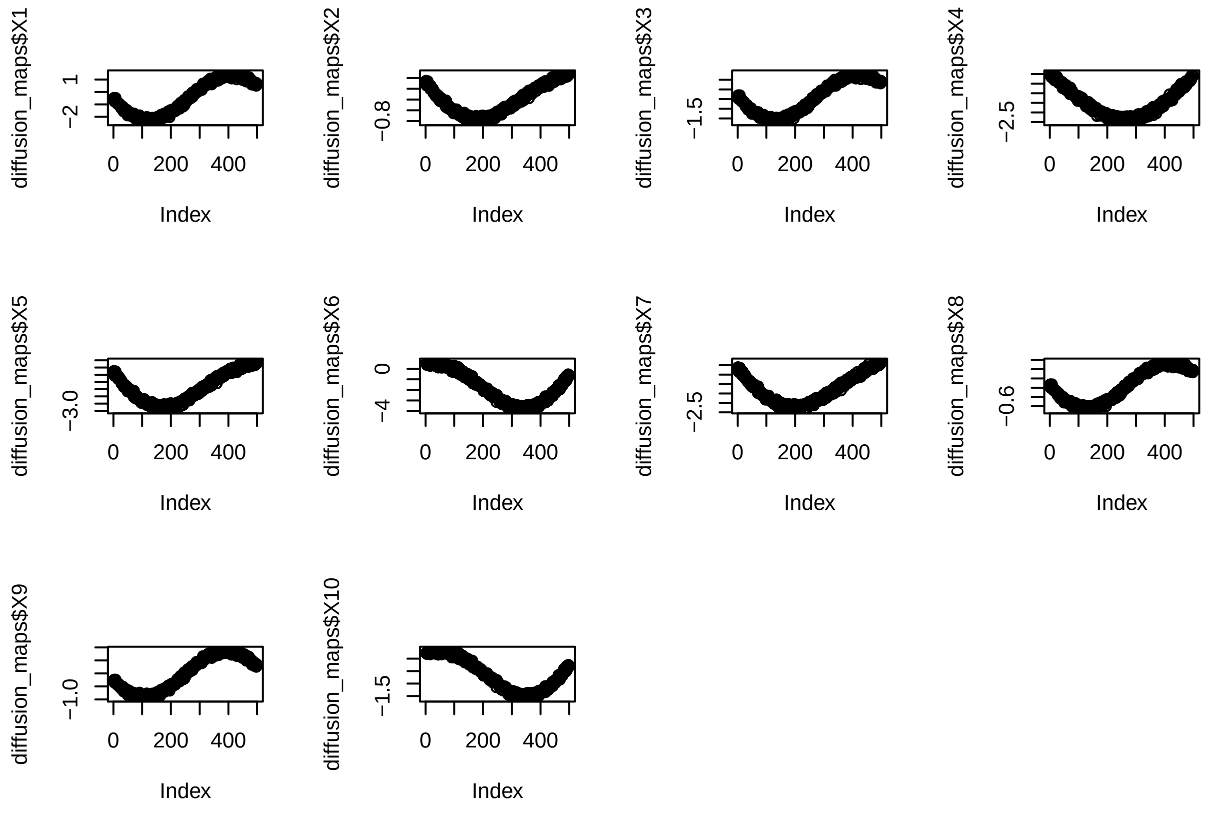
We see that the MH algorithm mostly does a good job of predicting when the casino is cheating. There is a little run from $t \in [86, 92]$ where the casino is cheating, but rolled two 6's anyways and our sampler dropped from a nearly 90% chance that the casino is cheating down to near 25%. But it quickly climbed up close to 75%.

2

a

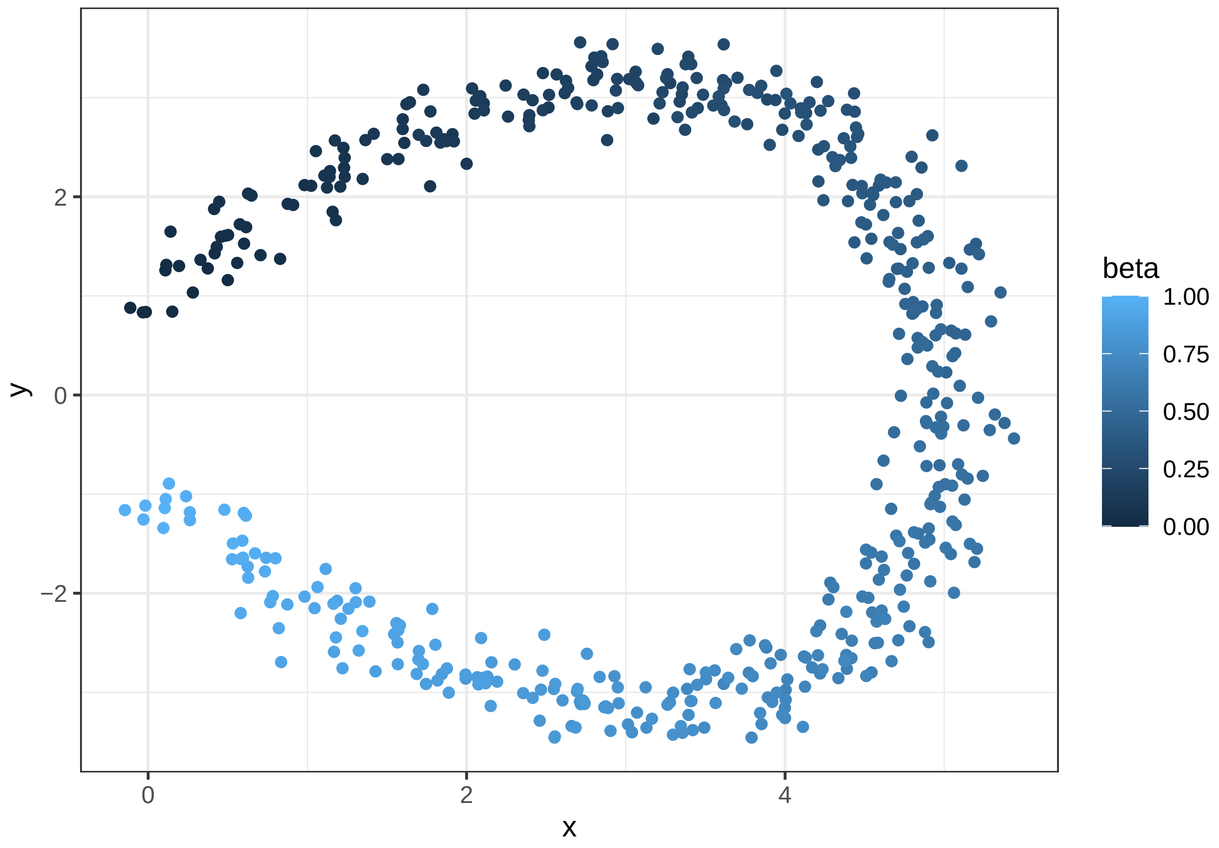
```
diffusion_maps <- read_csv("diffusion_maps_data.csv")

par(mfrow = c(3,4))
plot(diffusion_maps$X1)
plot(diffusion_maps$X2)
plot(diffusion_maps$X3)
plot(diffusion_maps$X4)
plot(diffusion_maps$X5)
plot(diffusion_maps$X6)
```

```r
plot(diffusion_maps$X7)
plot(diffusion_maps$X8)
plot(diffusion_maps$X9)
plot(diffusion_maps$X10)
```



b

```r
source("make_1d_manifold.R")
```

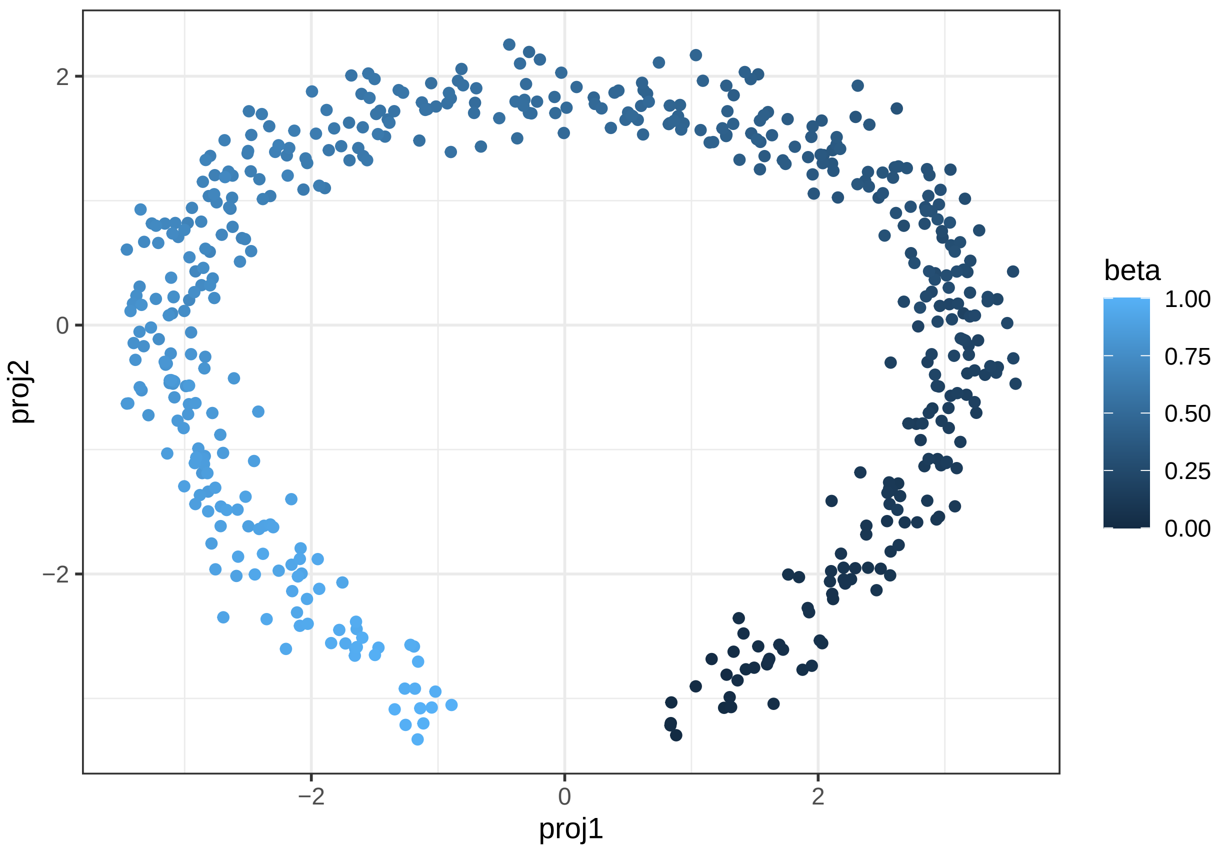


We see that the underlying data forms a horshoe shaped 1-d manifold. The X dimension is the sin curve from $[0, \pi]$, multiplied by 5, and injected with some noise.

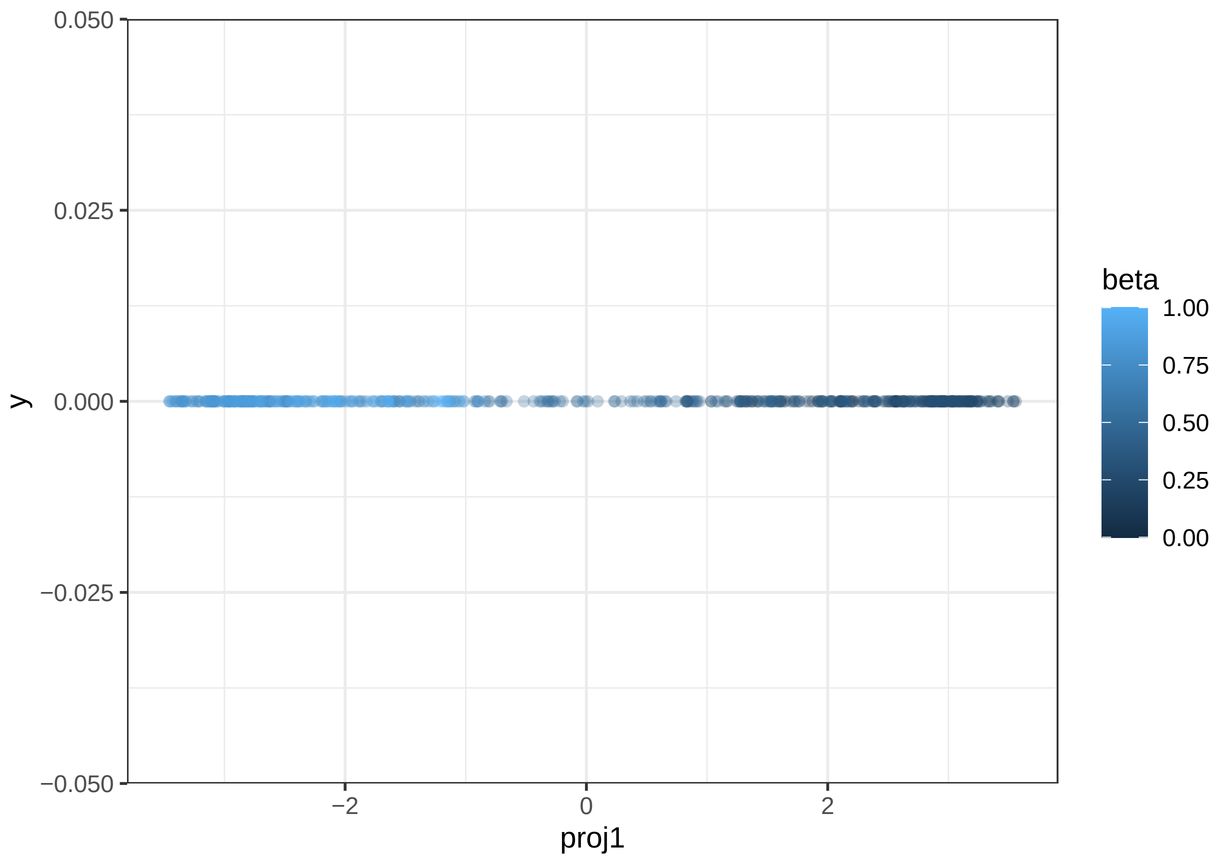
The Y dimension is mostly drive by cos curve from $[0, \pi]$, but it is scaled by $20\beta(1 - \beta)$. $\beta[1 - \beta]$ goes from 0, up to 0.25 when $\beta = 0.5$, and then drops down to 0 at $\beta = 1$. It then has an additive factor of $2(0.5 - \beta)$, whice creates the separation at the open end.

c

```r
make2dPCA <- function(data, plotTitle = NULL){

  ## Center data
  ## Center data
  mu <- colMeans(data)
  centeredData <- t(apply(data, 1, function(x){x-mu}))

  ## Calculate covariance matrix
  CovTS <- t(centeredData) %*% centeredData

  ## Compute eigendata, take eigenvectors
  ev <- eigen(CovTS)
  eigVec = ev$vectors
  eigVals = ev$values

  label1 <- glue::glue("proj1 ({round(100 * eigVals[1] / sum(eigVals),1)}% of variance captured)")
  label2 <- glue::glue("proj2 ({round(100 * eigVals[2] / sum(eigVals),1)}% of variance captured)")
  proj1 <- centeredData %*% eigVec[,1]
  proj2 <- centeredData %*% eigVec[,2]

  projection2D <- data.frame(proj1 = proj1,
                             proj2 = proj2)


  return(projection2D)



}

PCA_manifold <- make2dPCA(diffusion_maps[,1:10])

PCA_manifold %>%
  mutate(beta = diffusion_maps$beta) %>%
  ggplot(aes(x = proj1, y = proj2, color = beta)) +
  geom_point() +
  theme_bw()
```



```
PCA_manifold %>%
  mutate(beta = diffusion_maps$beta) %>%
  ggplot(aes(x = proj1,  color = beta)) +
  geom_point(aes(y=0), alpha = 0.3) +
  theme_bw()
```



d

i

The distance formula we use is a distance based off of probability from transitioning to one state given that we started in each state i or j after t steps. We then sum the difference in probabilty for arriving at each node to get the "distance". We divide by $\pi(x^{(k)})$, which is the probability of $x^{(k)}$ in the stationary distribution

ii

To calculate λ, first solve for A from K, where $A_{ij} = \frac{K(x^{(i)},x^{(j)})}{d^{(i)}}$

Then λ are the eignevalues of A

To solve for r:

First let D be the diagonal matrix with $D_{ii} = d_i$

$r^{(i)} = D^{-1/2}q^{(i)}$

where $q^{(i)}$ is the i^{th} eigenvector of A

The formula for $D_t^2(x^{(i)}, x^{(j)}) = ||y^{(i)} - y^{(j)}||$

iii

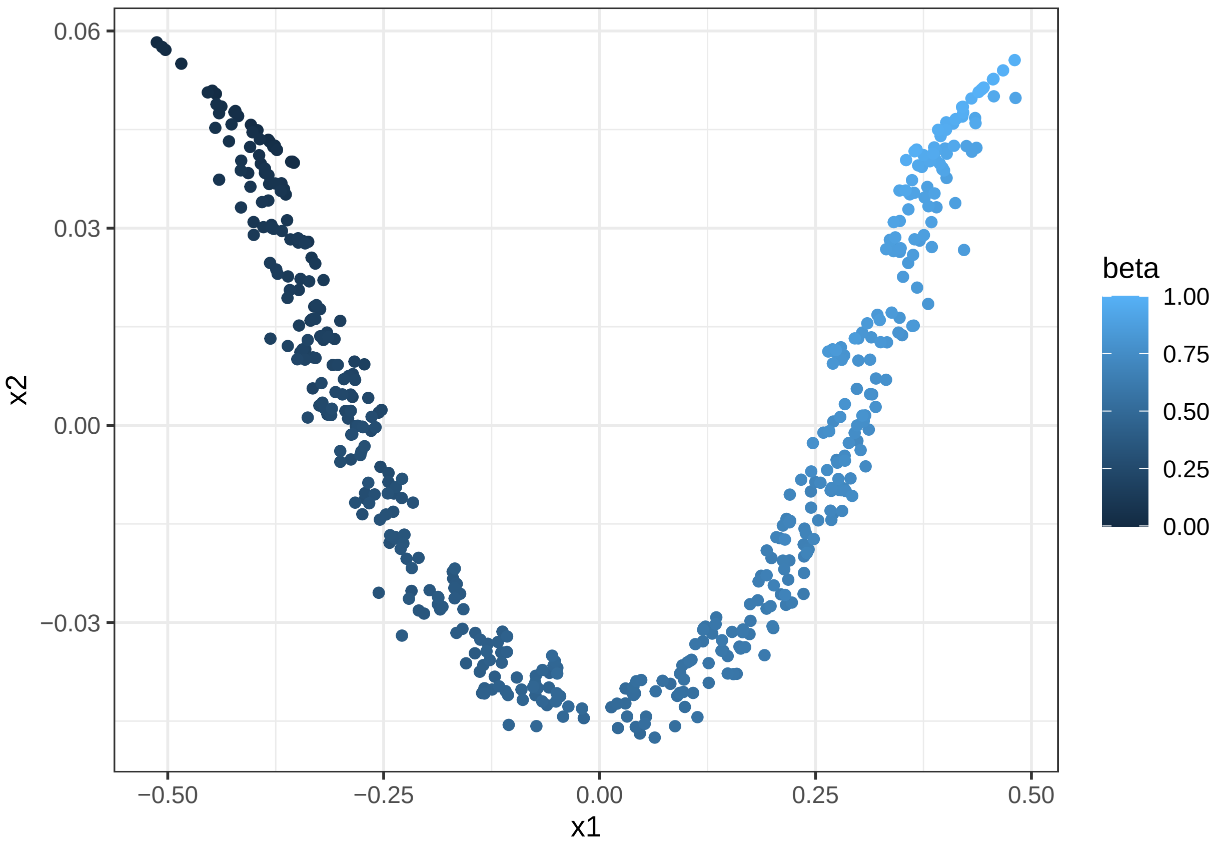
Diffusion maps are useful when trying to reduce the dimensionality of data when the underlying data structure is non-linear. Ie, for a helix we can reduce the dimension to the underlying circle. With PCA, it is typically not able to portray the non-linear aspects of the data as well, and parts that may be "far" apart in the data structure could be plotted close together. Ie in the graph above, the ends of the horshoe are relatively close to each other, but in the underlying data structure they are quite far apart

iv

To create the space of edges, I calculate the "distance" between each point, and take the top-15 closest points as our edge space. This means that an individual node could have more than 15 edges, if it is one of the 15 nearest for one point but that point is not among the 15 nearest. This is necessary for a symmetric K

```r
norm <- function(x){
  sqrt(sum(x^2))
}

dist <- function(i,j){
  return(norm(x = c(m_rot[i, ] - m_rot[j, ])))
  }

delta <- 0.7
nNearest <- 15

points <- expand.grid(1:nrow(m_rot), 1:nrow(m_rot))

edgeLengths <- mapply(dist, i = points$Var1, j = points$Var2)

Edges <- data.frame(x1 = points$Var2,
                    x2 = points$Var1,
                    distance = edgeLengths) %>%
  filter(distance != 0) %>%
  group_by(x1) %>%
  top_n(n = nNearest, wt = -distance)


K <- matrix(0, nrow = nrow(m_rot), ncol = nrow(m_rot))

for(i in 1:nrow(Edges)){

  K[Edges$x1[i], Edges$x2[i]] = exp(-Edges$distance[i]^2 / delta)
  K[Edges$x2[i], Edges$x1[i]] = exp(-Edges$distance[i]^2 / delta)

}


D <- diag(rowSums(K))
d <- rowSums(K)
D_minus <- diag(1 / (sqrt(d)))

A <- t(apply(K, 1, function(x){
  x = x / sum(x)
}))
```

```r
eigData <- eigen(A)
Q <- Re(eigData$vector)

Q <- apply(Q, 2, function(x){
  return(x / sum(x))
})



Lambda <- Re(eigData$value)

R <- (apply(Q, 2, function(q){
   r <- D_minus %*% q
   return(r)
}))

t <- 50

projection <- sapply(1:nrow(m_rot), function(x){
  Lambda[x]^t * R[,x]
})


data.frame(x1 = projection[,2],
           x2 = projection[,3],
           beta = beta) %>%
  ggplot(aes(x = x1, y = x2)) +
  geom_point(aes(color = beta)) +
  theme_bw()
```


0.06
0.03
0.00
−0.03
x2
−0.50
−0.25
0.00
0.25
0.50
x1
beta
1.00
0.75
0.50
0.25
0.00