Direct Approach KEF

Pouya Roudaki

2024-10-21

Kernel Mean Embedding Estimation

Set the randomness seed.

```
# set the seed

seed <- 7
set.seed(seed)
```

Specfiy the True density

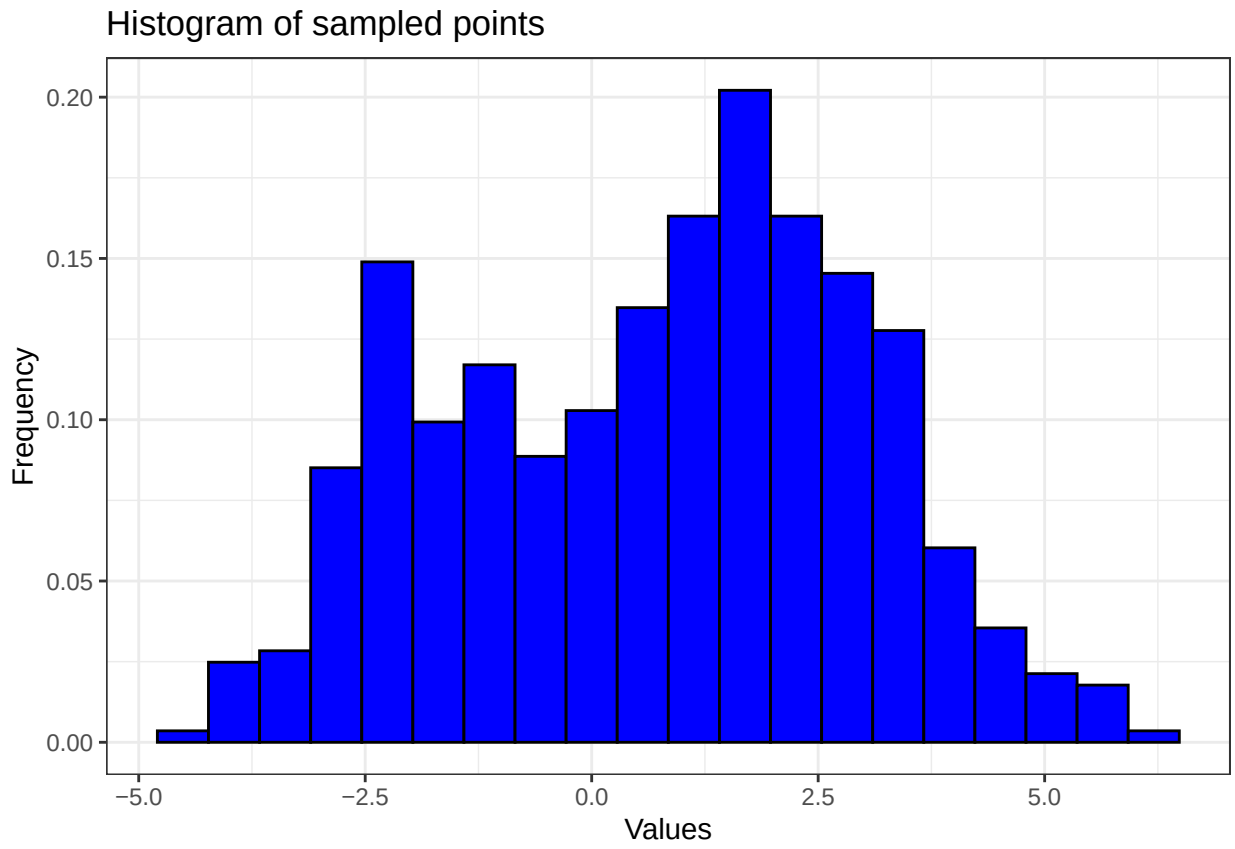
Set the mean and standard deviation of normal distribution P

```
# Set the mean and standard deviation of normal distribution P
 n = 500
 means = c(-2, 2)
 sds = c(1, 1.5)
 probabilities = c(0.3, 0.7)
```

Take Random samples

```
# vector of fixed points
vec_fixed_points <- sort(rnorm_mixture(n, means, sds, probabilities))

library(ggplot2)
# Assuming vec_fixed_points is your data
df <- data.frame(points = vec_fixed_points)

# Create the histogram with 20 breaks
ggplot(df, aes(x = points, y = ..density..)) +
  geom_histogram(bins = 20, fill = "blue", color = "black") +
  labs(title = "Histogram of sampled points", x = "Values", y = "Frequency")+
  theme_bw()
```

```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



Find Gram Matrix of the sampled points

```
# List of fixed points
list_fixed_points = as.list(vec_fixed_points)

centering_param <- 7
# Find the Gram matrix
gram <- gram_matrix(vec_list = list_fixed_points,
                    kernel_type = "uniform_centered_brownian",
                    kernel_params = list(length_scale = 1, degree = 2,
                                         free_add = 0, free_mult = 1,
                                         nu_matern = 1, centering_param = centering_param))
```

True Kernel Mean Embeddings

```
# Grid of 100 points from -10 to 10
u <- centering_param


lambda <- 1

# Kernel mean embedding be careful change the mean if change the mean of P


# Define the function f with point as a parameter
```

```r
f <- function(x, point, lambda, probabilities, means, sds, centering_param) {
  base_measure_gaussian_mixture <- 0
  for (i in 1:length(probabilities)) {
    base_measure_gaussian_mixture <- base_measure_gaussian_mixture +
      probabilities[i] / (sqrt(2 * pi) * sds[i]) * exp(- (x - means[i])^2 / (2 * sds[i]^2)) *
      exp(-lambda/4*(x^2/centering_param + centering_param/3))
  }

  return ((lambda / 2) * (-abs(x - point) + (x^2 + point^2) / (2 * centering_param) + centering_param /
}



# Define a wrapper function to perform integration
integrate_for_point <- function(point, lambda, probabilities, means, sds, centering_param) {
  # Calculate the integral part
  integral_result <- integrate(function(x) f(x, point, lambda, probabilities,
                                             means, sds, centering_param),
                               subdivisions = 10000, rel.tol = 1e-10,
                               abs.tol = 1e-10,lower = -centering_param, upper = centering_param)$value

  additional_terms <- 0
  for (i in 1:length(probabilities)) {
    term1 <- probabilities[i] * lambda *
      (centering_param^2 + 6 * centering_param * point - 3 * point^2) *
      exp(lambda/24*(12*means[i]+4*centering_param+3*sds[i]^2*lambda))/ (24 * centering_param)
    term2 <- probabilities[i] * lambda *
      (centering_param^2 - 6 * centering_param * point - 3 * point^2) *
      exp(lambda/24*(-12*means[i]+4*centering_param+3*sds[i]^2*lambda))/ (24 * centering_param)
    erf_component1 <- erfc((2*(means[i] + centering_param)+ sds[i]^2*lambda )/ (2*sqrt(2) * sds[i]))
    erf_component2 <- erfc((2*(-means[i] + centering_param)+ sds[i]^2*lambda )/ (2*sqrt(2) * sds[i]))

    additional_terms <- additional_terms - (term1 * erf_component1) - (term2 * erf_component2)
  }

  result <- integral_result + additional_terms

  return(result)
}

# Apply the integration function to each element of grid_point
KME_true <- sapply(vec_fixed_points, integrate_for_point,
                   lambda = lambda,
                   probabilities = probabilities,
                   means = means,
                   sds = sds,
                   centering_param = centering_param)




# Data frame with true kernel mean embeddings
```

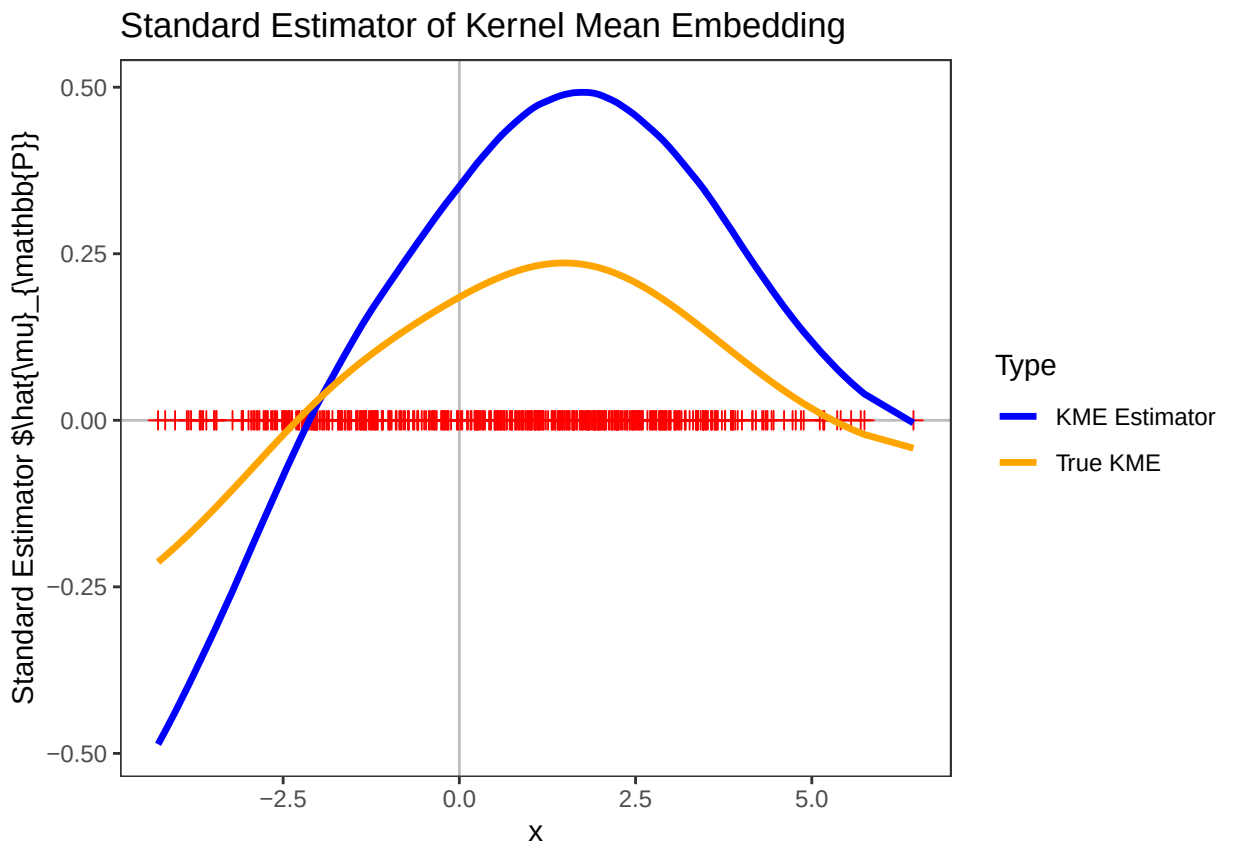
```r
true_KME <- data.frame(vec_fixed_points, true_KME = KME_true)
```

Kernel Mean Embedding Estimation: Standard Estimator

```r
# Data frame with estimated kernel mean embedding
df_std <- data.frame(vec_fixed_points, standard_KME = colMeans(gram))

# Data frame for fixed points: adding e
df_fixed_points <- data.frame(x = vec_fixed_points, y = rep(0, length(vec_fixed_points)))
```

Kernel Mean Embedding Estimator

```r
# Plot the results using ggplot
library(ggplot2)
# Create a combined data frame to handle both blue (standard) and orange (true KME) lines
df_combined <- rbind(
  data.frame(grid_points = df_std$vec_fixed_points, value = df_std$standard, line = "KME Estimator"),
  data.frame(grid_points = true_KME$vec_fixed_points, value = true_KME$true_KME, line = "True KME")
)

ggplot() +
  geom_hline(yintercept = 0, color = "gray", linetype = "solid", linewidth = 0.5) +  # Add axis y = 0
  geom_vline(xintercept = 0, color = "gray", linetype = "solid", linewidth = 0.5) +  # Add axis at x =
  geom_point(data = df_fixed_points, aes(x = x, y = y), color = 'red', size = 2 ,shape =3) +  # Plot x
  geom_line(data = df_combined, aes(x = grid_points, y = value, color = line), linewidth = 1.2) + # Plo
  labs(x = "x",
       y = "Standard Estimator $\\hat{\\mu}_{\\mathbb{P}}",
       title = "Standard Estimator of Kernel Mean Embedding") +
  theme_bw() +
  theme(panel.grid = element_blank(),  # Remove grid lines
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank()) +
  scale_color_manual(values = c("KME Estimator" = "blue", "True KME" = "orange")) +  # Custom colors fo
  guides(color = guide_legend(title = "Type"))  # Add a legend title
```



Pre calculations:

```
# Find the Gram matrix
gram <- gram_matrix(vec_list = list_fixed_points,
                    kernel_type = "uniform_centered_brownian",
                    kernel_params = list(length_scale = 1, degree = 2,
                                         free_add = 0, free_mult = 1,
                                         nu_matern = 1, centering_param = centering_param))

n = length(list_fixed_points)

rho = 1/(n^2) * sum(gram)

rho_with_stroke = 1/n * sum(diag(gram))

lambda_reg = n*(rho_with_stroke-rho) / ((n-1)*(n*rho-rho_with_stroke))
```

Find the shrinkage estimator of KME:

```
# Function evaluation on the grid
results <- sapply(vec_fixed_points, function(point) {
  reg_est_KME(evaluate_at = point,
              list_fixed = list_fixed_points,
              kernel_type = "uniform_centered_brownian",
              kernel_params = list(length_scale = 1, degree = 2, free_add = 0, free_mult = 1, nu_matern
              precomputed = list(lambda = lambda_reg))
})
```

```r
# Data frame with your data
df_reg <- data.frame(vec_fixed_points, regularized = results)
```

Shrinkage Estimator of Kernel Mean Embedding —-

Pre calculations:

```r
gram <- gram_matrix(vec_list = list_fixed_points,
                    kernel_type = "uniform_centered_brownian",
                    kernel_params = list(length_scale = 1, degree = 2,
                                         free_add = 0, free_mult = 1,
                                         nu_matern = 1,
                                       centering_param = centering_param))

n = length(list_fixed_points)

lambda_grid <- 10^seq(-14,10,1)

lambda_n_grid <- 10^seq(-14,10,1) * (n-1)

### Precompute regularized inverses for each lambda_n

regularized_inverse_grid <- lapply(lambda_n_grid, function(lambda_n){solve(gram + lambda_n * diag(n))})
```

LOOCV hyper parameter selection

```r
loocv_values <- rep(0,length = length(lambda_grid))
### Use sapply to iterate over lambda_grid
loocv_values <- sapply(1:length(lambda_grid), function(i) {
  loocv_shr(gram = gram, lambda = lambda_grid[i], precomp_reg_inv = regularized_inverse_grid[[i]])
})

### Create a dataframe to store lambda values and their corresponding LOOCV errors
loocv_df <- data.frame(lambda = lambda_grid, loocv = loocv_values)
```

```r
### Print the dataframe
print(loocv_df)
```

```
##    lambda    loocv
## 1   1e-14 1.310779
## 2   1e-13 1.292868
## 3   1e-12 1.292174
## 4   1e-11 1.292210
## 5   1e-10 1.292202
## 6   1e-09 1.292201
## 7   1e-08 1.292201
## 8   1e-07 1.292201
## 9   1e-06 1.292201
## 10  1e-05 1.292201
## 11  1e-04 1.292201
## 12  1e-03 1.292209
## 13  1e-02 1.292821
```

```
## 14  1e-01 1.308502
## 15  1e+00 1.429393
## 16  1e+01 1.543056
## 17  1e+02 1.563978
## 18  1e+03 1.566249
## 19  1e+04 1.566478
## 20  1e+05 1.566501
## 21  1e+06 1.566503
## 22  1e+07 1.566503
## 23  1e+08 1.566503
## 24  1e+09 1.566503
## 25  1e+10 1.566503
```

CV hyper parameter selection

```
lambda_grid <- c(10^seq(-14,10,1))

cv_values <- rep(0,length = length(lambda_grid))
### Use sapply to iterate over lambda_grid
cv_values <- sapply(1:length(lambda_grid), function(i) {
  cross_validation(gram = gram, lambda = lambda_grid[i], folds_number = nrow(gram),estimator_type = "sh
})

### Create a dataframe to store lambda values and their corresponding LOOCV errors
cv_df <- data.frame(lambda = lambda_grid, cv = cv_values)
```

```
### Print the dataframe
print(cv_df)
```

```
##    lambda       cv
## 1   1e-14 1.292201
## 2   1e-13 1.292201
## 3   1e-12 1.292201
## 4   1e-11 1.292201
## 5   1e-10 1.292201
## 6   1e-09 1.292201
## 7   1e-08 1.292201
## 8   1e-07 1.292201
## 9   1e-06 1.292201
## 10  1e-05 1.292201
## 11  1e-04 1.292201
## 12  1e-03 1.292209
## 13  1e-02 1.292821
## 14  1e-01 1.308502
## 15  1e+00 1.429393
## 16  1e+01 1.543056
## 17  1e+02 1.563978
## 18  1e+03 1.566249
## 19  1e+04 1.566478
## 20  1e+05 1.566501
## 21  1e+06 1.566503
## 22  1e+07 1.566503
## 23  1e+08 1.566503
## 24  1e+09 1.566503
```

```
## 25  1e+10 1.566503
```

```
#save.image("C:/Users/rouda/OneDrive/Research/Codes/R/kef/examples/rdata_direct_brownian_3.RData")
```

Find the best hyper parameter and corresponding beta weights

```
merged_num_anal <- merge(cv_df,loocv_df,by = "lambda")

# Choose the best lambda with lowest loocv
lambda_shr <- cv_df[which.min(cv_df$cv),"lambda"]

# Calculate inverse of regularizer
inverse_regularizer <- solve(gram + n * lambda_shr*diag(n))

# 1_n vector
one_n <- rep(1,n)/n

# Find beta
beta_s <- sqrt(n) * inverse_regularizer %*% gram %*%  one_n
```

Find the shrinkage estimator of KME

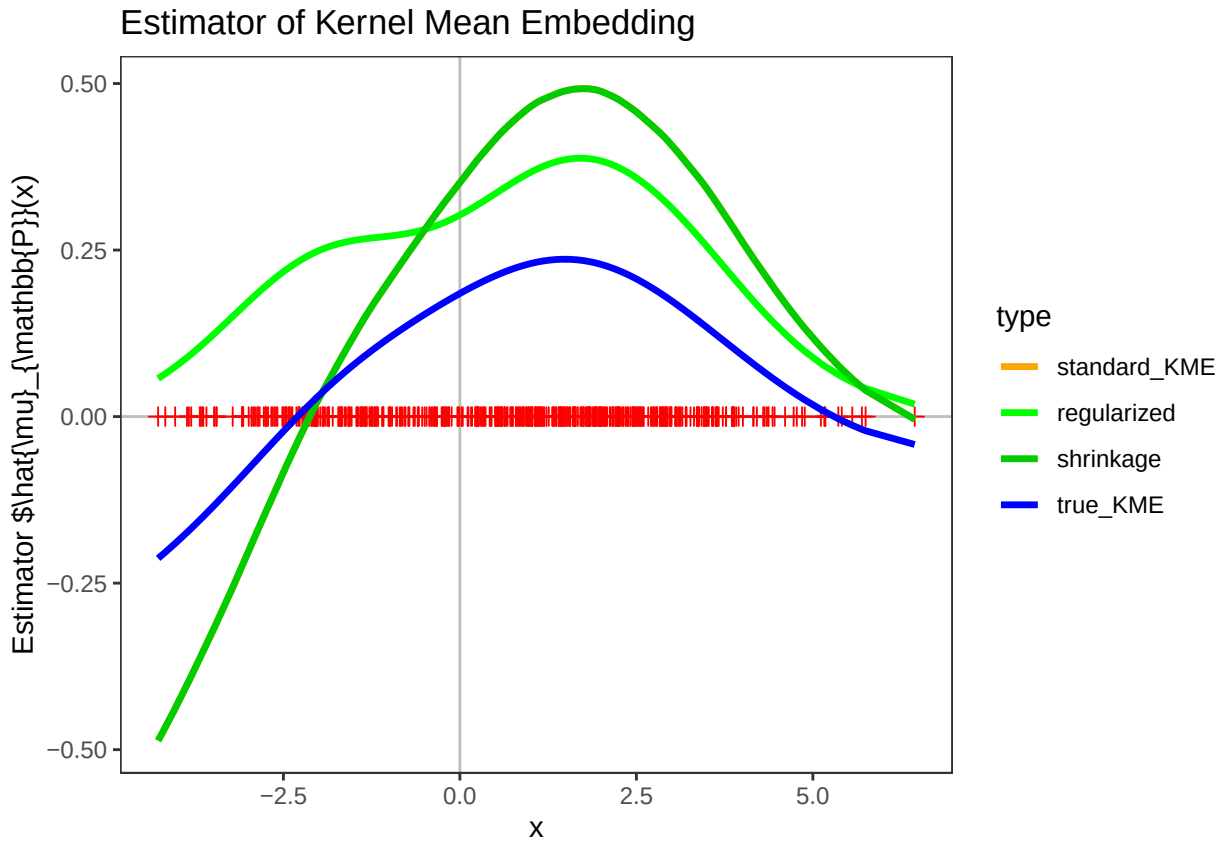
```
# Function evaluation on the grid
results <- sapply(vec_fixed_points, function(point) {
  shr_est_KME(evaluate_at = point,
              list_fixed = list_fixed_points,
              lambda_tunner = 1,
              kernel_type = "uniform_centered_brownian",
              kernel_params = list(length_scale = 1, degree = 2, free_add = 0, free_mult = 1, nu_matern
              precomputed = list(beta_s = beta_s))
})


# Data frame with your data
df_shr <- data.frame(vec_fixed_points, shrinkage = results)
```

Merge all the KME estimators and plot

```
df_all <- merge(df_std,df_reg,by = "vec_fixed_points")
df_all <- merge(df_all,df_shr,by = "vec_fixed_points")
df_all <- merge(df_all,true_KME,by = "vec_fixed_points")

#save.image("C:/Users/rouda/OneDrive/Research/Codes/R/kef/examples/rdata_direct_4.RData")
```

```
library(tidyr)

#gather data from columns 2 and 3
df_long <- gather(df_all, key="type", value="estimation", 2:5)

df_long$type <- factor(df_long$type, levels = colnames(df_all)[-1])

# Data frame for fixed points:
df_fixed_points <- data.frame(x = vec_fixed_points, y = rep(0, length(vec_fixed_points)))
```

```r
# Plot the results using ggplot
library(ggplot2)
ggplot(df_long, aes(x = vec_fixed_points, y = estimation, color = type)) +
  geom_hline(yintercept = 0, color = "gray", linetype = "solid", linewidth = 0.5) +  # Add axis y = 0
  geom_vline(xintercept = 0, color = "gray", linetype = "solid", linewidth = 0.5) +  # Add axis at x =
  geom_point(data = df_fixed_points, aes(x = x, y = y), color = 'red', size = 2 ,shape =3) +  # Plot x
  geom_line(linewidth = 1.2) + # Plot kernel mean embedding estimator
  labs(x = "x",
       y = "Estimator $\\hat{\\mu}_{\\mathbb{P}}(x)",
       title = "Estimator of Kernel Mean Embedding")+
  theme_bw()+
  theme(panel.grid = element_blank(),  # Remove grid lines
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank()) +  # Axis ticks color
  scale_color_manual(values = c('orange', 'green', 'green3', 'blue')) # Customize colors
```

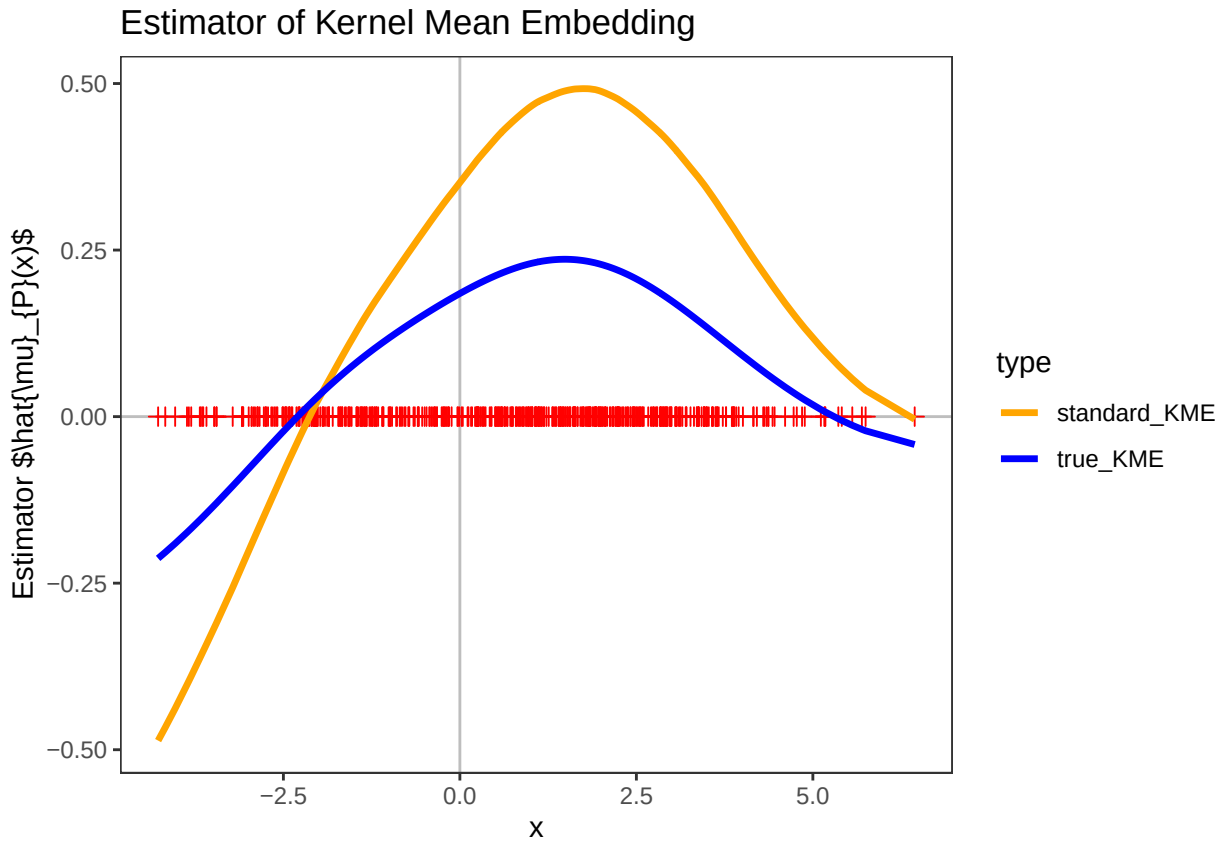



```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
```

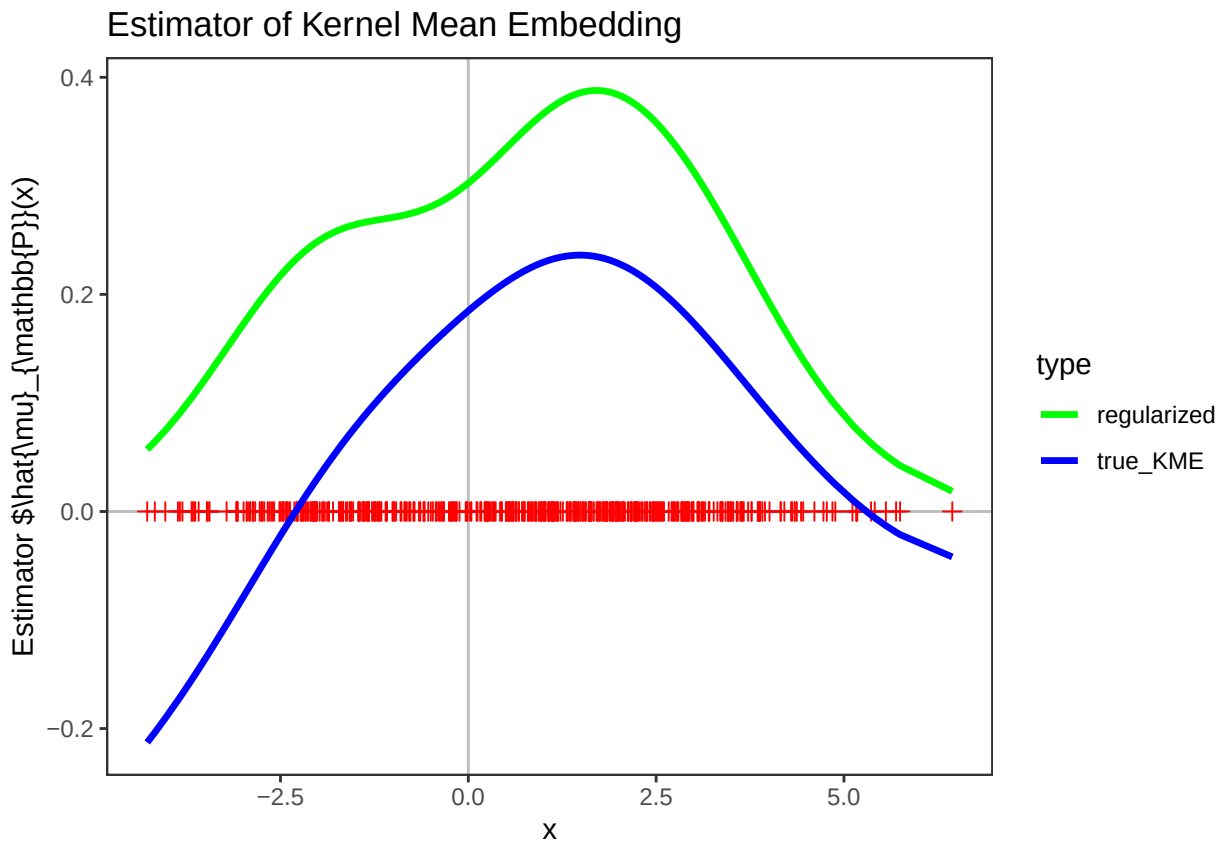
```
##     intersect, setdiff, setequal, union
```

```
# Plot the results using ggplot
ggplot(df_long %>% filter(type %in% c("standard_KME","true_KME")), aes(x = vec_fixed_points, y = estima
  geom_hline(yintercept = 0, color = "gray", linetype = "solid", linewidth = 0.5) +  # Add axis y = 0
  geom_vline(xintercept = 0, color = "gray", linetype = "solid", linewidth = 0.5) +  # Add axis at x =
  geom_point(data = df_fixed_points, aes(x = x, y = y), color = 'red', size = 2 ,shape =3) +  # Plot x
  geom_line(linewidth = 1.2) + # Plot kernel mean embedding estimator
  labs(x = "x",
       y = "Estimator $\\hat{\\mu}_{P}(x)$",
       title = "Estimator of Kernel Mean Embedding")+
  theme_bw()+
  theme(panel.grid = element_blank(),  # Remove grid lines
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank()) +  # Axis ticks color
  scale_color_manual(values = c('orange','blue')) # Customize colors
```

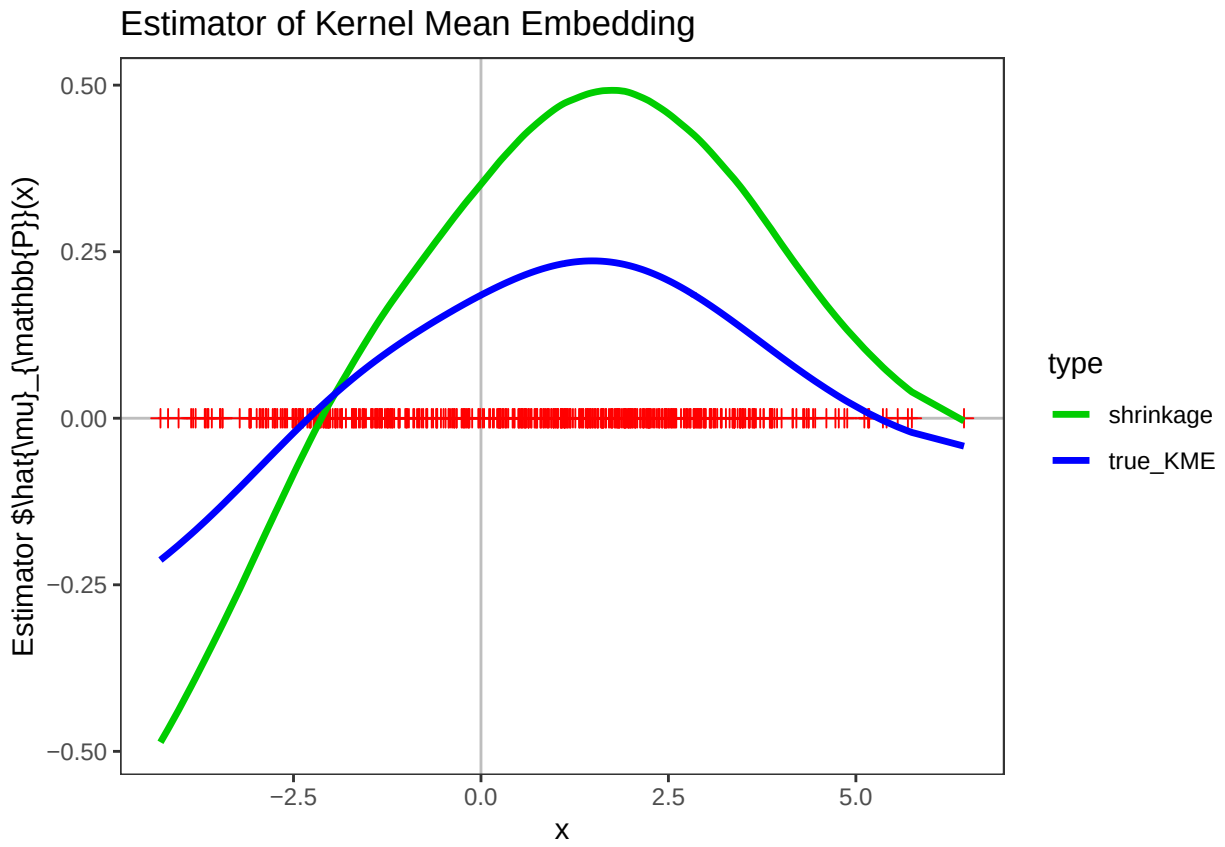



```
# Plot the results using ggplot
ggplot(df_long %>% filter(type %in% c("regularized","true_KME")), aes(x = vec_fixed_points, y = estimat
  geom_hline(yintercept = 0, color = "gray", linetype = "solid", linewidth = 0.5) +  # Add axis y = 0
  geom_vline(xintercept = 0, color = "gray", linetype = "solid", linewidth = 0.5) +  # Add axis at x =
  geom_point(data = df_fixed_points, aes(x = x, y = y), color = 'red', size = 2 ,shape =3) +  # Plot x
  geom_line(linewidth = 1.2) + # Plot kernel mean embedding estimator
  labs(x = "x",
       y = "Estimator $\\hat{\\mu}_{\\mathbb{P}}(x)",
       title = "Estimator of Kernel Mean Embedding")+
  theme_bw()+
```

```
  theme(panel.grid = element_blank(),  # Remove grid lines
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank()) +  # Axis ticks color
  scale_color_manual(values = c('green','blue')) # Customize colors
```




```
# Plot the results using ggplot
ggplot(df_long %>% filter(type %in% c("shrinkage","true_KME")), aes(x = vec_fixed_points, y = estimatio
  geom_hline(yintercept = 0, color = "gray", linetype = "solid", linewidth = 0.5) +  # Add axis y = 0
  geom_vline(xintercept = 0, color = "gray", linetype = "solid", linewidth = 0.5) +  # Add axis at x =
  geom_point(data = df_fixed_points, aes(x = x, y = y), color = 'red', size = 2 ,shape =3) +  # Plot x
  geom_line(linewidth = 1.2) + # Plot kernel mean embedding estimator
  labs(x = "x",
       y = "Estimator $\\hat{\\mu}_{\\mathbb{P}}(x)",
       title = "Estimator of Kernel Mean Embedding")+
  theme_bw()+
  theme(panel.grid = element_blank(),  # Remove grid lines
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank()) +  # Axis ticks color
  scale_color_manual(values = c('green3','blue')) # Customize colors
```



Find the probabilities using KME and centered kernel

The relationship for $\hat{\mu}(\mathbf{x}_i)$ is given by:

$$\hat{\mu}(\mathbf{x}_i) = \sum_{j=1}^{n} h(\mathbf{x}_i, \mathbf{x}_j) p_\theta(\mathbf{x}_j).$$

Furthermore, we can estimate $\hat{\mu}(\mathbf{x}_i)$ using methods like shrinkage or Bayesian approaches: Moreover, we can estimate $\hat{\mu}(\mathbf{x}_i)$ with methods like shrinkage methods provided by Gretton and colleagues. So we can say $\hat{\mathbf{p}}_\theta = \mathbf{H}_{sample}^{-1} \hat{\boldsymbol{\mu}}$

This allows us to form equations where only θ's are unknown: So we can make equations that only θ's are unknown, and we can find these parameters using estimated kernel mean embeddings.

Recall that the relationship for $p_\theta(\mathbf{x}_j)$ is:

$$\hat{p}_\theta(\mathbf{x}_j) = \frac{\exp(\hat{\theta}_n(\mathbf{x}_j))}{\sum_{j=1}^{n} \exp(\hat{\theta}_n(\mathbf{x}_j))}.$$

Specify the grid and centering grid

```
sampled_x <- vec_fixed_points
x_grid <-  seq(-7,7,length.out = 1000)
# centering_grid <- sampled_x This doesn't work because using this centering grid the kernel mean embed
centering_grid <- runif(min = -7,max = 7,n = 1000)
```

Find Kernel Matrices:

$\mathbf{H}_{sample}$:

```
centered_kernel_mat_at_sampled <- centered_kernel_matrix(first_vec_kernel = sampled_x,
                                                         second_vec_kernel = sampled_x,
                                                         centering_grid = centering_grid,
                                                         hurst_coef = 0.5)
```

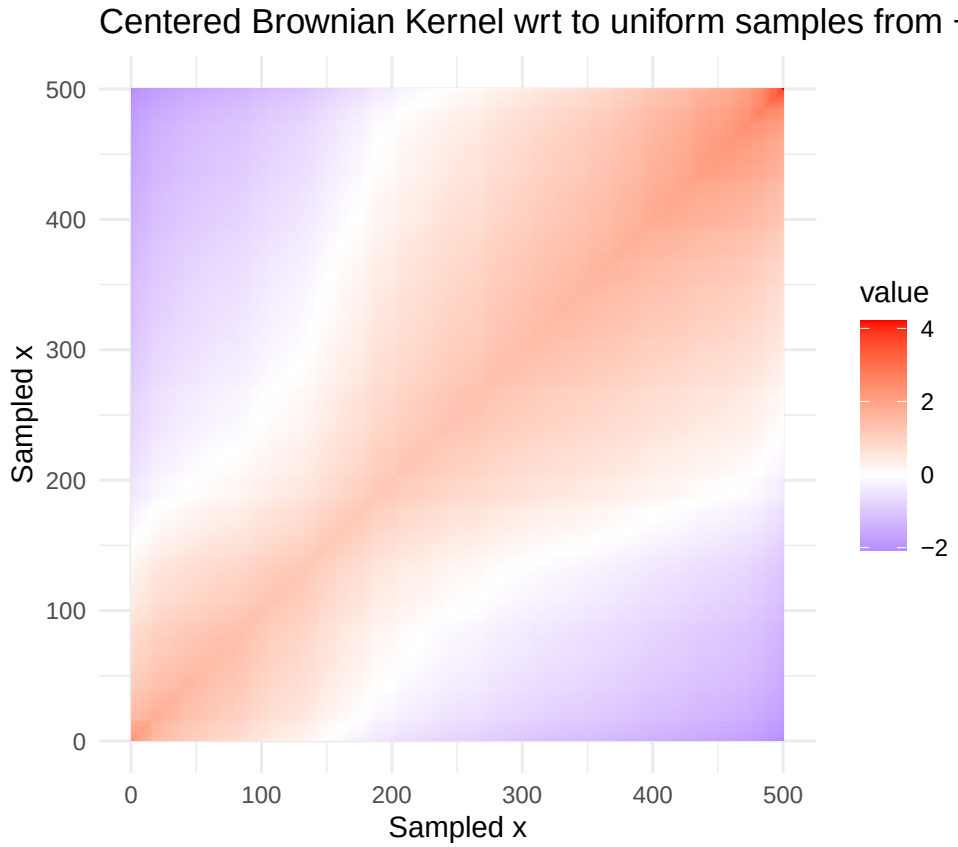
```
library(ggplot2)
library(reshape2)
```

```
## Warning: package 'reshape2' was built under R version 4.3.2
```

```
##
## Attaching package: 'reshape2'
```

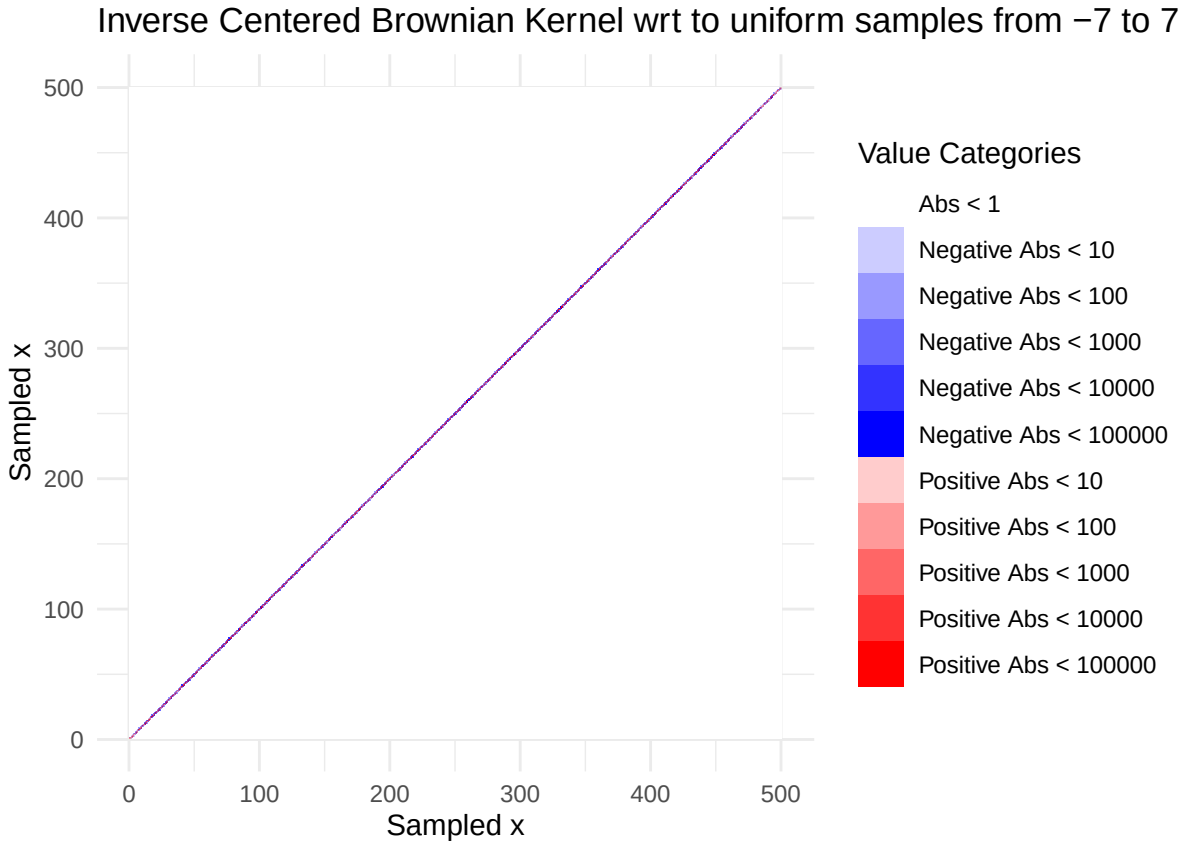
```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
# Convert matrix to a data frame
matrix_df <- melt(centered_kernel_mat_at_sampled)

# Plot using ggplot2
ggplot(matrix_df, aes(Var1, Var2, fill = value)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", mid = "white", high = "red") +
  theme_minimal() +
  coord_fixed() +  # Ensures the plot is square
  labs(title = "Centered Brownian Kernel wrt to uniform samples from -7 to 7", x = "Sampled x", y = "Sa
```

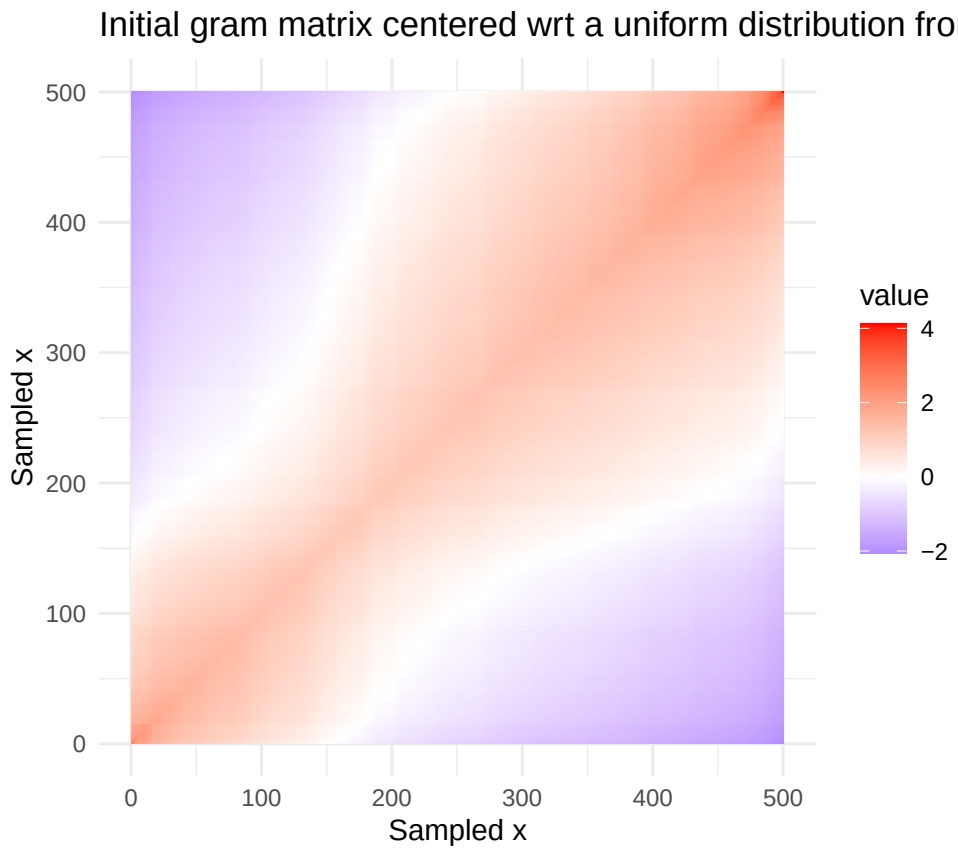



```r
library(ggplot2)
library(reshape2)
library(dplyr)

# Convert matrix to a data frame
matrix_df <- melt(solve(centered_kernel_mat_at_sampled))

# Define the color categories based on the absolute value and sign of 'value'
matrix_df <- matrix_df %>%
  mutate(category = case_when(
    abs(value) < 1 ~ "Abs < 1",
    value > 0 & abs(value) < 10 ~ "Positive Abs < 10",
    value < 0 & abs(value) < 10 ~ "Negative Abs < 10",
    value > 0 & abs(value) < 100 ~ "Positive Abs < 100",
    value < 0 & abs(value) < 100 ~ "Negative Abs < 100",
    value > 0 & abs(value) < 1000 ~ "Positive Abs < 1000",
    value < 0 & abs(value) < 1000 ~ "Negative Abs < 1000",
    value > 0 & abs(value) < 10000 ~ "Positive Abs < 10000",
    value < 0 & abs(value) < 10000 ~ "Negative Abs < 10000",
    value > 0 & abs(value) < 100000 ~ "Positive Abs < 100000",
    value < 0 & abs(value) < 100000 ~ "Negative Abs < 100000",
    TRUE ~ "Other"
  ))

# Map the categories to colors
category_colors <- c(
```

```
  "Abs < 1" = "white",
  "Positive Abs < 10" = rgb(1, 0, 0, alpha = 0.2),
  "Negative Abs < 10" = rgb(0, 0, 1, alpha = 0.2),
  "Positive Abs < 100" = rgb(1, 0, 0, alpha = 0.4),
  "Negative Abs < 100" = rgb(0, 0, 1, alpha = 0.4),
  "Positive Abs < 1000" = rgb(1, 0, 0, alpha = 0.6),
  "Negative Abs < 1000" = rgb(0, 0, 1, alpha = 0.6),
  "Positive Abs < 10000" = rgb(1, 0, 0, alpha = 0.8),
  "Negative Abs < 10000" = rgb(0, 0, 1, alpha = 0.8),
  "Positive Abs < 100000" = rgb(1, 0, 0, alpha = 1),
  "Negative Abs < 100000" = rgb(0, 0, 1, alpha = 1),
  "Other" = "black"
)

# Plot using ggplot2
ggplot(matrix_df, aes(Var1, Var2, fill = category)) +
  geom_tile() +
  scale_fill_manual(values = category_colors, name = "Value Categories") +
  theme_minimal() +
  coord_fixed() +  # Ensures the plot is square
  labs(
    title = "Inverse Centered Brownian Kernel wrt to uniform samples from -7 to 7",
    x = "Sampled x", y = "Sampled x"
  )
```



```r
# Convert matrix to a data frame
matrix_df <- melt(gram)

# Plot using ggplot2
ggplot(matrix_df, aes(Var1, Var2, fill = value)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", mid = "white", high = "red") +
  theme_minimal() +
  coord_fixed() +  # Ensures the plot is square
  labs(title = "Initial gram matrix centered wrt a uniform distribution from -7 to 7", x = "Sampled x",
```




```r
# Positive Definiteness of centered_kernel_mat_at_sampled

# Get the eigenvalues
eigenvalues <- eigen(centered_kernel_mat_at_sampled)$values

# Check if all eigenvalues are positive
is_positive_definite <- all(eigenvalues > 0)
is_positive_definite
```

```
## [1] TRUE
```

```r
# Check if the matrix is symmetric
is_symmetric <- all(centered_kernel_mat_at_sampled == t(centered_kernel_mat_at_sampled))

is_symmetric
```

```
## [1] FALSE
```

```
all(df_reg$regularized>0)
```

```
## [1] TRUE
```

$\mathbf{H}_{grid}$:

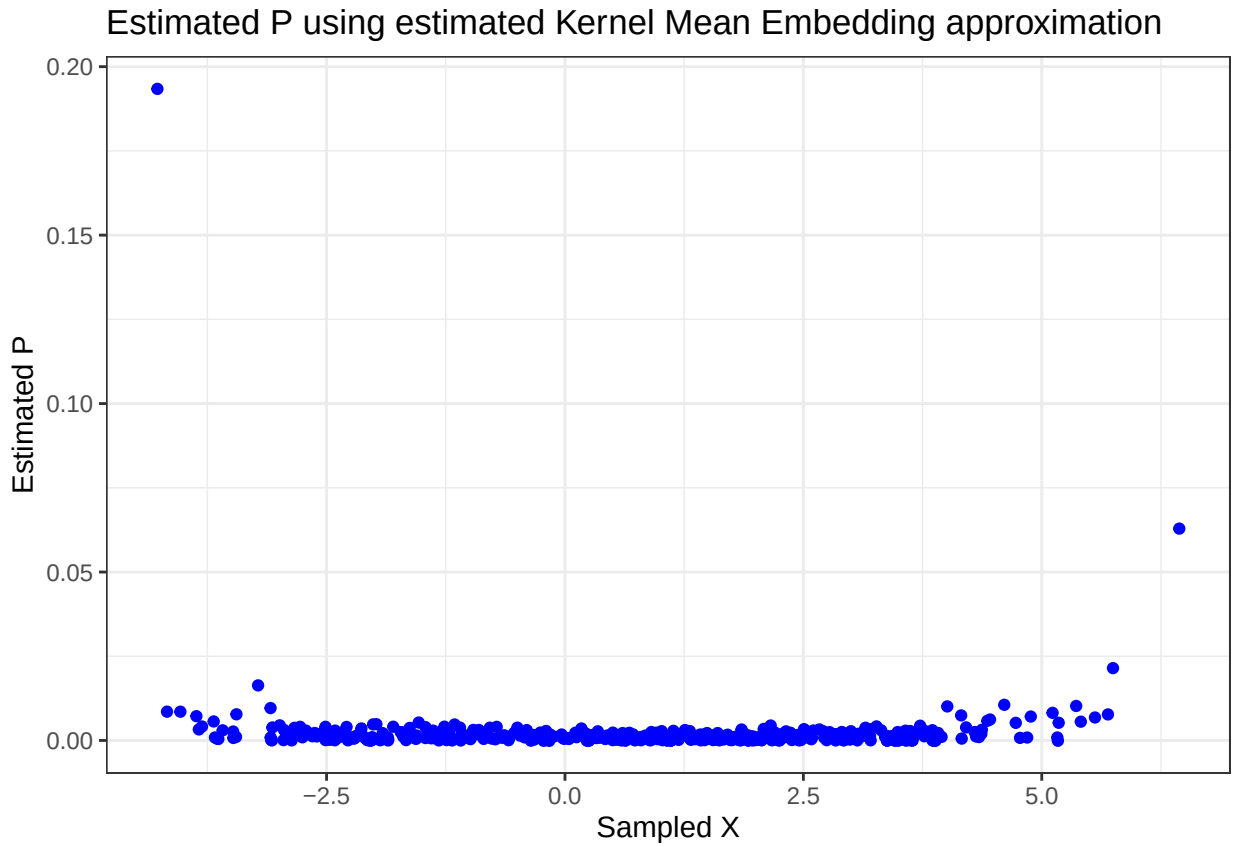
```
centered_kernel_mat_at_grid <- centered_kernel_matrix(first_vec_kernel = sampled_x,
                                                      second_vec_kernel = x_grid,
                                                      centering_grid = centering_grid,
                                                      hurst_coef = 0.5)
```

Find the probabilities:

```
p <- solve(centered_kernel_mat_at_sampled) %*% df_all$regularized
sum(p)
```

```
## [1] 4.743184
```

```
p <- p/sum(p)

plot_data <- data.frame(x = vec_fixed_points, prob = p)

ggplot(plot_data, aes(x,prob)) + geom_point(color = "blue")  + labs(title = "Estimated P using estimate
       x = "Sampled X", y = "Estimated P") + theme_bw()
```



Find ln(p):

```
ln_p <- log(p - 2* min(p))
```

Find U weights:

```
one_n <- rep(1, length(sampled_x))
one_m <- rep(1, length(x_grid))

u_vec <-  one_m %*% t(centered_kernel_mat_at_grid) %*% solve(centered_kernel_mat_at_sampled)
```

Find the θ's:

```
C <-  - as.vector(u_vec %*% ln_p) / as.vector(one_n %*% ln_p)

theta <- ln_p + C
```

Estimate the probabilities:

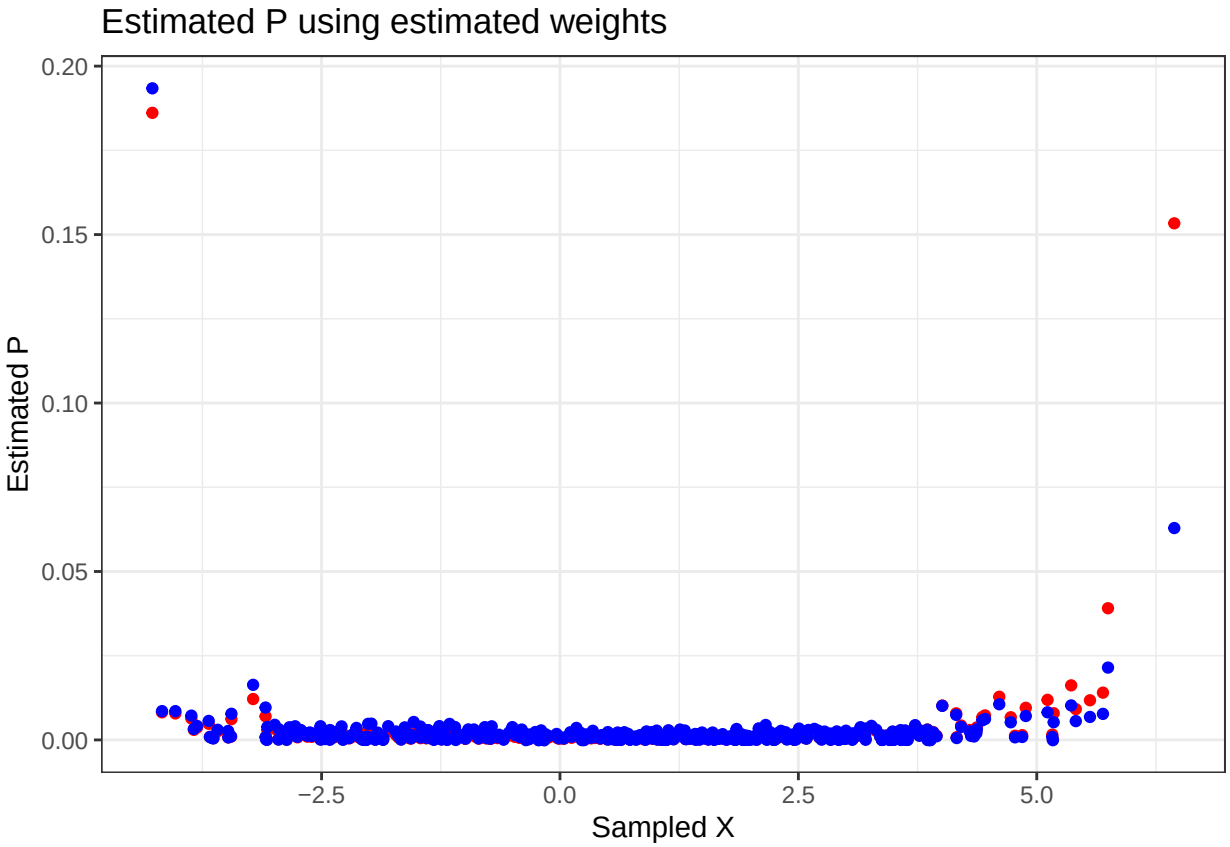
```
centered_kernel_self_grid <- diag(centered_kernel_mat_at_sampled)

estimated_p <- exp(theta + 0.5 * centered_kernel_self_grid)
```

```
sum(estimated_p,na.rm = T)
```

```
## [1] 2.941353
```

Show the result:

```
# Combine into a data frame
df_result <- data.frame(sampled_x, estimated_p)


# Create the ggplot
ggplot() +
  geom_point(data = df_result, aes(x = sampled_x, y = estimated_p/sum(estimated_p)), color = "red") +
  geom_point(data = plot_data, aes(x = x, y = prob), color = "blue") +                    # Blue points
  labs(title = "Estimated P using estimated weights",
       x = "Sampled X", y = "Estimated P") +
  theme_bw()
```


Estimated P using estimated weights
0.20
0.15
0.10
0.05
0.00
Estimated P
−2.5
0.0
2.5
5.0
Sampled X