Stat 3005 – Nonparametric Statistics

Final project

Time limit: 72 hours

Fall 2024

Instructions

1. The full mark of the project is 100.
2. There is an optional bonus question that is worth 10 marks. (E.g., if you get 95 + 10, your project score is 100.)
3. Complete the project solely by yourself. No collaboration is allowed. You may consult lecture notes, tutorial notes, homework assignments, quizzes, textbooks, online materials, etc, however, you cannot solicit or obtain assistance from or provide assistance to other people in any form for any specific content on the project. Activities considered cheating include, but are not limited to, copying, rephrasing or modifying contents from websites, discussing project questions with other people, and asking for help on Internet forums. All cheating cases will be passed to the the Senate Committee on Student Discipline with no exception.
4. Generative AI tools (e.g., ChatGPT) can be used for auxiliary purposes so that the answers reflect your own intellectual ideas and independent effort. The graders have the ultimate power to decide the appropriateness and the level of originality of the submitted answers.
5. Quote the results that appear in the lecture notes if you wish to use them, e.g., write "by Theorem 3.1", "by the CLT for rank statistics", etc.
6. Prove the results that do not appear in lecture notes if you wish to use them.
7. Define your abbreviations clearly unless they have been defined in the lecture notes.
8. Submit the following two documents to Blackboard before the deadline.
   - Written part. Either (i) handwrite and scan your answers, or (ii) type your answers in LaTeX. Use A4-size papers. Compile your answers in one single `.pdf` file. Sign the Honor Code below, and attach this page as a cover of your submitted file. Name the document in the format `S3005_F_sid_name.pdf`. E.g., `S3005_F_1155001234_ChanKinWai.pdf`. Note that all plots, numerical answers, simulation results, etc must be included in the written part. Graders will not run your submitted codes to check the answers.
   - Computation part. Save your `R` codes in one single `.r` file and name it in the formate `S3005_F_sid_name.r`. E.g., `S3005_F_1155001234_ChanKinWai.r`.

   Submission without properly signed honor code will not be graded. Submissions with incorrect formats will not be graded. Unclear or blurred answers will not be graded. Late submission will not be graded. You may re-submit your answers as many time as you wish, however, only the last submission will be graded.

Honor Code

Please read the Honor Code below and sign your name.

The Chinese University of Hong Kong places very high importance on honesty in academic work submitted by students, and adopts a policy of zero tolerance on academic dishonesty. All work presented in this project, including ideas, discoveries, interpretations, codes, derivations, and conclusions, should solely reflect independent effort, and strictly adhere to the standard of academic integrity and the instructions above (see the academic honesty guide provided by CUHK for more details).

I understand the above instructions, and affirm that all work conforms to the standards of the Honor Code.

Signature:	[illegible]
Print Name:	Lam Hoi Chun
Student ID:	1155192755
Date:	15-12-2024

STAT3005 Final Project
Lam Hoi Chun (1155192755)

Question 1

For each of the following tests, simulations are carried out to test the three desired results. For H_0, we assume that $X_{1:n} \overset{\text{IID}}{\sim} N(1,1)$ and $Y_{1:n} \overset{\text{IID}}{\sim} N(1,1)$, where $X_{1:n}$ and $Y_{1:n}$ are two independent random samples. Before reporting the result, the intermediate steps are shown, which are the size computed under H_0, the power under H_1 and the p-values computed using the given dataset (code in Appendix). Note that the power is computed based on $X_{1:n} \overset{\text{IID}}{\sim} N(2,2^2)$ and $Y_{1:n} \overset{\text{IID}}{\sim} N(1,1)$.

Test	Arnold	Brian	Calvin	Daniel	Edwin	Francis	Gavin	Hannah
(i) Size under H_0	0.0498	0.0488	0.0965	0.234	0.2437	0.058	0.0529	0.072
(ii) Power under H_1	0.6425	0.6839	0.9266	0.97	0.9635	0.648	0.6412	0.970
(iii) p-value / CI	0.0117	0.1402	0.0117	0.0003	0.000191	0.0096	0.0041	(-5541.17, 717.28)

Then, for each test, we argue whether it has a reasonably high power under H_1.

1. Rank sum test: It has a power of 0.6425, which is not particularly high. Although it detects differences in location well, its power is limited as it cannot detect the difference in variances.
2. Ansari-Bradley test: It has a power of 0.6839, which is not particularly high. Although it detects differences in scale well, its power is limited as it cannot detect the difference in location.
3. Calvin's test: It has a power of 0.9266, indicating a high power. As it combines rank sum test and Ansari-Bradley test, it can detect the differences in both location and scale, leading to a high power.
4. Cramér–von Mises test: It has a power of 0.97, indicating a high power. As it is sensitive to general distributional differences, it can detect the differences in both location and scale, leading to a high power.
5. `ks.test(x, ecdf(y))`: It has a power of 0.9635, indicating a high power. As it is sensitive to general distributional differences, it can detect the differences in both location and scale, leading to a high power.
6. Permutation 2-sample t-test: It has a power of 0.648, which is not particularly high. Although it detects differences in location well, its power is limited as it cannot detect the difference in variances.
7. Paired exact sign-rank test: It has a power of 0.6412, which is not particularly high. Although it detects differences in location well, its power is limited as it cannot detect the difference in variances.
8. 95% bootstrap CI: It has a power of 0.970, indicating a high power. As it bootstraps the samples and estimate the variance difference, it can detect the differences in both location and scale, leading to a high power.

Question 1 (Cont'd)

Before reporting the result, there are some remarks:

- Row (i) is determined by whether the simulated size under H_0 is close to 0.05
- Row(ii) is determined by whether the power is high under H_1
- Row(iii) is determined by whether the test's p-value is smaller than or equal to 0.05
- $H_0 : X_{1:n} \overset{\text{IID}}{\sim} N(1,1)$ and $Y_{1:n} \overset{\text{IID}}{\sim} N(1,1)$ and $H_1 : X_{1:n} \overset{\text{IID}}{\sim} N(2,2^2)$ and $Y_{1:n} \overset{\text{IID}}{\sim} N(1,1)$ are considered in the simulation result.

Based on the simulation result, we can report the following.

Test	Arnold	Brian	Calvin	Daniel	Edwin	Francis	Gavin	Hannah
(i) has correct size	✓	✓				✓	✓	
(ii) has high power			✓	✓	✓			✓
(iii) reject the null	✓		✓	✓	✓	✓	✓	

Question 2 (1a)

We first let $u = (x - X_i)/\ell$, so $du/dx = 1/\ell$. Then,

$$\begin{aligned}
\widehat{f}_\ell(x) &= \frac{1}{n\ell}\sum_{i=1}^{n} K\left(\frac{X_i - x}{\ell}\right) \\
&= \frac{1}{n\ell}\sum_{i=1}^{n} K\left(\frac{x - X_i}{\ell}\right) \\
\widehat{f}_\ell^{(1)}(x) &= \frac{1}{n\ell}\sum_{i=1}^{n} K'\left(\frac{X_i - x}{\ell}\right) \cdot \frac{d}{dx}\left(\frac{X_i - x}{\ell}\right) \\
&= \frac{1}{n\ell}\sum_{i=1}^{n} K'(u) \cdot \frac{1}{\ell} \\
&= \frac{1}{n\ell^2}\sum_{i=1}^{n} K'(u) \\
\widehat{f}_\ell^{(2)}(x) &= \frac{1}{n\ell^2}\sum_{i=1}^{n} K''(u) \cdot \frac{du}{dx} \\
&= \frac{1}{n\ell^3}\sum_{i=1}^{n} K''(u)
\end{aligned}$$

Therefore, we can derive that the general formula is

$$\widehat{f}_\ell^{(r)}(x) = \frac{1}{n\ell^{r+1}}\sum_{i=1}^{n} K^{(r)}\left(\frac{x - X_i}{\ell}\right)$$

Question 2 (1b)

Note that the following are some important R functions used in this question.

```r
ddnorm = function(x,mu,sigma,r=0){
 if(r==0){
   out = dnorm(x,mu,sigma)
 }
 if(r==1){
   out = dnorm(x,mu,sigma)*(-1/sigma^2)*(x-mu)
 }
 if(r>=2){
   out = -(r-1)/sigma^2*ddnorm(x,mu,sigma,r=r-2)-(x-mu)/sigma^2*ddnorm(x,mu,
      sigma,r=r-1)
 }
 out
}

compute_lambda_K_r <- function(r) {
  integrate(function(t){(ddnorm(t,0,1,r=r))^2}, lower=-Inf, upper=Inf)$value
}

compute_int_df_square <- function(r, sigma_f) {
  integrate(function(t){(ddnorm(t,0,sigma_f,r=r+2))^2},
              lower=-Inf, upper=Inf)$value
}
```

Given that

$$\ell_r^* = \left[\frac{(2r+1)\lambda_K^{(r)}}{n\sigma_K^4\int_{-\infty}^{\infty}\{f^{(r+2)}(t)\}^2\,dt}\right]^{\frac{1}{2r+5}}$$

When $r=0$

$$\begin{aligned}
\ell_0^* &= \left[\frac{(2\cdot 0+1)\lambda_K^{(0)}}{n\sigma_K^4\int_{-\infty}^{\infty}\{f^{(0+2)}(t)\}^2\,dt}\right]^{\frac{1}{2\cdot 0+5}} \\
&= \left[\frac{\int_{-\infty}^{\infty}[K(t)]^2dt}{n\cdot(1)\cdot\int_{-\infty}^{\infty}[f''(t)]^2dt}\right]^{1/5} \\
&= \left[\frac{\int_{-\infty}^{\infty}[K(t)]^2dt}{n\cdot\int_{-\infty}^{\infty}[f''(t)]^2dt}\right]^{1/5}
\end{aligned}$$

Question 2 (1b) (Cont'd)

We can find the coefficient of ℓ_0^* using the following R code:

```r
# Compute lambda_K_0
lambda_K_0 = compute_lambda_K_r(0)
# Output = 0.2820948

# Compute (f^{(0+2)}(t))^2 dt
int_df_2_square = compute_int_df_square(0,1)
# Output = 0.2115711

# Required coefficient
(lambda_K_0/int_df_2_square)^(1/5)
# Output = 1.059224
```

Then, we consider

$$\begin{aligned}
f(t) &= \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{t}{2\sigma^2}\right)\\
f'(t) &= -\frac{t}{\sigma^3\sqrt{2\pi}} \exp\left(-\frac{t}{2\sigma^2}\right)\\
f''(t) &= \frac{t^2-\sigma^2}{\sigma^5\sqrt{2\pi}} \exp\left(-\frac{t}{2\sigma^2}\right)\\
f''(t)^2 &= \frac{(t^2-\sigma^2)^2}{2\pi\sigma^{10}} \exp\left(-\frac{t}{\sigma^2}\right)\\
\int_{-\infty}^{\infty} f''(t)^2 dt &= \frac{1}{2\pi\sigma^{10}} \int_{-\infty}^{\infty} (t^2-\sigma^2)^2 \exp\left(-\frac{t}{\sigma^2}\right) dt\\
\text{Let } z &= \frac{t}{\sigma}, \text{ we have}\\
\int_{-\infty}^{\infty} f''(t)^2 dt &= \frac{1}{2\pi\sigma^5} \int_{-\infty}^{\infty} (z^4-2z^2+1)\cdot\exp(-z^2) dz
\end{aligned}$$

Therefore, we know the index of the σ and we can have

$$\begin{aligned}
\ell_0^* &\approx 1.059 \cdot \left(\frac{1}{n/\sigma^5}\right)^{1/5}\\
&= 1.059\sigma/n^{1/5}
\end{aligned}$$

Question 2 (1b) (Cont'd)

When $r = 2$,

$$\begin{aligned}
\ell_2^* &= \left[\frac{(2\cdot 2+1)\lambda_K^{(2)}}{n\sigma_K^4\int_{-\infty}^{\infty}\{f^{(2+2)}(t)\}^2\,dt}\right]^{\frac{1}{2\cdot 2+5}} \\
&= \left[\frac{5\lambda_K^{(2)}}{n\sigma_K^4\int_{-\infty}^{\infty}\{f^{(4)}(t)\}^2\,dt}\right]^{1/9} \\
&= \left[\frac{5\int_{-\infty}^{\infty}[K''(t)]^2}{n\sigma_K^4\int_{-\infty}^{\infty}\{f^{(4)}(t)\}^2\,dt}\right]^{1/9}
\end{aligned}$$

We can find the coefficient of ℓ_2^* using the following R code:

```r
# Compute lambda_K_2
lambda_K_2 = compute_lambda_K_r(2)
# Output = 0.2115711

# Compute (f^{(0+2)}(t))^2 dt
int_df_4_square = compute_int_df_square(2,1)
# Output = 1.851247

# Required coefficient
(5*lambda_K_2/int_df_4_square)^(1/9)
# Output = 0.9397142
```

Then, we consider

$$\begin{aligned}
f^{(3)}(t) &= \frac{t^3 - 3t\sigma^2}{\sigma^7\sqrt{2\pi}}\exp\left(-\frac{t}{2\sigma^2}\right) \\
f^{(4)}(t) &= \frac{t^4 - 6t^2\sigma^2 + 3\sigma^4}{\sigma^9\sqrt{2\pi}}\exp\left(-\frac{t}{2\sigma^2}\right) \\
f^{(4)}(t)^2 &= \frac{(t^4 - 6t^2\sigma^2 + 3\sigma^4)^2}{2\pi\sigma^{18}}\exp\left(-\frac{t}{\sigma^2}\right) \\
\int_{-\infty}^{\infty} f^{(4)}(t)^2 dt &= \frac{1}{2\pi\sigma^{18}}\int_{-\infty}^{\infty}(t^4 - 6t^2\sigma^2 + 3\sigma^4)^2\exp\left(-\frac{t}{\sigma^2}\right)dt \\
&= \frac{1}{2\pi\sigma^{18}}\int_{-\infty}^{\infty}(t^8 - 12t^6\sigma^2 + 42t^4\sigma^4 - 36t^2\sigma^6 + 9\sigma^8)\exp\left(-\frac{t}{\sigma^2}\right)dt \\
\text{Let } z &= \frac{t}{\sigma}, \text{ we have} \\
\int_{-\infty}^{\infty} f^{(4)}(t)^2 dt &= \frac{1}{2\pi\sigma^9}\int_{-\infty}^{\infty}(z^8 - 12z^6 + 54z^4 - 108z^2 + 9)\cdot e^{-z^2}dz
\end{aligned}$$

Therefore, we know the index of the σ and we can have

$$\begin{aligned}
\ell_2^* &\approx 0.940\cdot\left(\frac{1}{n/\sigma^9}\right)^{1/9} \\
&= 0.940\sigma/n^{1/9}
\end{aligned}$$

Question 2 (1c)

The R function `kdeDerivative = function(xeval, data, r=0)` is as follows:

```r
kdeDerivative = function(xeval, data, r=0) {
  n = length(data)
  sigma_hat = sd(data)
  IQR_hat = IQR(data)
  s_hat = min(sigma_hat, IQR_hat/1.34)
  sigma_K_fourth = 1

  # Compute lambda_K^{(r)}
  lambda_K_r = compute_lambda_K_r(r)

  # Compute integral for f^{(r+2)}
  int_df_square = compute_int_df_square(r, s_hat)

  # Compute ell_r*
  ell=(((2*r+1)*lambda_K_r)/(n*sigma_K_fourth*int_df_square))^(1/(2*r+5))*s_hat

  # Compute Z_ij = (x_j - X_i)/ell
  Z = outer(xeval, data, FUN=function(x, Xi) (x - Xi)/ell)

  # Compute the Summation of K^(r)(Z)
  K_r_Z = ddnorm(Z, 0, 1, r=r)
  sum_K_r = rowSums(K_r_Z)

  # Final output
  f_r_est = 1/(n*ell^(r+1))*sum_K_r
  f_r_est
}
```

Question 2.2

First, we can compute the ballon estimator $\widehat{f}_{\text{BAL}}(x)$ using the following self-written R function `kdeBallon = function(xeval, data)`.

```r
kdeBallon = function(xeval, data) {
  n = length(data)
  lambda_K0 = compute_lambda_K_r(0)
  sigma_K_4 = 1

  # Compute the pilot estimates: f(x) and f''(x)
  f_est = kdeDerivative(xeval, data, r=0)
  f_2nd_est = kdeDerivative(xeval, data, r=2)
  # Compute ell*(x)
  ell_x = ((f_est*lambda_K0)/(n*(f_2nd_est^2)* sigma_K_4))^(1/5)
  # Compute the balloon estimator
  results = numeric(length(xeval))
  for (j in 1:length(xeval)) {
    x = xeval[j]
    ell_j = ell_x[j]
    Z = (x - data) / ell_j
    K_values = dnorm(Z, 0, 1)
    f_Bal_x = (1/(n * ell_j)) * sum(K_values)
    results[j] = f_Bal_x
  }
  results
}
```

Then, we would like to compare the ballon estimate with the true density and the standard KDE using the following non-adaptive bandwidths: (i). Rule of thumb bandwidth, (ii). Cross Validation bandwidth and (iii). Plug in bandwidth.

Thus, we can find the different densities using the following R code.

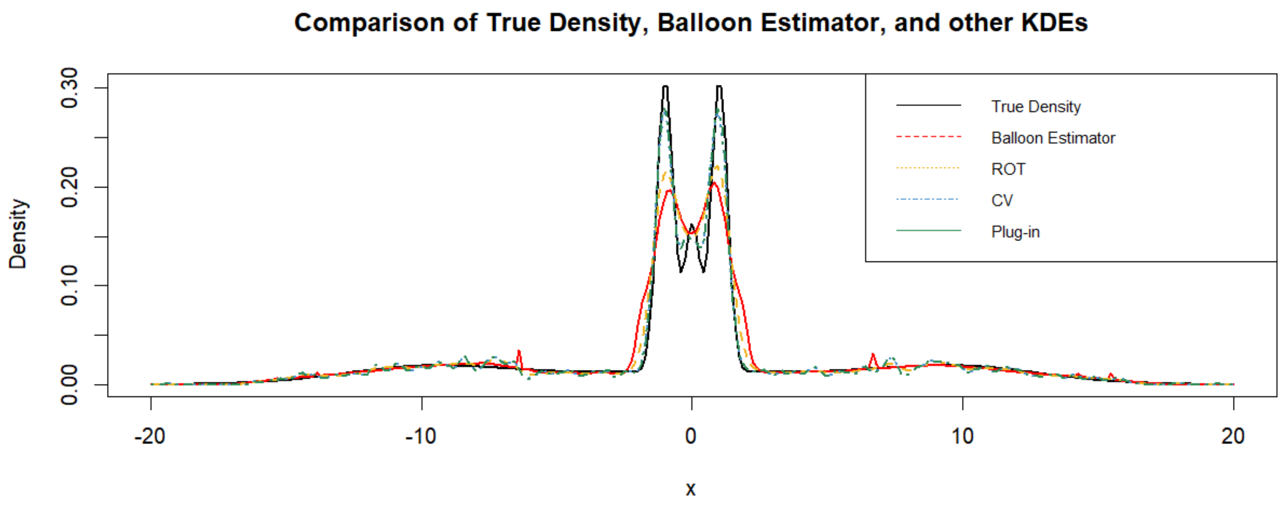
```r
data = read.csv("Q2.csv")
n = 3005
mu = c(-10, -3, -1, 0, 1, 3, 10)
sigma = c(3, 6, 0.3, 0.3, 0.3, 6, 3)
weights = c(1,1,2,1,2,1,1)/9
t = seq(from=-20,to=20,length=301)

# True density
f = colSums(simplify2array(lapply(t,function(t)dnorm(t, mu,sigma)))*p)

# Balloon estimate
f_ballon = kdeBallon(t, data$x)

# Rule of Thumb BW
f_rot = density(data$x, bw=bw.nrd0(data$x), from=-20, to=20, n=301)$y

# Cross Validation BW
f_cv = density(data$x, bw=bw.ucv(data$x), from=-20, to=20, n=301)$y

# Plug-in BW
f_pin = density(data$x, bw=bw.SJ(data$x), from=-20, to=20, n=301)$y
```

Question 2.2 (Cont'd)

Then, we can generate the plot for comparison using the following R code.

```r
# Plot
plot(t, f, type="l", lwd=2, col="black",
     ylim=c(0, max(f, f_ballon, f_rot, f_cv, f_pin)),
     main="Comparison of True Density, Balloon Estimator, and other KDEs",
     xlab="x", ylab="Density")
lines(t, f_ballon, col="red", lwd=2, lty=1)
lines(t, f_rot, col="goldenrod2", lwd=2, lty=2)
lines(t, f_cv, col="steelblue3", lwd=2, lty=3)
lines(t, f_pin, col="seagreen4", lwd=2, lty=4)
legend("topright",
       legend=c("True Density", "Balloon Estimator", "ROT", "CV", "Plug-in"),
       col=c("black","red","goldenrod2","steelblue3","seagreen4"),
       lty=c(1,2,3,4), lwd=1, cex=0.75)
```

And the following plot is generated.



Key findings

The ballon estimator perform poorly around the peaks of the true density. A possible reason is that it adjusts its bandwidth locally based on estimates of the density and its derivatives. However, the derivatives near the peaks may be fluctuated, thus leading errors in selecting bandwidths.

For non-adaptive bandwidths, the cross validation bandwidth and plug-in bandwidth perform nearly the same, while the rule of thumb bandwidth perform the worst. In fact, three of them do not provide a good estimation. A possible reason is that the bandwidth selections do not change based on the data's local structure. A single bandwidth cannot simultaneously handle flat regions and highly peaked areas well.

The particular bad performance of the rule of thumb bandwidth may be explained by its use of sample standard deviation during calculation. This scale parameter is insufficient to represent the spread of the data with a true density of complex multi-modal structure.

Question 2.3

First, we can compute the sample point estimator $\widehat{f}_{\text{SAM}}(x)$ using the following self-written R function `kdeSample = function(xeval, data)`.

```
kdeSample = function(xeval, data) kdeSample = function(xeval, data) {
  n = length(data)

  # Plug-in BW
  ell_0 = bw.SJ(data)

  # Compute the kernel input
  Z = outer(data, data, FUN=function(xi, xj) (xi - xj)/ell_0)
  phi_Z = dnorm(Z)

  hat_f_ell0_Xi = rowSums(phi_Z) / (n * ell_0)
  bar_f_GM = exp(mean(log(hat_f_ell0_Xi)))
  ell_i = ell_0 * sqrt(bar_f_GM / hat_f_ell0_Xi)

  f_sam = numeric(length(xeval))
  for (k in 1:length(xeval)) {
    x = xeval[k]
    Zx = (x - data)/ell_i
    phi_Zx = dnorm(Zx)
    f_sam[k] = mean(phi_Zx / ell_i)
  }
  f_sam
}

# Density for sample-point estimator
f_sam = kdeSample(t, data$x)
```

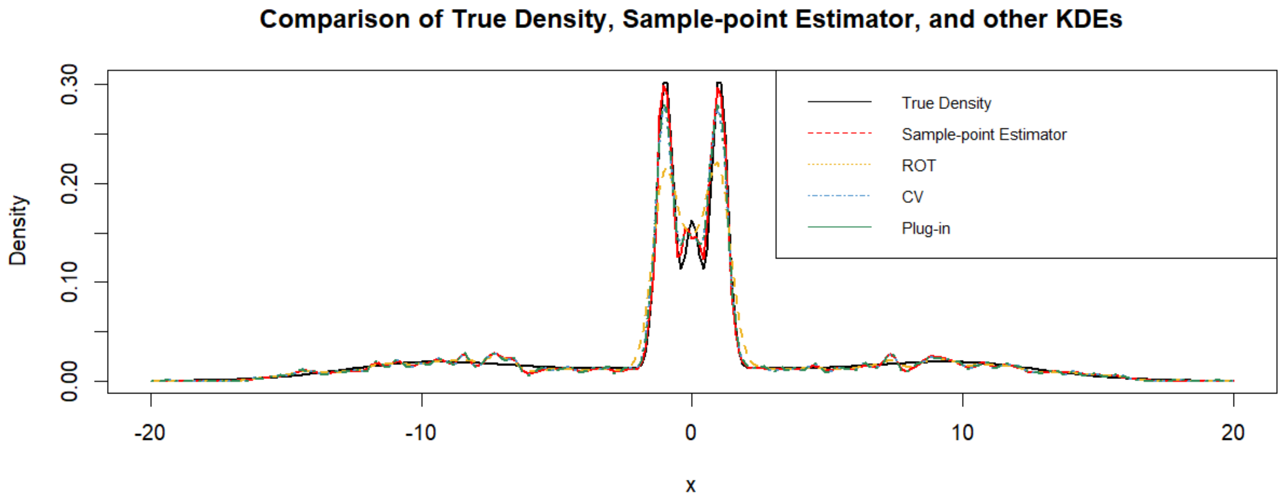
Similar to Question 2.2, we would like to compare the sample-point estimate with the true density and the standard KDE using the following non-adaptive bandwidths: (i). Rule of thumb bandwidth, (ii). Cross Validation bandwidth and (iii). Plug in bandwidth.

We can generate the plot for comparison using the following R code.

```
# Plot
plot(t, f, type="l", lwd=2, col="black",
     ylim=c(0, max(f, f_sam, f_rot, f_cv, f_pin)),
     main="Comparison of True Density, Sample-point Estimator, and other KDEs",
     xlab="x", ylab="Density")
lines(t, f_sam, col="red", lwd=2, lty=1)
lines(t, f_rot, col="goldenrod2", lwd=2, lty=2)
lines(t, f_cv, col="steelblue3", lwd=2, lty=3)
lines(t, f_pin, col="seagreen4", lwd=2, lty=4)
legend("topright",
       legend=c("True Density", "Sample-point Estimator","ROT","CV","Plug-in"),
       col=c("black","red","goldenrod2","steelblue3","seagreen4"),
       lty=c(1,2,3,4), lwd=1, cex=0.75)
```

Question 2.3 (Cont'd)

And the following plot is generated.



Key findings

The sample-point estimator has a very good performance. A possible reason is that for each X_i, it adapts the bandwidth by scaling the bandwidth ℓ_0 inversely with $\widehat{f}_{\ell_0}(X_i)$. Therefore, for the high density region, ℓ_0 becomes small, vise versa. It ensures that the estimator captures sharp peaks while maintaining flatter regions, leading to its strong performance.

The sample-point estimator makes uses of the plug-in bandwidth and the former performs better. As mentioned last paragraph, the sample-point estimator apply local scaling, which could solve the shortcomings of the global plug-in bandwidth by adapting to the varying density structure,

Question 3.1

Kruskal–Wallis statistic T is non-parametric as it only relies on the rank of the observations and assume the data are IID continuous, without any distributional assumption, while the parametric F-statistic is derived under normality assumption.

Question 3.2

$$\begin{aligned}
T &= \frac{12}{N(N+1)} \sum_{j=1}^{k} n_j (\bar{R}_{.j} - \bar{R}_{..})^2 \\
&= \frac{12}{N(N+1)} \sum_{j=1}^{k} n_j (\bar{R}_{.j}^2 - 2\bar{R}_{..}\bar{R}_{.j} + \bar{R}_{..}^2) \\
&= \frac{12}{N(N+1)} \left(\sum_{j=1}^{k} n_j \bar{R}_{.j}^2 - 2\bar{R}_{..} \sum_{j=1}^{k} n_j \bar{R}_{.j} + \sum_{j=1}^{k} n_j \bar{R}_{..}^2 \right) \\
&= \frac{12}{N(N+1)} \left[\sum_{j=1}^{k} n_j \bar{R}_{.j}^2 - 2\left(\frac{N+1}{2}\right)\left(\frac{N(N+1)}{2}\right) + N\left(\frac{N(N+1)}{2}\right)^2 \right] \\
&= \frac{12}{N(N+1)} \left[\sum_{j=1}^{k} n_j \bar{R}_{.j}^2 - \frac{N(N+1)^2}{4} \right] \\
&= -3(N+1) + \frac{12}{N(N+1)} \sum_{j=1}^{k} n_j \bar{R}_{.j}^2
\end{aligned}$$

Under H_0, all observations are IID from the same continuous distribution. Therefore, the rank of the observations and the permutation of rank are both uniformly distributed. Since T only depends only on these ranks and not on the actual data values or underlying distribution, except the CIID assumption, its distribution under H_0 does not depend on any parameters of the underlying distribution. Hence, T is distribution-free under H_0.

Question 3.3

Under H_1, at least one treatment group tends to have larger observations compared to other groups. The rank of that group would be distributed towards the larger values, i.e. larger $\bar{R}_{.j}$ in that group. Thus, it would lead to differences in the distribution of ranks among all the groups.

F-statistic increases when between-group variance is larger relative to within-group variance, similarly, T-statistic also increases when the within-group rank means differ more from the between-group rank mean. Therefore, under H_1, the observed differences in location cause $\bar{R}_{.j}$ to differ from $\bar{R}_{..}$, thus making T become large.

Question 3.4

First, we let $S(u) = u$, thus

$$\begin{aligned}
s(i) &= S\left(\frac{i}{N+1}\right) = \frac{i}{N+1} \\
\mu_s &= \frac{1}{N}\sum_{i=1}^{N} s(i) = \frac{1}{N}\sum_{i=1}^{N}\frac{i}{N+1} = \frac{1}{2} \\
\sigma_s^2 &= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{i}{N+1} - \frac{1}{2}\right)^2 = \frac{1}{N}\cdot\frac{N(N-1)}{12(N+1)} \overset{N\to\infty}{=} \frac{1}{12}
\end{aligned}$$

Then, we consider

$$\begin{aligned}
Q &= \frac{1}{\sigma_s^2}\sum_{j=1}^{k}\frac{1}{n_j}\left[\sum_{i=1}^{n_j}\{s(R_{ij}) - \mu_s\}\right]^2 \\
&= \frac{1}{\sigma_s^2}\sum_{j=1}^{k}\frac{1}{n_j}\left[\sum_{i=1}^{n_j}\left(\frac{R_{ij}}{N+1} - \frac{1}{2}\right)\right]^2 \\
&= \frac{1}{\sigma_s^2}\sum_{j=1}^{k}\frac{1}{n_j}\cdot\left[n_j\cdot\left(\frac{\bar{R}_{\cdot j}}{N+1} - \frac{1}{2}\right)\right]^2 \\
&= \frac{1}{\sigma_s^2}\sum_{j=1}^{k} n_j\cdot\left(\frac{\bar{R}_{\cdot j}}{N+1} - \frac{1}{2}\right)^2 \\
&= \frac{1}{\sigma_s^2\cdot(N+1)^2}\sum_{j=1}^{k} n_j\left(\bar{R}_{\cdot j} - \frac{N+1}{2}\right)^2 \\
&= \frac{1}{\sigma_s^2\cdot(N+1)^2}\left[\sum_{j=1}^{k} n_j\cdot\bar{R}_{\cdot j}^2 - \frac{N(N+1)^2}{4}\right] \\
&= \frac{1}{\sigma_s^2\cdot(N+1)^2}\left[\frac{T+3(N+1)}{12}\cdot N(N+1) - \frac{N(N+1)^2}{4}\right] \\
&= \frac{N}{12\sigma_s^2\cdot(N+1)}T \\
&\overset{N\to\infty}{\approx} \frac{N}{12(1/2)N+1}T \\
&= \frac{N}{N+1}T \\
T &\overset{N\to\infty}{\approx} \frac{N+1}{N}Q \\
&\overset{N\to\infty}{\approx} Q \\
\Rightarrow T &\xrightarrow{d} \chi^2_{(k-1)}
\end{aligned}$$

Therefore, under H_0 and when $\min(n_1, \ldots, n_k) \to \infty$, the limiting distribution $T \xrightarrow{d} \chi^2_{(k-1)}$.

Question 3.5

Permutation Procedure

We perform k-sample group permutation under H_0. We first combine k samples of sizes $n_1, n_2, \ldots, n_k$ from the null distribution into a single pooled dataset of size $N = n_1 + n_2 + \ldots + n_k$. Then, we randomly reassign the N samples into k groups with sizes of $n_1, n_2, \ldots, n_k$. It will give new treatment labels to the original pooled data.

For the permutated data, we will then compute the rank of them and thus the Kruskal–Wallis statistic T. The above process will be repeated for a large number. Thus, we can obtain the permutation distribution of T under H_0.

With the originally observed Kruskal–Wallis statistic T_{obs}, we can compute the p-value, which is the proportion of permuted T-values $\geq T_{obs}$. If the p-value $\leq$ the significance level α, we would reject H_0.

Validity

The permutation process for this problem is valid since

- Under H_0, all observations are IID from the same continuous distribution. Therefore, the rank of the observations and the permutation of rank are both uniformly distributed.
- The groups sizes remain unchanged.
- Kruskal–Wallis test statistic T is distribution-free.

Maximum possible number of permutation

The required value is the number of unique ways to assign N distinct observations into k different labeled groups with fixed sizes of $n_1, n_2, \ldots, n_k$. Thus, the maximum possible number of permutation is

$$\frac{N!}{n_1! n_2! \cdots n_k!}$$

Question 3.6

We compute the three p-values using the self-written R function `kw.test(x, group, method=method, B=1000)`, and setting the seed $= 3005$.

Before running R function, there is a remark: The exact test is not really "exact", due to usually large number of the total sample size N. (In this case, $N = 292$, where the numerator of the maximum possible number of permutations is $292! \approx \infty$). So it is difficult to perform that large number of computations. A simulation is a solution for the exact test. Please see the R code for detailed implementation.

```r
data = read.csv("Q3.csv")
set.seed(3005)
kw.test(data$x, data$group, method = "exact", B=100000)$p.value
# Output = 0.04994
kw.test(data$x, data$group, method="asymptotic")$p.value
# Output = 0.05107126
kw.test(data$x, data$group, method = "permutation", B=100000)$p.value
# Output = 0.05011
```

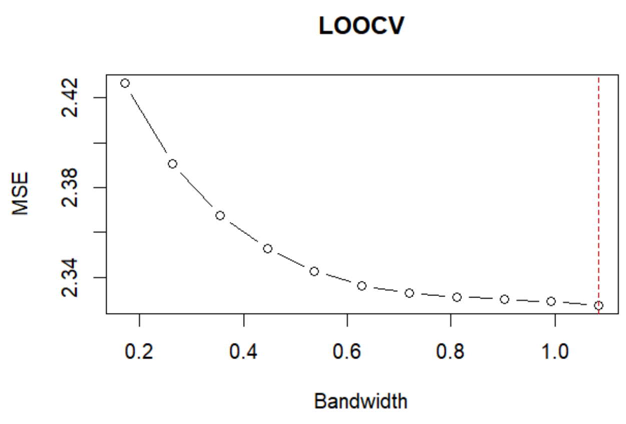
Therefore, the required p-values are

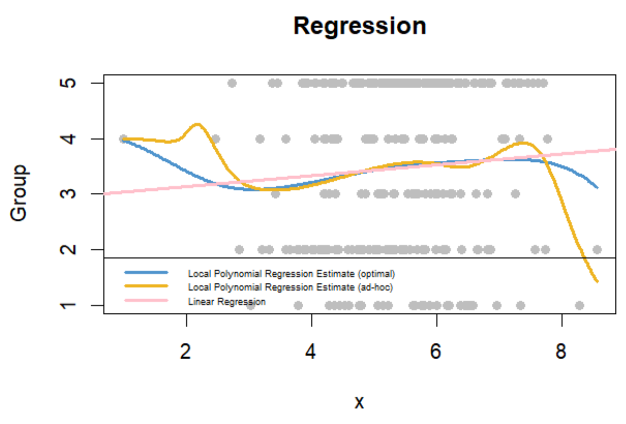
- Exact p-value: 0.04994
- Asymptotic p-value: 0.05107126
- Permutation p-value: 0.05011

Question 4: Q3

Based on the data in Q3, we conduct a regression analysis to compare linear regression with two local polynomial regression estimates. The goal is to demonstrate how selecting an appropriate smoothing bandwidth can influence the shape and interpretability of the fitted regression curve, and to contrast these results with traditional linear regression.

First, we choose the optimal bandwidth by applying Leave-One-Out Cross Validation (LOOCV), using the following R code.





The optimal bandwidth can be found by the function output and also from the plot, which is 1.084344. Hence, we use the optimal bandwidth and another ad-hoc bandwidth choice: 0.05 to plot the corresponding regression, with linear regression.

From the plot, we can see that the local polynomial regression estimate with optimal bandwidth captures nonlinear variations in the data smoothly, without fluctuated lines. Moreover, it shows that the data has non-monotonic patterns, where the curve flexibly bends to reflect these local changes.

Besides, the curve of the estimate with ad-hoc bandwidth appear to be more fluctuated and follow too close with some individual data points, especially the boundary points. It leads to overfitting, as it captures noise rather than the underlying signal.

Lastly, the linear regression line is a simple approximation that does not adjust to local patterns in the data and therefore overestimate the relationship between x and Group. It is easy to conclude that it may not be a good fit, as the true relationship is very likely to be nonlinear.

To conclude, a local polynomial regression with a suitable bandwidth selection would help us identify the relationship between the data, thus assisting us to forecasting outcomes and detecting non-linearity with the data.

Question 1.1: Appendix (R code)

```r
# Data and Simulation parameters
data = read.csv("Q1.csv")
Nrep = 10000
alpha = 0.05
n = 30

# Question 1.1 Rank Sum Test
# (i) Size under H_0
set.seed(3005)
reject_count_H0 = 0
for(i in 1:Nrep){
  x = rnorm(n, mean=1, sd=1)
  y = rnorm(n, mean=1, sd=1)
  test_res = rankSum.test(x, y, alternative="two.sided")
  if(test_res$p.value < alpha) {
    reject_count_H0 = reject_count_H0 + 1
  }
}
size_estimate = reject_count_H0 / Nrep
cat("Estimated size under H0:", size_estimate, "\n")
# Estimated size under H0: 0.0498

# (ii) Power
theta_x_values = c(1.0, 1.5, 2, 2.5)
power_estimates = numeric(length(theta_x_values))
set.seed(3005)
for(j in seq_along(theta_x_values)){
  theta_x = theta_x_values[j]
  reject_count_H1 = 0
  for(i in 1:Nrep){
    x = rnorm(n, mean=theta_x, sd=theta_x)
    y = rnorm(n, mean=1, sd=1)
    test_res = rankSum.test(x, y, alternative="two.sided")
    if(test_res$p.value < alpha){
      reject_count_H1 = reject_count_H1 + 1
    }
  }
  power_estimates[j] = reject_count_H1 / Nrep
}
list(theta = theta_x_values, Power = power_estimates)
# $theta
# [1] 1.0 1.5 2.0 2.5
#
# $Power
# [1] 0.0498 0.3064 0.6425 0.8054

# (iii) Performing Test
rankSum.test(data$x1, data$x2, alternative = "two.sided")$p.value
# Output = 0.01173492
```

Question 1.2: Appendix (R code)

```r
# Question 1.2 Ansari-Bradley Test
# (i) Size under H_0
set.seed(3005)
reject_count_H0 = 0
for(i in 1:Nrep){
  x = rnorm(n, mean=1, sd=1)
  y = rnorm(n, mean=1, sd=1)
  test_res = ansari.test(x, y, alternative="two.sided")
  if(test_res$p.value < alpha) {
    reject_count_H0 = reject_count_H0 + 1
  }
}
size_estimate = reject_count_H0 / Nrep
cat("Estimated size under H0:", size_estimate, "\n")
# Estimated size under H0: 0.0488

# (ii) Power
theta_x_values = c(1.0, 1.5, 2, 2.5)
power_estimates = numeric(length(theta_x_values))
set.seed(3005)
for(j in seq_along(theta_x_values)){
  theta_x = theta_x_values[j]
  reject_count_H1 = 0
  for(i in 1:Nrep){
    x = rnorm(n, mean=theta_x, sd=theta_x)
    y = rnorm(n, mean=1, sd=1)
    test_res = ansari.test(x, y, alternative="two.sided")
    if(test_res$p.value < alpha){
      reject_count_H1 = reject_count_H1 + 1
    }
  }
  power_estimates[j] = reject_count_H1 / Nrep
}
list(theta = theta_x_values, Power = power_estimates)
# $theta
# [1] 1.0 1.5 2.0 2.5
#
# $Power
# [1] 0.0488 0.3367 0.6839 0.8367

# (iii) Performing Test
ansari.test(data$x1, data$x2, alternative = "two.sided")$p.value
# Output = 0.1401889
```

Question 1.3: Appendix (R code)

```r
# Question 1.3 Calvin's Test
# (i) Size under H_0
set.seed(3005)
reject_count_H0 = 0
for (i in 1:Nrep) {
  x = rnorm(n, mean=1, sd=1)
  y = rnorm(n, mean=1, sd=1)

  rs_test = rankSum.test(x, y, alternative = "two.sided")
  ab_test = ansari.test(x, y, alternative = "two.sided")

  # Combined test rejects if either rejects
  if (rs_test$p.value < alpha  ab_test$p.value < alpha) {
    reject_count_H0 = reject_count_H0 + 1
  }
}

size_estimate = reject_count_H0 / Nrep
cat("Estimated size under H0:", size_estimate, "\n")
# Estimated size under H0: 0.0965

# (ii) Power
theta_x_values = c(1.0, 1.5, 2, 2.5)
power_estimates = numeric(length(theta_x_values))
set.seed(3005)
for(j in seq_along(theta_x_values)){
  theta_x = theta_x_values[j]
  reject_count_H1 = 0
  for(i in 1:Nrep){
    x = rnorm(n, mean=theta_x, sd=theta_x)
    y = rnorm(n, mean=1, sd=1)
    rs_test = rankSum.test(x, y, alternative = "two.sided")
    ab_test = ansari.test(x, y, alternative = "two.sided")
    if (rs_test$p.value < alpha  ab_test$p.value < alpha) {
      reject_count_H1 = reject_count_H1 + 1
    }
  }
  power_estimates[j] = reject_count_H1 / Nrep
}
list(theta = theta_x_values, Power = power_estimates)

# $theta
# [1] 1.0 1.5 2.0 2.5
#
# $Power
# [1] 0.0965 0.5575 0.9266 0.9901

# (iii) Performing Test
min(rankSum.test(data$x1, data$x2, alternative = "two.sided")$p.value,
    ansari.test(data$x1, data$x2, alternative = "two.sided")$p.value)
# Output = 0.01173492
```

Question 1.4: Appendix (R code)

```r
# Question 1.4 Cramer-von Mises Test
# Test Function
cvm.test0 = function(x, F0, exact=TRUE){
  n = length(x)
  u = F0(sort(x))
  i = 1:n
  C = 1/12/n + sum((u-(2*i-1)/2/n)^2)
  if(exact){
    nRep = 10000
    out = rep(NA,nRep)
    for(iRep in 1:nRep) out[iRep] = cvm.test0(runif(n), F0=punif, exact=FALSE)$
        statistic
    p = mean(out>C)
  }else{
    p = NA
  }
  list(statistic=C, p.value=p)
}

# (i) Size under H_0
set.seed(3005)
Nrep = 500 # Lower the number of rep due to bad performance of the function
reject_count_H0 = 0
for(i in 1:Nrep){
  x = rnorm(n, mean=1, sd=1)
  y = rnorm(n, mean=1, sd=1)
  test_res = cvm.test0(x, ecdf(y))
  if(test_res$p.value < alpha) {
    reject_count_H0 = reject_count_H0 + 1
  }
  if(i %% 10 == 0) {
    cat("Simulation process:", i, "\n")
  }
}
size_estimate = reject_count_H0 / Nrep
cat("Estimated size under H0:", size_estimate, "\n")
# Estimated size under H0: 0.234

# (ii) Power
Nrep = 100 # Lower the number of rep due to bad performance of the function
theta_x_values = c(1.0, 1.5, 2, 2.5)
power_estimates = numeric(length(theta_x_values))
set.seed(3005)
for(j in seq_along(theta_x_values)){
  theta_x = theta_x_values[j]
  reject_count_H1 = 0
  for(i in 1:Nrep){
    x = rnorm(n, mean=theta_x, sd=theta_x)
    y = rnorm(n, mean=1, sd=1)
    test_res = cvm.test0(x, ecdf(y))
    if(test_res$p.value < alpha){
      reject_count_H1 = reject_count_H1 + 1
    }
    if(i %% 10 == 0) {
      cat("Simulation process:", i, "\n")
    }
  }
  power_estimates[j] = reject_count_H1 / Nrep
}
list(theta = theta_x_values, Power = power_estimates)

# $theta
# [1] 1.0 1.5 2.0 2.5
#
# $Power
# [1] 0.24 0.64 0.97 1.00

# (iii) Performing Test
set.seed(3005)
cvm.test0(data$x1, ecdf(data$x2))$p.value
# Output: 3e-04
```

Question 1.5: Appendix (R code)

```r
# Question 1.5 ks.test() function
# (i) Size under H_0
set.seed(3005)
reject_count_H0 = 0
for(i in 1:Nrep){
  x = rnorm(n, mean=1, sd=1)
  y = rnorm(n, mean=1, sd=1)
  test_res = ks.test(x, ecdf(y))
  if(test_res$p.value < alpha) {
    reject_count_H0 = reject_count_H0 + 1
  }
  if(i %% 1000 == 0) {
    cat("Simulation process:", i, "\n")
  }
}
size_estimate = reject_count_H0 / Nrep
cat("Estimated size under H0:", size_estimate, "\n")
# Estimated size under H0: 0.2437

# (ii) Power
theta_x_values = c(1.0, 1.5, 2, 2.5)
power_estimates = numeric(length(theta_x_values))
set.seed(3005)
for(j in seq_along(theta_x_values)){
  theta_x = theta_x_values[j]
  reject_count_H1 = 0
  for(i in 1:Nrep){
    x = rnorm(n, mean=theta_x, sd=theta_x)
    y = rnorm(n, mean=1, sd=1)
    test_res = ks.test(x, ecdf(y))
    if(test_res$p.value < alpha){
      reject_count_H1 = reject_count_H1 + 1
    }
    if(i %% 1000 == 0) {
      cat("Simulation process:", i, "\n")
    }
  }
  power_estimates[j] = reject_count_H1 / Nrep
}
list(theta = theta_x_values, Power = power_estimates)

# $theta
# [1] 1.0 1.5 2.0 2.5
#
# $Power
# [1] 0.2437 0.7003 0.9635 0.9959

# (iii) Performing Test
ks.test(data$x1, ecdf(data$x2))$p.value
# Output: 0.0001908552
```

Question 1.6: Appendix (R code)

```r
# Question 1.6 2-sample Permutation test
# Test function
group.ptest = function(x, y, FUN=NULL, B=1e4, alternative=c("two.sided","less",
    "greater"), plot=FALSE){
  # FUN         = function of test statistic
  # mu0         = null value of mu
  # B           = number of Monte Carlo replications
  # alternative = direction of alternative hypothesis (e.g., "less" means H1:
      mu(x)<mu(y))
  # plot        = logical indicating whether to plot permutation distribution
  alternative = match.arg(alternative)
  if(is.null(FUN)) FUN = function(x,y) sum(rank(c(x,y))[1:length(x)])
  n1 = length(x)
  n2 = length(y)
  xy = c(x,y)
  n = n1+n2
  Tobs = FUN(x,y)
  Tb = rep(NA,B)
  for(b in 1:B){
    I = sample(1:n,n1,replace=FALSE)
    xb = xy[I]
    yb = xy[-I]
    Tb[b] = FUN(xb,yb)
  }
  if(plot){
    hist(Tb,nclass=30,freq=FALSE,xlim=range(c(Tb,Tobs)),xlab=expression(italic(
        T)),
         main="Permutation distribution",
         col="pink",border=FALSE,cex.lab=1.5, cex.axis=1.5, cex.main=1.5)
    abline(v=Tobs,col="blue4",lty=2,lwd=2)
  }
  switch(alternative, "two.sided"=min(mean(Tb<=Tobs),mean(Tb>=Tobs))*2,
         "less"=mean(Tb<=Tobs),
         "greater"=mean(Tb>=Tobs))
}
# (i) Size under H_0
set.seed(3005)
Nrep = 500 # Lower the number of rep due to bad performance of the function
FUN_tTest = function(x,y) mean(x) - mean(y) # Permutation t-test use mean diff
    as test statistic
reject_count_H0 = 0
for(i in 1:Nrep){
  x = rnorm(n, mean=1, sd=1)
  y = rnorm(n, mean=1, sd=1)
  test_res = group.ptest(x, y, FUN=FUN_tTest)
  if(test_res < alpha) {
    reject_count_H0 = reject_count_H0 + 1
  }
  if(i %% 100 == 0) {
    cat("Simulation process:", i, "\n")
  }
}
size_estimate = reject_count_H0 / Nrep
cat("Estimated size under H0:", size_estimate, "\n")
# Estimated size under H0: 0.058
# (ii) Power
theta_x_values = c(1.0, 1.5, 2, 2.5)
power_estimates = numeric(length(theta_x_values))
FUN_tTest = function(x,y) mean(x) - mean(y) # Permutation t-test use mean diff
    as test statistic
set.seed(3005)
for(j in seq_along(theta_x_values)){
  theta_x = theta_x_values[j]
  reject_count_H1 = 0
  for(i in 1:Nrep){
    x = rnorm(n, mean=theta_x, sd=theta_x)
    y = rnorm(n, mean=1, sd=1)
    test_res = group.ptest(x, y, FUN=FUN_tTest)
    if(test_res < alpha){
      reject_count_H1 = reject_count_H1 + 1
    }
    if(i %% 100 == 0) {
      cat("Simulation process:", i, "\n")
    }
  }
  power_estimates[j] = reject_count_H1 / Nrep
}
list(theta = theta_x_values, Power = power_estimates)

# $theta
# [1] 1.0 1.5 2.0 2.5
#
# $Power
# [1] 0.058 0.332 0.648 0.820

# (iii) Performing Test
set.seed(3005)
group.ptest(data$x1, data$x2,FUN=FUN_tTest)
# Output: 0.0096
```

Question 1.7: Appendix (R code)

```r
# Question 1.7 Paired sign-rank test
# (i) Size under H_0
set.seed(3005)
reject_count_H0 = 0
for(i in 1:Nrep){
  x = rnorm(n, mean=1, sd=1)
  y = rnorm(n, mean=1, sd=1)
  test_res = wilcox.test(x, y, mu=0, paired=TRUE, exact=TRUE, alternative="two.
     sided")$p.value
  if(test_res < alpha) {
    reject_count_H0 = reject_count_H0 + 1
  }
  if(i %% 1000 == 0) {
    cat("Simulation process:", i, "\n")
  }
}
size_estimate = reject_count_H0 / Nrep
cat("Estimated size under H0:", size_estimate, "\n")
# Estimated size under H0: 0.0529

# (ii) Power
theta_x_values = c(1.0, 1.5, 2, 2.5)
power_estimates = numeric(length(theta_x_values))
set.seed(3005)
for(j in seq_along(theta_x_values)){
  theta_x = theta_x_values[j]
  reject_count_H1 = 0
  for(i in 1:Nrep){
    x = rnorm(n, mean=theta_x, sd=theta_x)
    y = rnorm(n, mean=1, sd=1)
    test_res = wilcox.test(x, y, mu=0, paired=TRUE, exact=TRUE, alternative="
       two.sided")$p.value
    if(test_res < alpha){
      reject_count_H1 = reject_count_H1 + 1
    }
    if(i %% 1000 == 0) {
      cat("Simulation process:", i, "\n")
    }
  }
  power_estimates[j] = reject_count_H1 / Nrep
}
list(theta = theta_x_values, Power = power_estimates)

# $theta
# [1] 1.0 1.5 2.0 2.5
#
# $Power
# [1] 0.0529 0.3031 0.6412 0.8169


# (iii) Performing Test
wilcox.test(data$x1, data$x2, mu=0, paired=TRUE, exact=TRUE, alternative="two.
    sided")$p.value
# Output = 0.004101777
```

Question 1.8: Appendix (R Code)

```r
# Question 1.8 Bootstrap CI
# Test function
ci.bcaboot_2sample = function(x, y, FUN, alpha0=.05, B=10000){
  n1 = length(x)
  n2 = length(y)
  est = FUN(x,y)

  # Bootstrap replicates
  boot = numeric(B)
  for(b in 1:B){
    x_star = sample(x, n1, replace=TRUE)
    y_star = sample(y, n2, replace=TRUE)
    boot[b] = FUN(x_star, y_star)
  }

  jack_x = numeric(n1)
  for(i in 1:n1) jack_x[i] = FUN(x[-i], y)

  jack_y = numeric(n2)
  for(j in 1:n2) jack_y[j] = FUN(x, y[-j])

  jack = c(jack_x, jack_y)
  jackBar = mean(jack)

  A = sum((jack - jackBar)^3)/(6 * (sum((jack - jackBar)^2)^(3/2)))
  Z = qnorm(mean(boot < est))
  alpha1 = pnorm(Z + (Z + qnorm(alpha0/2)) / (1 - A*(Z + qnorm(alpha0/2))))
  alpha2 = pnorm(Z + (Z + qnorm(1 - alpha0/2)) / (1 - A*(Z + qnorm(1 - alpha0/2))))

  CI = quantile(boot, c(alpha1, alpha2))
  names(CI) = c("lower","upper")
  CI
}
FUN_diffvar = function(x,y) var(x)-var(y)

# (i) Size under H_0
Nrep = 500 # Lower the number of rep due to bad performance of the function
set.seed(3005)
reject_count_H0 = 0
for(i in 1:Nrep){
  x = rnorm(n, mean=1, sd=1)
  y = rnorm(n, mean=1, sd=1)
  CI = ci.bcaboot_2sample(x,y,FUN=FUN_diffvar,alpha0=alpha,B=2000) # B=2000 for speed
  if(CI[1]>0  CI[2]<0) reject_count_H0 = reject_count_H0 + 1
  if(i %% 100 == 0) {
    cat("Simulation process:", i, "\n")
  }
}

size_estimate = reject_count_H0 / Nrep
cat("Estimated size under H0:", size_estimate, "\n")
# Estimated size under H0: 0.072

# (ii) Power
set.seed(3005)
Nrep = 500 # Lower the number of rep due to bad performance of the function
theta_x_values = c(1.0, 1.5, 2, 2.5)
power_estimates = numeric(length(theta_x_values))

for(j in seq_along(theta_x_values)){
  theta_x = theta_x_values[j]
  reject_count_H1 = 0
  for(i in 1:Nrep){
    x = rnorm(n, mean=theta_x, sd=theta_x)
    y = rnorm(n, mean=1, sd=1)

    CI = ci.bcaboot_2sample(x,y,FUN=FUN_diffvar,alpha0=alpha,B=2000)
    if(CI[1]>0  CI[2]<0) reject_count_H1 = reject_count_H1 + 1
    if(i %% 100 == 0) {
      cat("Simulation process:", i, "\n")
    }
  }
  power_estimates[j] = reject_count_H1 / Nrep
}
list(theta = theta_x_values, Power = power_estimates)

# $theta
# [1] 1.0 1.5 2.0 2.5
#
# $Power
# [1] 0.072 0.590 0.970 1.000

# (iii) Performing Test
set.seed(3005)
(CI = ci.bcaboot_2sample(data$x1,data$x2,FUN=FUN_diffvar,alpha0=alpha,B=2000))
#      lower      upper
# -5541.1703   717.2781
```

Question 3.6: Appendix (R Code)

```r
kw.test = function(x, g,
                    method=c("asymptotic", "exact", "permutation"), B=1000) {
  method = match.arg(method)

  if (!is.factor(g)) {
    g = factor(g)
  }

  N = length(x)
  group_sizes = table(g)
  k = length(group_sizes)
  ranks = rank(x)

  # Compute T_obs
  Rbar_dotdot = (N+1)/2
  Rbar_j = tapply(ranks, g, mean)
  n_j = as.numeric(group_sizes)
  T_obs = (12/(N*(N+1)))*sum(n_j*(Rbar_j-Rbar_dotdot)^2)

  # T-stat Function
  kw_stat = function(r, grp, N) {
    Rbar_dotdot = (N+1)/2
    Rbar_j = tapply(r, grp, mean)
    n_j = table(grp)
    (12/(N*(N+1)))*sum(n_j*(Rbar_j-Rbar_dotdot)^2)
  }

  if (method == "asymptotic") {
    # Built-in kruskal.test() function is used for asymptotic p-value
    kt = kruskal.test(x, g)
    return(list(
      statistic = T_obs,
      p.value = kt$p.value,
      method = "Asymptotic Kruskal-Wallis Test"
    ))
  }

  # Permutation Test Function
  perform_permutation_test = function(ranks, group_sizes, k, B) {
    N = length(ranks)
    T_values = numeric(B)
    for (b in seq_len(B)) {
      permutated_ranks = sample(ranks, size=N, replace=FALSE)
      permutated_groups = factor(rep(seq_len(k), times=group_sizes), levels=seq
          _len(k))
      T_values[b] = kw_stat(permutated_ranks, permutated_groups, N)
    }
    return(T_values)
  }

  if (method == "exact") {
    T_values = perform_permutation_test(ranks, n_j, k, B)
    p_val = mean(T_values >= T_obs)
    return(list(
      statistic = T_obs,
      p.value = p_val,
      method = "Exact Kruskal-Wallis Test"
    ))
  }

  if (method == "permutation") {
    # Perform Monte Carlo permutation test
    T_values = perform_permutation_test(ranks, n_j, k, B)
    p_val = mean(T_values >= T_obs)
    return(list(
      statistic = T_obs,
      p.value = p_val,
      method = "Permutation Kruskal-Wallis Test"
    ))
  }
}
```

Question 4: Appendix (R Code)

The following R code is used for computing the optimal bandwidth and generating the regression plot for Question 4.

```r
data = read.csv("Q3.csv")

loocv = function(x,y,p=1,K=NULL,B=11,from=NULL,to=NULL,plot=TRUE,...){
  # preliminary definitions
  #------------------------------------------------------------
  if(is.null(K)) K = dnorm
  n = length(x)
  X = array(1,dim=c(n,p+1))
  out = rep(NA,n)
  R = diff(range(x))
  if(is.null(from)) from=R/n^(4/5)
  if(is.null(to)) to=R/10
  bw.all = seq(from=from,to=to,length=B)
  MSE = rep(NA,length(bw.all))
  L = rep(NA,n)
  # LOOCV
  #------------------------------------------------------------
  for(i.bw in 1:length(bw.all)){
    cat(i.bw," >> ")
    bw = bw.all[i.bw]
    for(i in 1:n){
      at = x[i]
      if(p>0){
        for(j in 1:p){
          X[,j+1] = (x-at)^j
        }
      }
      W = diag(K((x-at)/bw))
      XW = t(X)%*%W
      temp = solve(XW%*%X,XW)[1,]
      out[i] = sum(temp*y)
      L[i] = temp[i]
    }
    MSE[i.bw] = mean(((y-out)/(1-mean(L)))^2)
  }
  bw.opt = bw.all[which.min(MSE)]
  # plots
  #------------------------------------------------------------
  if(plot){
    plot(bw.all, MSE, type="b", ylab="MSE", xlab="Bandwidth",...)
    abline(v=bw.opt, col="red3", lty=2)
  }
  # return optimal BW
  #------------------------------------------------------------
  list(bw=bw.all, MSE=MSE, bw.opt=bw.opt)
}
lreg = function(x,y,p=1,K=NULL,at=NULL,bw=NULL,plot=TRUE,...){
  # preliminary definitions
  #------------------------------------------------------------
  if(is.null(K)) K = dnorm
  n = length(x)
  X = array(1,dim=c(n,p+1))
  if(is.null(at)){
    at = seq(min(x),to=max(x),length=301)
  }else{
    at = sort(at)
  }
  if(is.null(bw)) bw = (max(x)-min(x))/sqrt(n)
  n.at = length(at)
  at.all = at
  out = rep(NA,n.at)
  # compute the estimate
  #-------------------------------------------------------------
  for(i.at in 1:n.at){
    at = at.all[i.at]
    if(p>0){
      for(j in 1:p){
        X[,j+1] = (x-at)^j/factorial(j)
      }
    }
    # Method 1: more transparent
    W = diag(K((x-at)/bw))
    XW = t(X)%*%W
    out[i.at] = sum(solve(XW%*%X,XW)[1,]*y)
    # Method 2: faster and easier
    # out[i.at] = lm(y~X-1, weights=K((x-at)/bw))$coef[1]
  }
  # plot the data and regression line
  #-------------------------------------------------------------
  if(plot){
    optional = list(...)
    if(is.null(optional$type)) type="l"
    if(is.null(optional$col)) col="red4"
    if(is.null(optional$lwd)) lwd=3
    if(is.null(optional$lty)) lty=1
    I = order(at.all)
    plot(x,y, pch=19, col="grey", ...)
    points(at.all[I], out[I], type=type, col=col,lwd=lwd, lty=lty)

  }
  # return estimates
  #-------------------------------------------------------------
  list(at=at.all,fit=out,bw=bw)
}

x = data$x
y = data$group
K = function(t) dnorm(t)
R = diff(range(x))
n = length(x)
at = seq(from=min(x), to=max(x), length=301)

par(mfrow=c(1,2))

out.cv = loocv(x,y,p=1,K=K, from=R/n^(2/3), to=R/7, plot=TRUE, main="LOOCV")
out.cv$bw.opt

plot(x,y,pch=19,col="grey",xlab="x", ylab="Group", main="Regression")
```

```r
lines(at, lreg(data$x, data$group, p=1, K=K, at=at, bw=out.cv$bw.opt, plot=
    FALSE)$fit, col="steelblue3", lwd=3)
lines(at, lreg(data$x, data$group, plot=FALSE)$fit, bw=0.05, col="goldenrod2",
    lwd=3)

linear_reg = lm(y~x)$coef
abline(a=linear_reg[1], b=linear_reg[2], col="pink", lwd=3)
legend("bottomleft",
       c("Local Polynomial Regression Estimate (optimal)",
         "Local Polynomial Regression Estimate (ad-hoc)",
         "Linear Regression"),
       col=c("steelblue3", "goldenrod2", "pink"),
       lty=1, lwd=3, cex=0.45,
       bg="transparent"
       )
```