SS-2864B Take Home Exam

Ali Al-Musawi

19/04/2020

Question 1

(a)

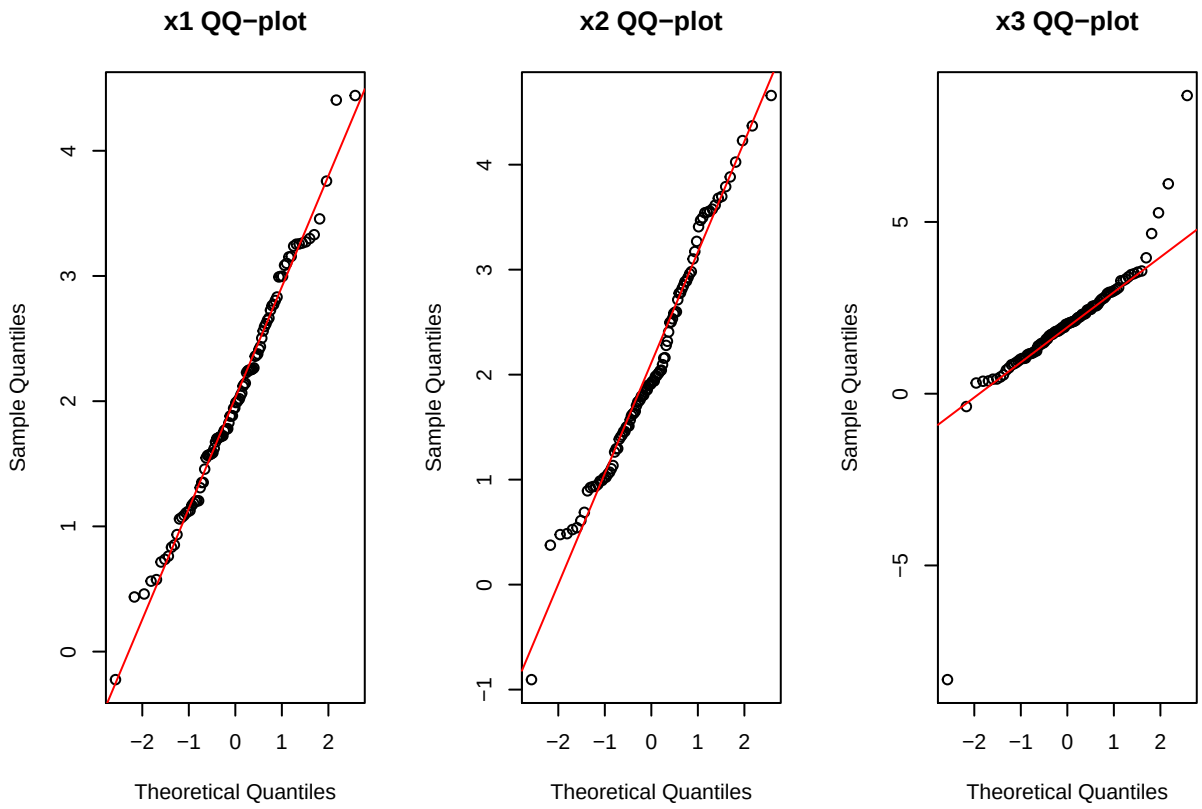
```
# This function returns the JB statistic given a sequence of random variables
# Input:
# x: a numeric vector
# Output:
# The JB statistic of x[1], x[2], ..., x[n]
JB = function(x) {
  n = length(x)
  x.bar = mean(x)
  s = sd(x)
  gamma = (1/(n*s^3))*sum((x - x.bar)^3)
  kappa = (1/(n*s^4))*sum((x - x.bar)^4)
  jb = (n*gamma^2)/6 + (n*(kappa-3)^2)/24
  return(jb)
}
# Test The function
set.seed(0)
JB(rnorm(100))
```

```
## [1] 0.8768345
```

(b)

```
n = 100
set.seed(0)
# Simulate x1
eps = 0;
x1 = rnorm(n, 2, 1+5*rbinom(n, 1, eps))
# Simulate x2
eps = 0.01
x2 = rnorm(n, 2, 1+5*rbinom(n, 1, eps))
# Simulate x3
eps = 0.05
x3 = rnorm(n, 2, 1+5*rbinom(n, 1, eps))
# Test Normality
par(mfrow=c(1,3))
qqnorm(x1, main = "x1 QQ-plot")
qqline(x1, col = 2)
qqnorm(x2, main = "x2 QQ-plot")
qqline(x2, col = 2)
```

```r
qqnorm(x3, main = "x3 QQ-plot")
qqline(x3, col = 2)
```



Based on the result, we conclude x_1, x_2 datasets can be approximated by a normal distribution whereas x_3 cannot. It seems like increasing **eps** even slightly causes the dataset to deviate from normality.

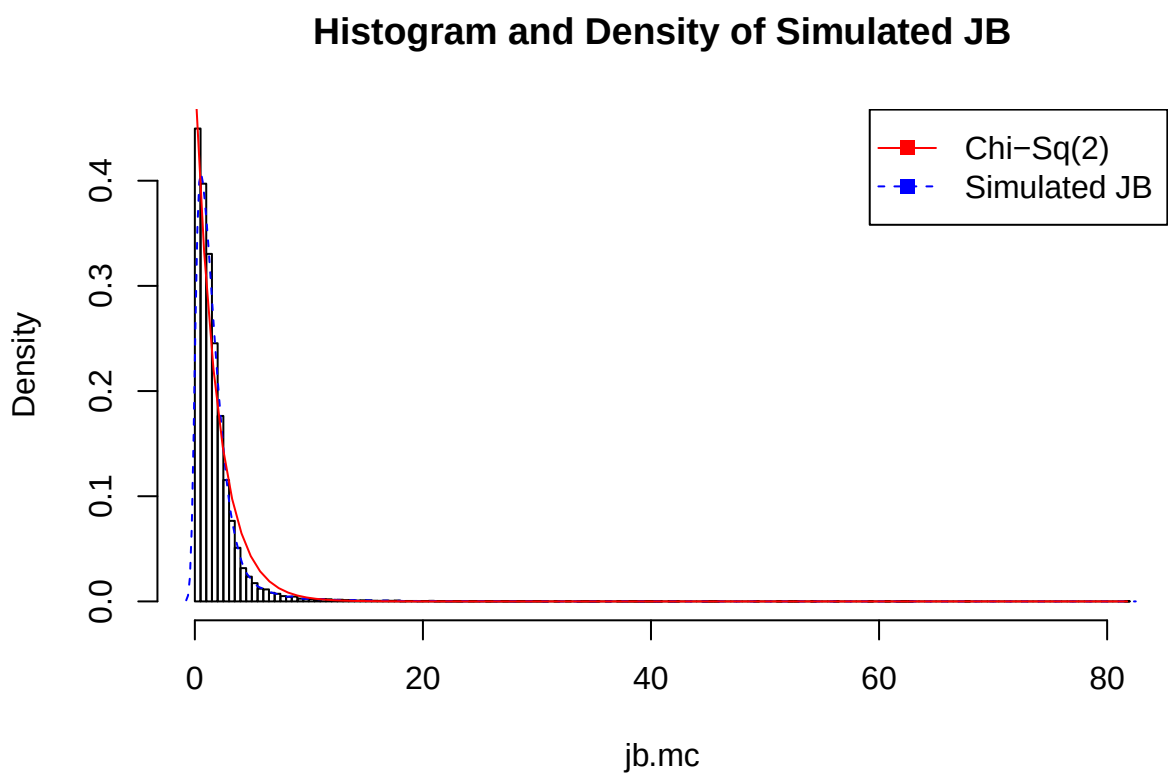
(c)

```r
# This function uses Monte Carlo simulation to compute the JB statistic
# under the assumption of normality.
# Input:
# n: the simulated sample size
# K: the iterations of the simulation
# Output:
# The JB statistic of the K iterations
JB.MC = function(n, K = 50000) {
  X = replicate(K, rnorm(n))
  jb.mc = apply(X = X, MARGIN = 2, FUN = JB)
  return(jb.mc)
}
```

(d)

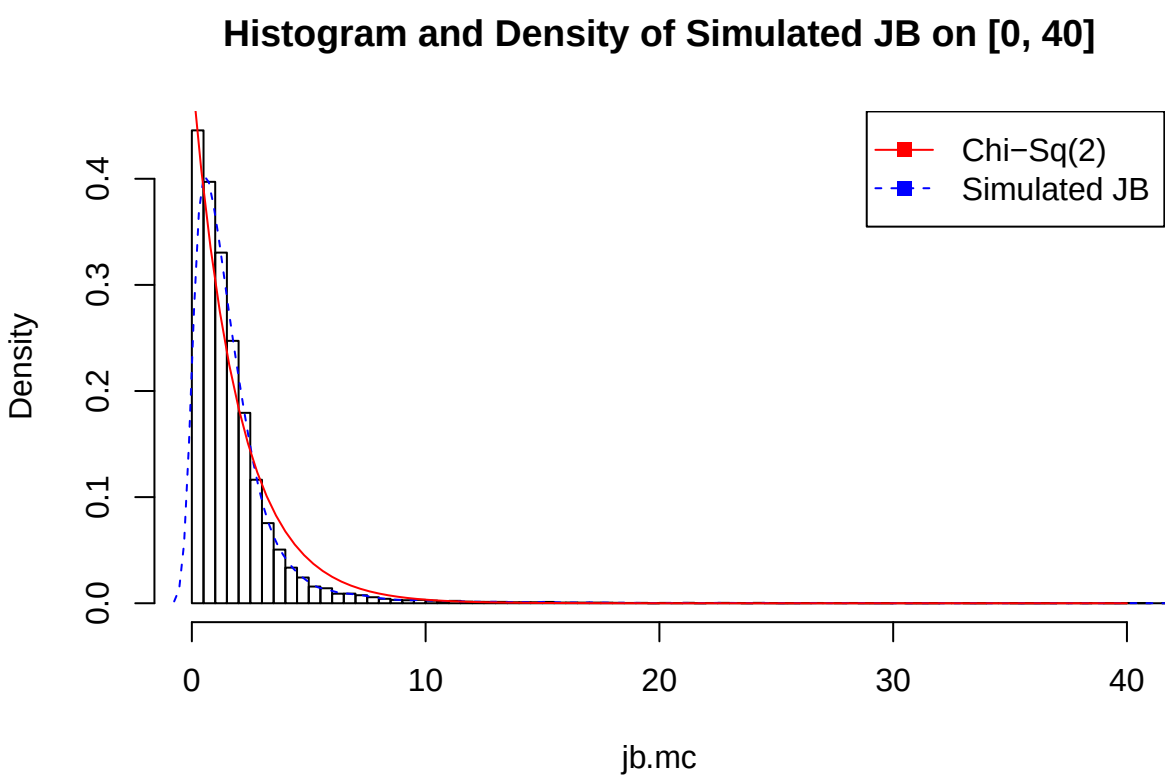
```r
# Use monte carlo to simulate some 50,000 values of the JB statistic
jb.mc = JB.MC(100)
hist(x = jb.mc, breaks = 225, probability = T, main = "Histogram and Density of Simulated JB")
```

```
lines(density(jb.mc, adjust = 2), col = 4, lty = 'dashed')
curve(dchisq(x, df = 2), from = min(jb.mc), to = max(jb.mc), col = 2, add = T)
legend("topright", legend = c("Chi-Sq(2)", "Simulated JB"), col = c(2, 4),
       pch = 15, lty = c('solid', 'dashed'))
```



Note it is hard to see the behavior since most of the hump is concentrated on the left, so we will view the same plot on the sub-interval $x \in [0, 40]$.

```
jb.mc = JB.MC(100)
hist(x = jb.mc, breaks = 225, probability = T,
     main = "Histogram and Density of Simulated JB on [0, 40]",
     xlim = c(0, 40))
lines(density(jb.mc, adjust = 2), col = 4, lty = 'dashed')
curve(dchisq(x, df = 2), from = 0, to = 40, col = 2, add = T)
legend("topright", legend = c("Chi-Sq(2)", "Simulated JB"), col = c(2, 4),
       pch = 15, lty = c('solid', 'dashed'))
```



Based on the generated output, we observe that both densities are skewed to the right and have close maxima. Overall, they trace out the same curve family. On the other hand, it seems like the MC-simulated density decreases faster than the $\chi^2(2)$ density. Asymptotically, the densities converge.

(e)

```
set.seed(0)
# compute the JB stat on x1, x2, and x3:
jb1 = JB(x1)
jb2 = JB(x2)
jb3 = JB(x3)
# compute the p-value for each jb using the cdf of chisq(2)
p1.chisq = pchisq(jb1, 2, lower.tail = F)
p1.chisq
```

```
## [1] 0.6450566
```

```
p2.chisq = pchisq(jb2, 2, lower.tail = F)
p2.chisq
```

```
## [1] 0.5832885
```

```
p3.chisq = pchisq(jb3, 2, lower.tail = F)
p3.chisq
```

```
## [1] 3.897483e-274
```

```
# compute the p-value for each jb using monte carlo
p1.mc = sum((sort(jb.mc) > jb1))/length(jb.mc)
```

```
p1.mc
```

```
## [1] 0.62498
```

```
p2.mc = sum((sort(jb.mc) > jb2))/length(jb.mc)
p2.mc
```

```
## [1] 0.55014
```

```
p3.mc = sum((sort(jb.mc) > jb3))/length(jb.mc)
p3.mc
```

```
## [1] 0
```

Note that the p-value results from the theoretical and simulated distributions are similar, and in fact identical for the dataset x_3. These p-values illustrate that x_1, x_2 come from a normal distribution with significance level $\alpha = 0.01$ while x_3 does not for any significance level. These results match with our QQ-plots.

Question 2

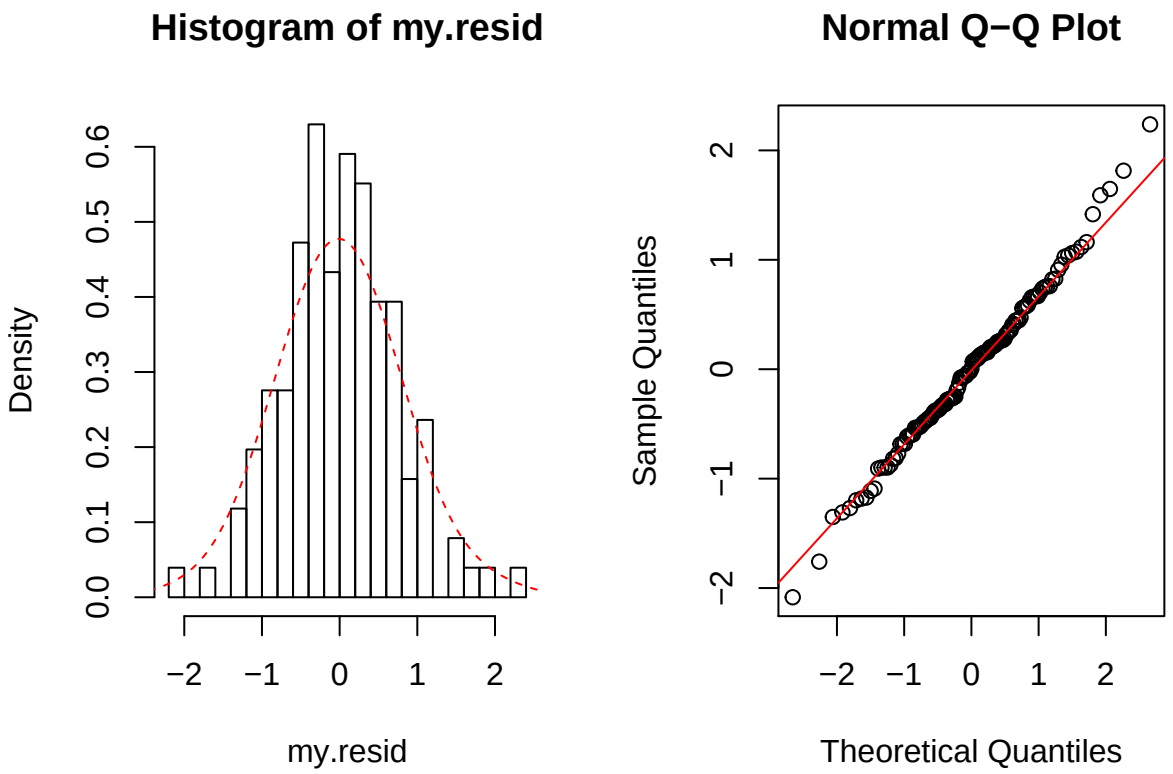
First off, we will import the huron dataset.

(a)

```
# This function estimates the coefficient theta of an AR(1) time series model
# Input:
# x: a numeric vector with mean 0
# Output:
# The least-squares estimator of theta in AR(1)
theta.est = function(x) {
  n = length(x)
  x.lag = c(0, x[-n])
  x.sumsq = sum(x^2)
  theta = sum(x.lag*x)/x.sumsq
  return(theta)
}
# Test the function
my.est = theta.est(huron - mean(huron))
```

(b)

```
x = huron - mean(huron)
n = length(x)
x.lag = c(0, x[-n])
eps.hat = x - my.est*x.lag
my.resid = eps.hat - mean(eps.hat)
par(mfrow = c(1, 2))
hist(my.resid, breaks = 2*ceiling(sqrt(n)), probability = T)
lines(density(my.resid, adjust = 2), col = 2, lty = 'dashed')
qqnorm(my.resid)
qqline(my.resid, col = 2)
```



There is a strong of evidence of normality based on the output generated. That is, the centered residuals are normally distributed.

(c)

```
# The following function defines one boostrap simulation on global variables.
one.boot = function(eps = my.resid, theta = my.est) {
  n = length(eps)
  eps.star = sample(eps, replace = T, size = n)
  x.star = numeric(n)
  x.star[1] = eps.star[1]
  for (i in 2:n) {
    x.star[i] = theta*x.star[i-1] + eps.star[i]
    }
  return (theta.est(x.star))
}

# Test the function to see if the output is different
one.boot()
```

```
## [1] 0.8165481
```

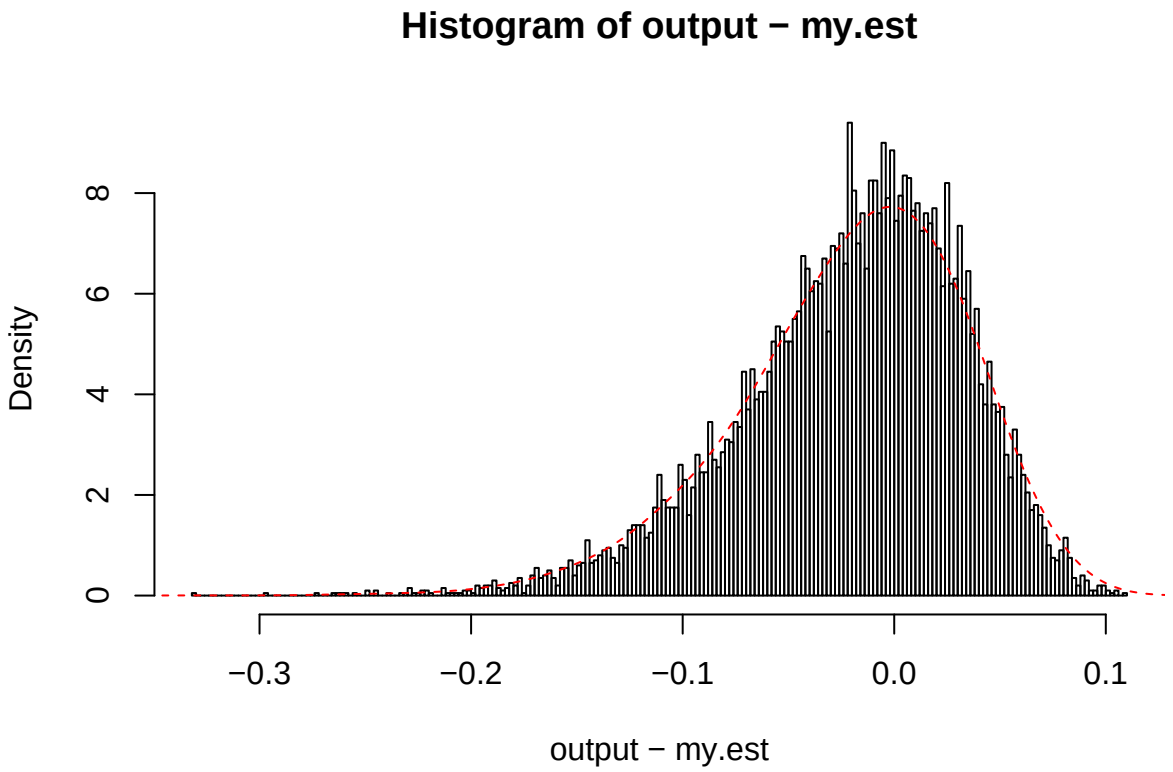
```
one.boot()
```

```
## [1] 0.8828675
```

```
# Run 10000 bootsrap rounds
output = replicate(10000, one.boot())
```

(d)

```
hist(x = output-my.est, breaks = 200, probability = T)
lines(density(output-my.est, adjust = 2), col = 2, lty = 'dashed')
Rmisc::CI(output, ci = 0.95)
```



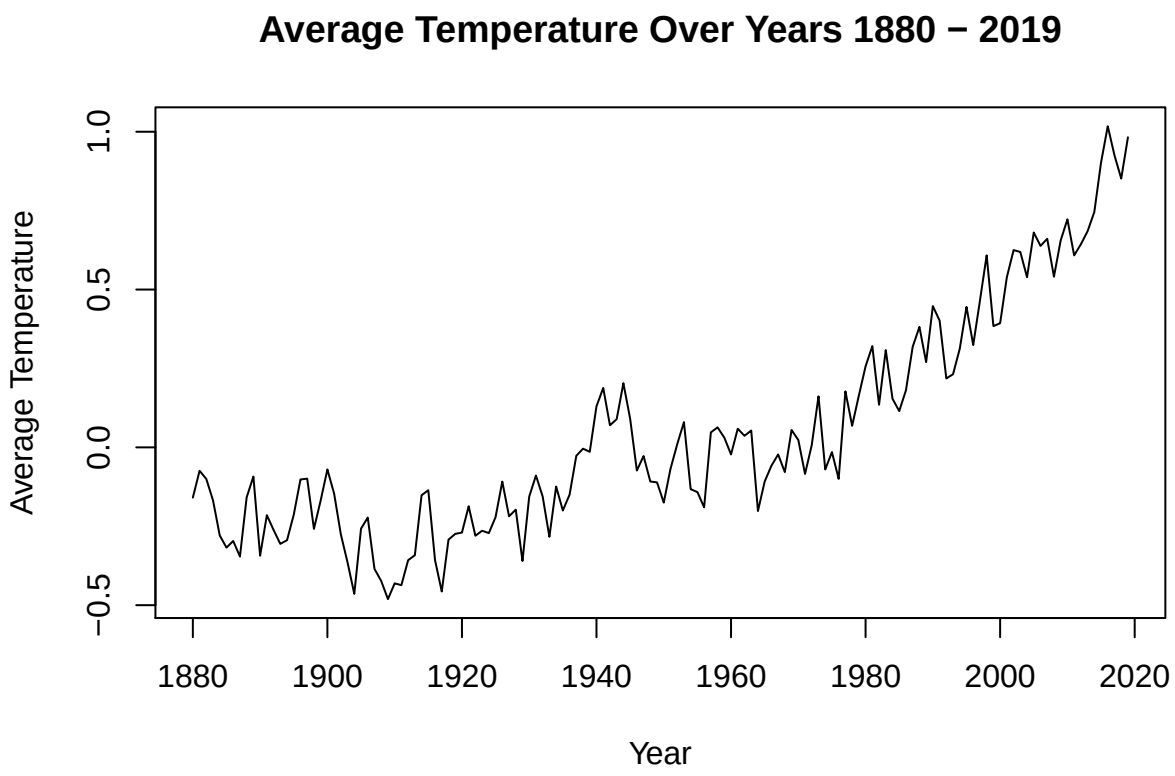

```
##     upper      mean     lower
## 0.8123914 0.8113497 0.8103080
```

The boostrap method produced a narrow interval centred around a value close to the least-square estimate of theta, but the least squares estimate falls outside of the CI.

Question 3

(a)

```
GLB.Ts_dSST = read.csv('GLB.Ts_dSST.csv', header = T,
                       sep = ',', na.strings = '***')
mydata = GLB.Ts_dSST[, 2:13]
yearly.temp = apply(mydata, 1, mean)
year = 1880:2019
plot(year, yearly.temp, type = 'l', xlab = 'Year', ylab = 'Average Temperature',
      main= 'Average Temperature Over Years 1880 - 2019', col = 1)
```

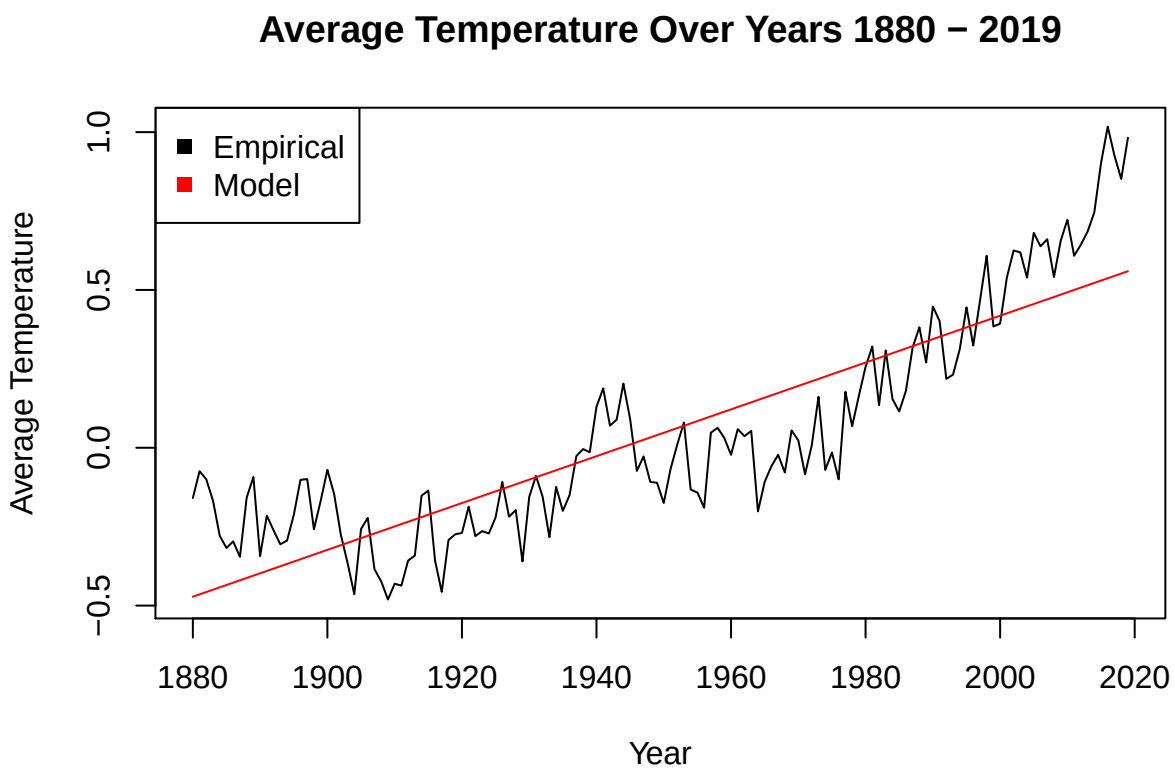


We see there is a cyclic pattern that reveals itself in the periods 1880-1940, and 1940-upwards, but after that, the trend is increased average temperature which is similar to the overall trend.

(b)

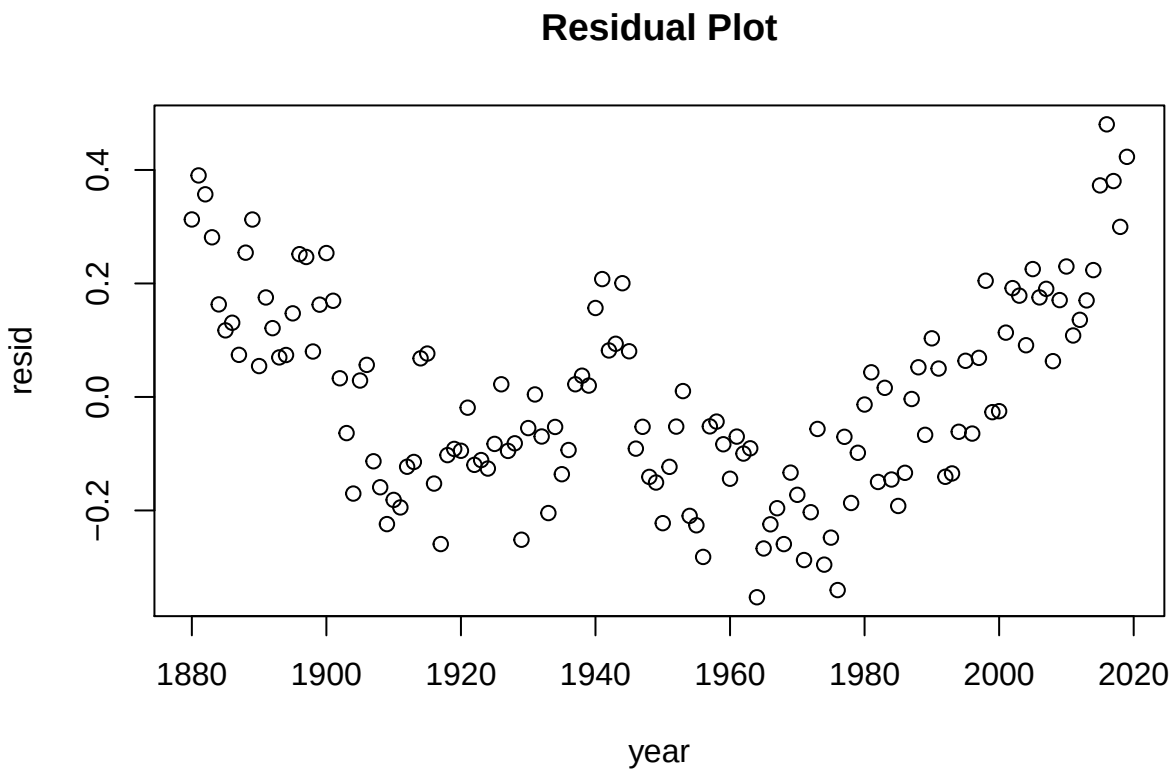
```
# This is an objective function based on the sum of squares
# here a = par[1], b = par[2]
sumsqr = function(par) {
  result = sum((yearly.temp - par[1] - par[2]*year)^2)
  return(result)
}
ls.est = nlminb(start = c(-10, 0.1), objective = sumsqr)$par
linear.fit = ls.est[1] + ls.est[2]*year
resid = yearly.temp - linear.fit
plot(year, yearly.temp, type = 'l', xlab = 'Year', ylab = 'Average Temperature',
```

```
     main= 'Average Temperature Over Years 1880 - 2019', col = 1)
lines(year, linear.fit, col = 2)
legend(x = 'topleft', col = c(1, 2), legend = c('Empirical', 'Model'), pch = 15)
```



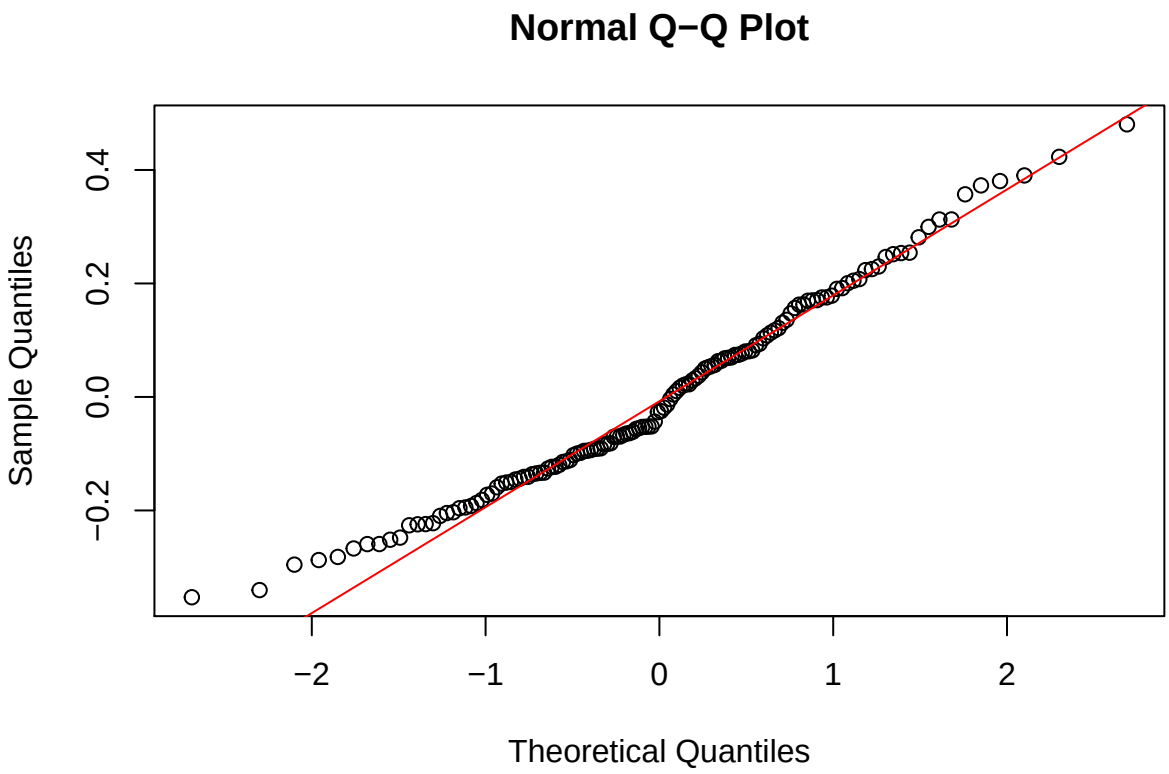
The fitted line captures the overall trend, however, it is not a great model as it neglects the cyclical pattern I mentioned earlier. Morever, it underestimates the temperature for Years > 2000.

```
plot(x = year, y = resid, main = "Residual Plot")
```



Clearly, the residuals make a convex pattern in the cycles 1880-1940, and 1940-upwards. This suggests that a linear model is inappropriate for the given data.

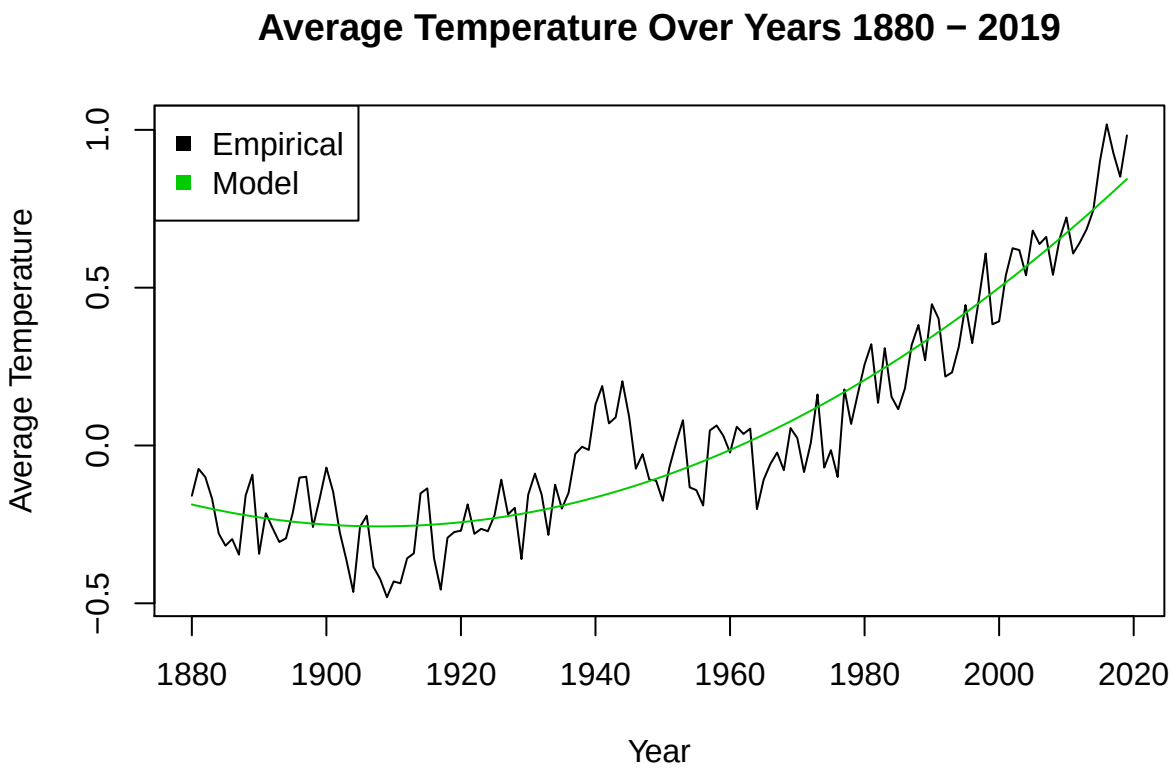
```
qqnorm(resid)
qqline(resid, col = 2)
```



Based on the plot, the ends and centre of the data do not conform to the straight line, hence the residuals are not normally distributed.

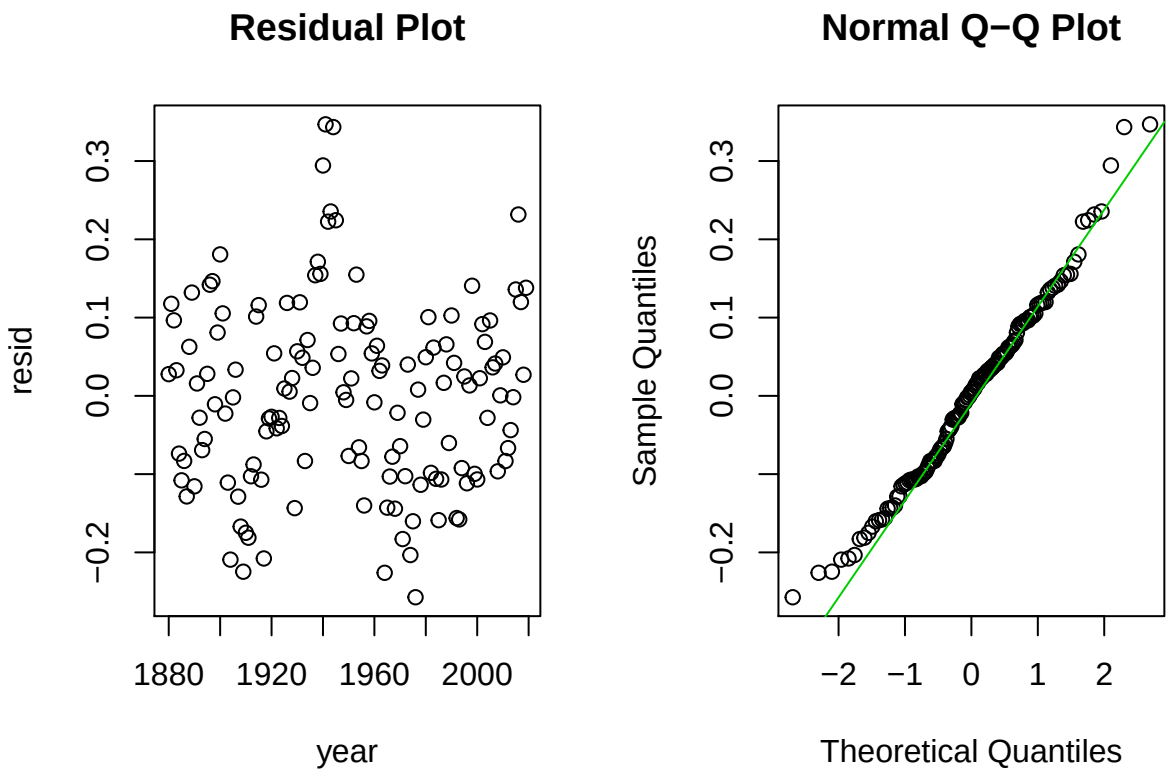
(c)

```
# This is an objective function based on a quadratic transformation to the data
# here a = par[1], b = par[2], c = par[3]
sumsqr = function(par) {
  result = sum((yearly.temp - par[1] - par[2]*year - par[3]*year^2)^2)
  return(result)
}
ls.est = nlminb(start = c(300, -1, 0.1), objective = sumsqr,
                scale = c(1, 300, 1000))$par
quadratic.fit = ls.est[1] + ls.est[2]*year + ls.est[3]*year^2
resid = yearly.temp - quadratic.fit
plot(year, yearly.temp, type = 'l', xlab = 'Year', ylab = 'Average Temperature',
      main= 'Average Temperature Over Years 1880 - 2019', col = 1)
lines(year, quadratic.fit, col = 3)
legend(x = 'topleft', col = c(1, 3), legend = c('Empirical', 'Model'), pch = 15)
```



This model outperforms the linear model. It captures both the cyclical pattern as well as the overall trend. Additionally, it does not deviate a lot from the actual temperatures.

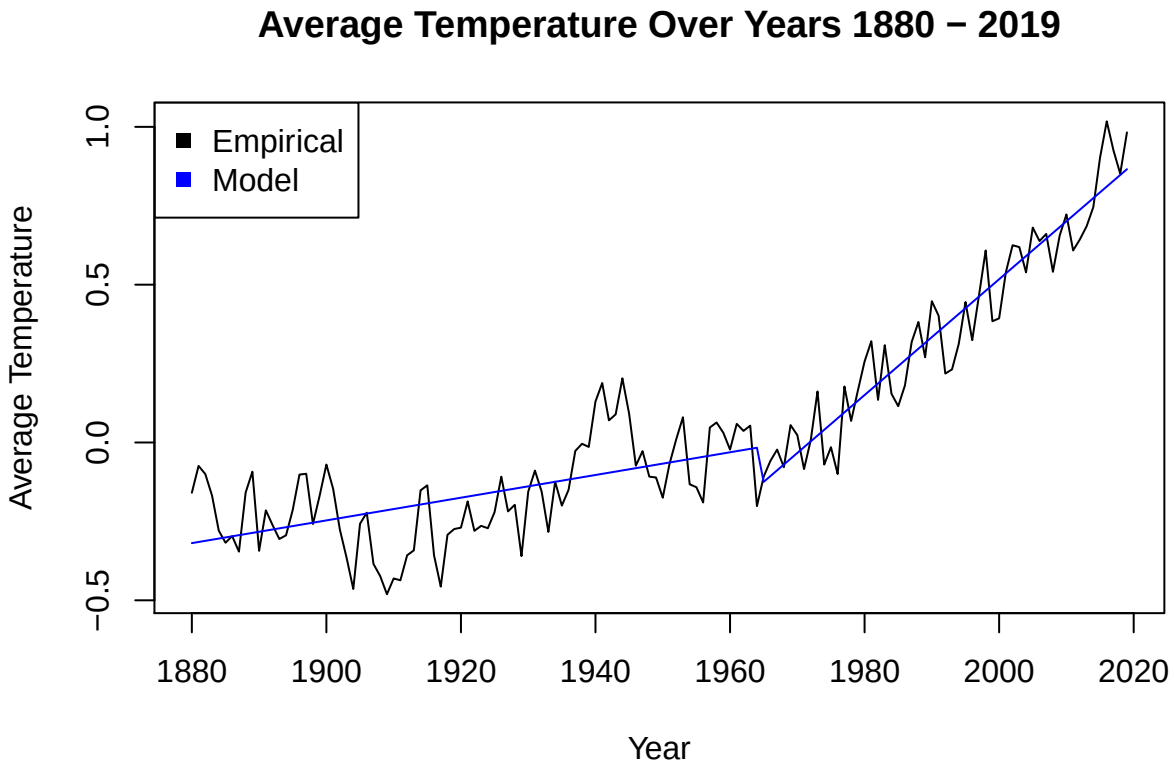
```
par(mfrow= c(1, 2))
plot(x = year, y = resid, main = "Residual Plot")
qqnorm(resid)
qqline(resid, col = 3)
```



The residual plot shows not so random residuals (rather, they are just the same as the linear model but amplified), and the QQ-plot shows a close fit to normality except at the extrema, hence this model may not be appropriate.

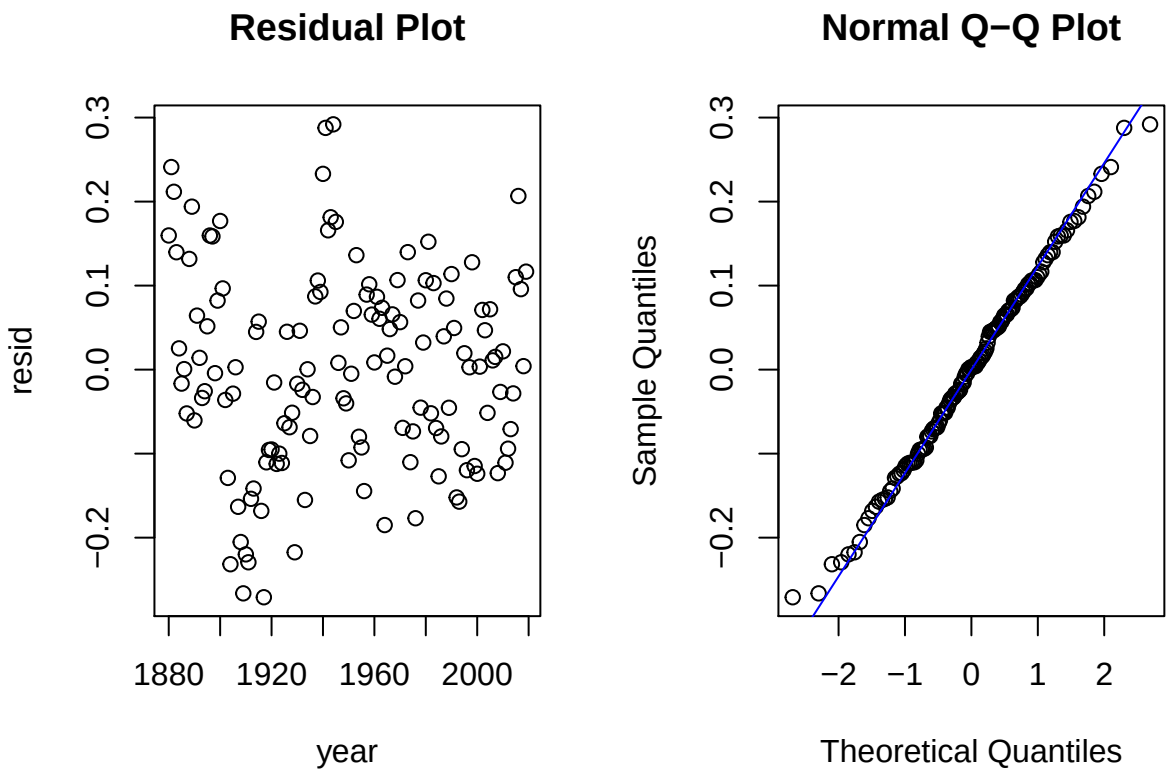
(d)

```
# This is an objective function with a dummy vector
# here a = par[1], b = par[2]
dummy= c(rep(0,85), rep(1,55))
sumsqr = function(par) {
  result = sum((yearly.temp - par[1] - par[2]*year - par[3]*dummy - par[4]*dummy*year)^2)
  return(result)
}
ls.est = nlminb(start = c(-1, 1, -1, 1), objective = sumsqr)$par
dummy.linear.fit = ls.est[1] + ls.est[2]*year + ls.est[3]*dummy + ls.est[4]*dummy*year
resid = yearly.temp - dummy.linear.fit
plot(year, yearly.temp, type = 'l', xlab = 'Year', ylab = 'Average Temperature',
      main= 'Average Temperature Over Years 1880 - 2019', col = 1)
lines(year, dummy.linear.fit, col = 4)
legend(x = 'topleft', col = c(1, 4), legend = c('Empirical', 'Model'), pch = 15)
```



This model seems to do well in generalizing to the trend in every cycle without overfitting to the data. This model is great for making predictions. The kink around 1965 is looking unnatural.

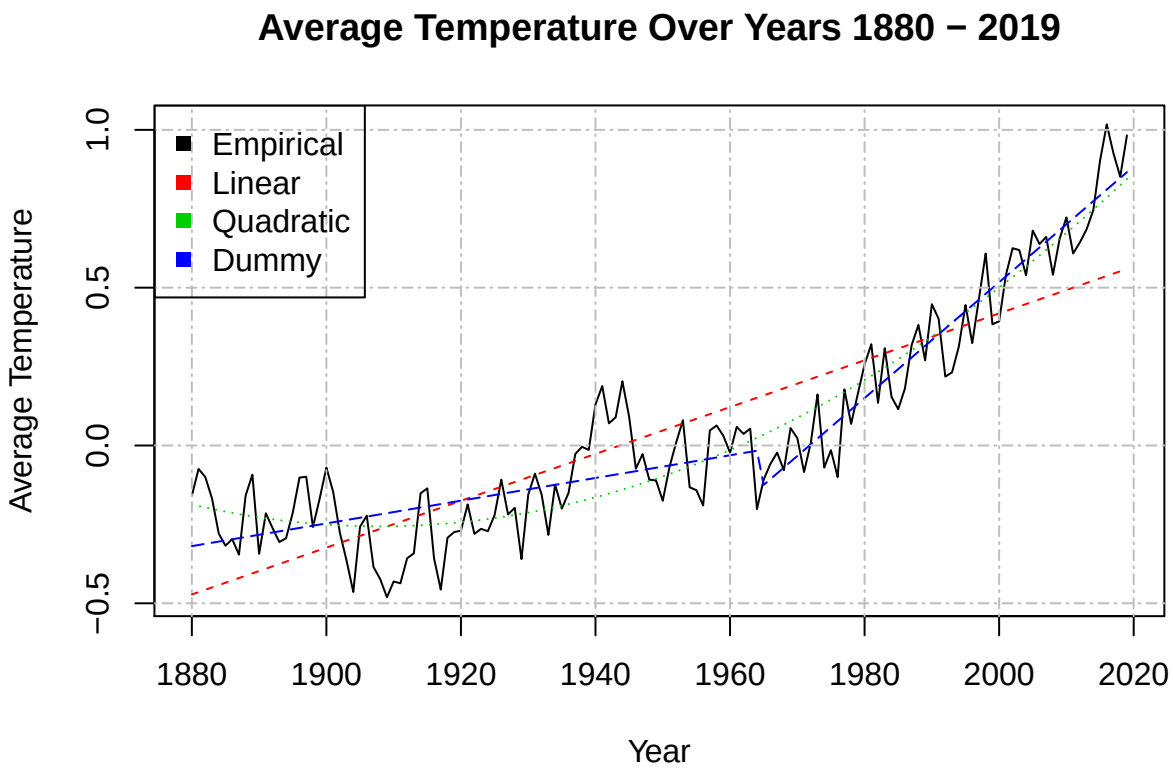
```
par(mfrow= c(1, 2))
plot(x = year, y = resid, main = "Residual Plot")
qqnorm(resid)
qqline(resid, col = 4)
```



This model has completely random residuals, and the quantiles lies almosr perfectly on the theoretical line, which is evidence for normality.

(e)

```
plot(year, yearly.temp, type = 'l', xlab = 'Year', ylab = 'Average Temperature',
      main= 'Average Temperature Over Years 1880 - 2019', col = 1)
lines(year, linear.fit, col = 2, lty = 'dashed')
lines(year, quadratic.fit, col = 3, lty = 'dotted')
lines(year, dummy.linear.fit, col = 4, lty = 'longdash')
grid(lty = 'twodash', col = 'gray')
legend(x = 'topleft', col = 1:4, legend = c('Empirical', 'Linear', 'Quadratic', 'Dummy'),
       pch = 15)
```



All models show overall trend of increasing yearly average temperature. The quadratic model provides a smooth curve which is very natural and nice to work with for theoretical purposes (i.e differentiable). It closely matches the mean of the interval neighborhood it is at, however, it does not quite capture the cyclical pattern that renews around year 1940, and rather just captures a bigger concave-up pattern. It has risks of overfitting to the data, so it should not be used for predictions. The linear model is terrible to say the least. While it is very simple and easy to work with theoretically, it is highly biased. Finally, the dummy-linear model is the best since it captures the cyclical pattern by changing the slope around the kink. Additionally, it does not overfit to the data. This means it is great for predictions. It maintains low bias and low variance. However the kink around 1960 provides a challenege to work with the model in a theoretical setting.

Based on the models above, the average temperature per decade is increasing polynomially over time. This is a concern expressed climate scientists and activists.