# Mortgages

Nathan Cantafio

October 15, 2024

# Contents

# Motivations

I was playing around with calculating the total cost of a mortgage, and I realized that for $700000 home, you could easy pay $1.1 million on a mortgage. Growing up my dad has always thought that renting is a waste of money, but this got me wondering if that's actually true. Could renting sometimes be a better financial choice than buying?

```
# returns monthly payment to pay off mortgage in the specified time (in years)
# interest rate is annualized and downpayment is given as a percentage
# assumes monthly interest accrual and payments
payment <- function(principal=500000, rate=0.05, downpayment=0.05, time = 25) {
  return((rate/12)*((1+rate/12)^(12*time))*
           (principal-downpayment*principal)/((1+rate/12)^(12*time)-1))
}
300*payment(principal = 700000) # total cost of mortgage is over 1.1 million!!!
```

```
## [1] 1166257
```

This question essentially boils down to finding a way to estimate the total cost of a mortgage. For fixed rate mortgages this comes down to some linear algebra. For variable rate mortgages, we need a way to predict (or at least generate plausible) interest rates. This is a challenging problem. In this project I will develop a few different methods for doing this, and go over their pros and cons. In the end we will compare the total cost of a mortgage to a rental and see if we can say anything meaningful.

# Background

In this section we will develop some of the theory that we later apply to the problem of predicting (or generating) interest rates. This involves topics ranging from numerical methods to statistical inference/probability theory.

## Monthly payment formula for a fixed rate mortgage

To find a formula for what the monthly payment must be in order to pay off a fixed rate mortgage in a specified number of years I used linear algebra to solve the recurrence. This is unique compared to the other approaches I've seen online which used sums.

Assume that interest is accrued over the same period as payments are made. Let $a_n$ be the amount owed after $n$ payment periods. Then

$$a_n = (1+r)a_{n-1} - c,$$

where $r$ is the interest rate and $c$ is the monthly payment.

This relation can be rewritten as:

$$\begin{bmatrix} a_n \\ c \end{bmatrix} = \begin{pmatrix} 1+r & -1 \\ 0 & 1 \end{pmatrix} \begin{bmatrix} a_{n-1} \\ c \end{bmatrix},$$

and then further rewritten as:

$$\begin{bmatrix} a_n \\ c \end{bmatrix} = \begin{pmatrix} 1+r & -1 \\ 0 & 1 \end{pmatrix}^n \begin{bmatrix} a_0 \\ c \end{bmatrix} = A^n \begin{bmatrix} a_0 \\ c \end{bmatrix}.$$

Call the matrix $A$. Then $A = MDM^{-1}$ where $D$ is a diagonal matrix. In fact,

$$A^n = \begin{pmatrix} 1/r & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1+r \end{pmatrix}^n \begin{pmatrix} 0 & 1 \\ 1 & -1/r \end{pmatrix} = \begin{pmatrix} (1+r)^n & \frac{1}{r}(1-(1+r)^n) \\ 0 & 1 \end{pmatrix}.$$

And we have:

$$a_n = a_0(1+r)^n + c\frac{1-(1+r)^n}{r}.$$

If the goal is to pay off the mortgage in $N$ periods, then setting $a_N = 0$ yields:

$$c = a_0\frac{r(1+r)^N}{(1+r)^N - 1}, \qquad \text{where } a_0 \text{ is the principal.}$$

## Vasicek and CIR models

The Vasicek and CIR models use Stochastic Differential Equations to model interest rates.

The basic idea of both is that there is a value (which we call $\theta$) to which the interest rate should gravitate towards over long enough time scales.

Let $r$ be the interest rate we are modeling. Then the change in the rate, $dr$ should be in the direction $\theta - r$ so that

$$r + dr = r + \theta - r = \theta,$$

in other words $r$ is being pulled towards $\theta$.

We don't want it to be pulled all the way though, otherwise the model wouldn't be very interesting. So we add another parameter $\alpha$ which acts as the speed. Thus

$$dr = \alpha(\theta - r).$$

The above on its own is called the drift term. There is also some volatility or random noise introduced with a parameter sigma and voila:

$$dr = \alpha(\theta - r)dt + \sigma dW.$$

Above is the Vasicek model, we add a $dt$ into the equation to represent the time-step.

The CIR model is similar but volatility is also proportional to the square root of the rate:

$$dr = \alpha(\theta - r)dt + \sigma\sqrt{r}dW.$$

The CIR model is nice because at least analytically, as long as $2\alpha\theta > \sigma^2$ (Feller condition) then the rate will never be negative. This is because when $r$ becomes close to zero, the $\sqrt{r}$ term becomes very small and allows the drift term to dominate over the volatility term; pulling $r$ up. However, this does not work out so nicely in simulation due to errors introduced by discretization which can result in negative rates.

We can discretize the CIR model as follows (Euler discretization scheme):

$$r[t] = r[t-1] + \alpha(\theta - r[t-1])dt + \sigma\sqrt{r[t-1]}dW.$$

We can then rearrange this as:

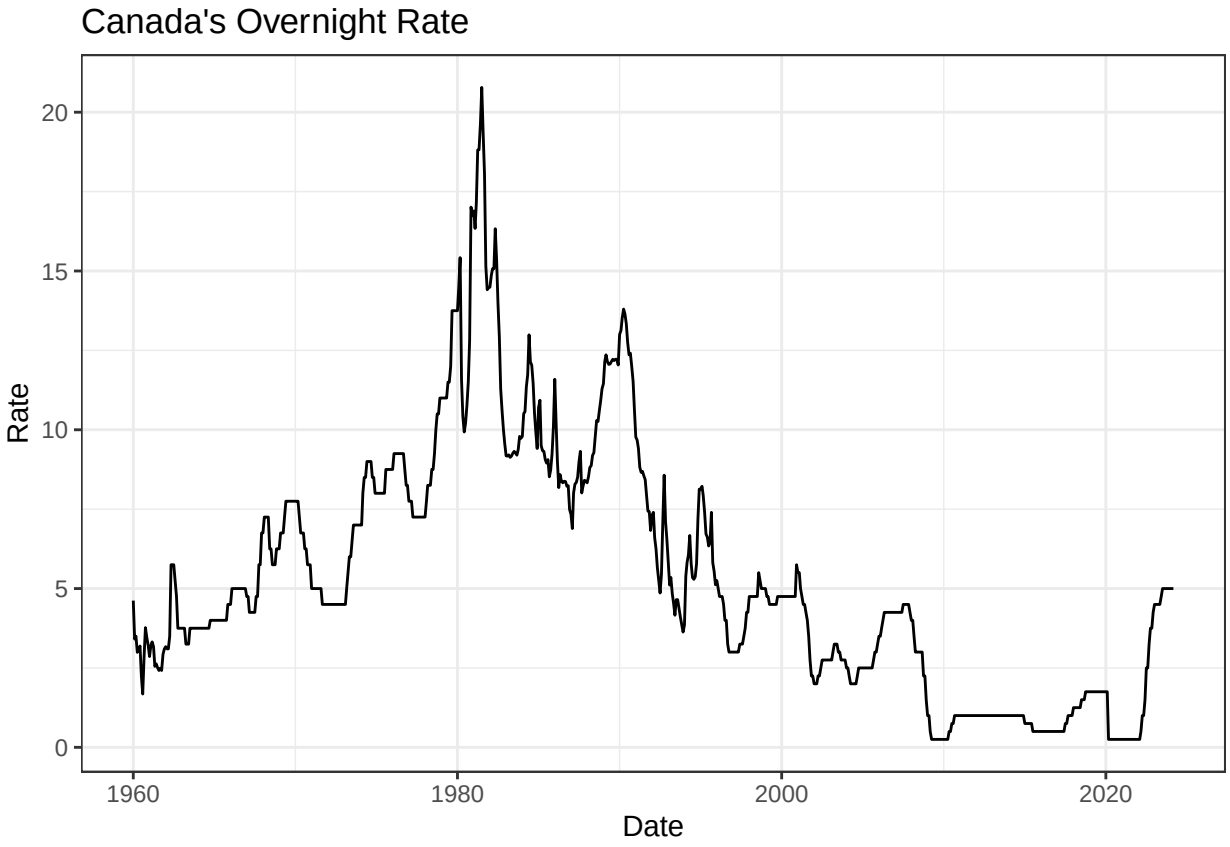
$$\frac{r[t]}{\sqrt{r[t-1]}} = a\frac{r[t-1]}{\sqrt{r[t-1]}} + b\frac{1}{\sqrt{r[t-1]}} + \varepsilon,$$

where $a = 1 - \alpha dt$, $b = \alpha\theta dt$ and $\varepsilon = \sigma dW$ so that we can run a linear regression.

## Predicting (or generating) interest rates

Here is a plot of Canada's overnight rate from 1960 to March 2024

```
# load data
canadarates <- read.csv(file = "canadarates.csv")
colnames(canadarates) <- c("date", "prime", "overnight")
canadarates <- tail(canadarates, -26)
dates <- as.Date(canadarates$date, format = "%d %B %Y")
canadarates$date <- dates

ggplot(data = canadarates) +
  geom_line(aes(x = date, y = overnight)) +
  labs(title = "Canada's Overnight Rate", y = "Rate", x = "Date") +
  theme_bw()
```




This is the data we will use to generate our predictions.

### Approach 1 - Generating plausible interest rates

Rather than trying to make predictions about future interest rates from previous rates, what if we had access to the distribution of interest rate trajectories? Then we could check how often in this distribution is, for example, the total cost of a mortgage more than double the principal?

In some sense we are trying to estimate the distribution of a vector in $\mathbb{R}^T$ where $T$ is the number of rates we have observed, and each component is likely highly correlated with each other. And all from a single

observation! When I write it down like this it seems fruitless. When I first started this project I didn't have as clear an understanding of what I was doing. In any case here are the results of this first approach.

I chose to use a CIR model to avoid negative interest rates. However, due to errors introduced by the discretization, I added a step of taking the absolute value. The function for simulating a CIR model is below.

```
cir <- function(alpha, theta, sigma, steps) {
  dt <- 1 # time step

  r <- vector(length=steps)
  r[1] <- theta # initial interest rate
  # r[1] <- 5 # could choose any initial rate we want


  for (i in 2:steps) {
    dW <- rnorm(1, mean=0, sd=1) # generate random noise

    r[i] <- abs(r[i-1] + alpha*(theta - r[i-1])*dt +      # modified euler
                  sigma*sqrt(r[i-1])*dt*dW)
  }

  return(r)
}
```

To choose parameters that yield realistic interest rates, I used data of Canada's overnight rate from 1960 to March of 2024. I then ran a linear regression, predicting the last $n-1$ rates from the first $n-1$ rates. Using the formula from before to get the parameters from the coefficients. These parameters act as an initial guess for the MLE estimation which follows. I could have just stopped at the regression, but I wanted to use MLE to refine and hopefully improve the choice of parameters.

```
r = canadarates$overnight/100
n = length(r)

rates <- data.frame(date = dates, rate = r)

# predicting last n-1 rates based on the first n-1 to fit CIR Model

x <- r[-n]
y = r[-1]/sqrt(x)

X <- matrix(c(x/sqrt(x), rep(1, n-1)/sqrt(x)), ncol=2)
B <- solve(t(X)%*%X)%*%t(X)%*%y

e <- y - X%*%B # for calculating standard error

# r[t]/sqrt(r[t-1]) ~ a*r[t-1]/sqrt(r[t-1]) + b*(1/sqrt(r[t-1])) + ep
# a = (1 - alpha*dt), b = alpha*theta*dt, ep = sigma*dW

a <- B[1]
b <- B[2]
MSE <- (t(e)%*%e)[1,1]/(n-3)

dt <- 1 # time step
```

```
# initial parameters from least squares regression
alpha = (1 - a)/dt # Long-term mean or equilibrium interest rate
theta = b/(1 - a) # Speed of mean reversion
sigma <- sqrt(MSE)/sqrt(dt) # Volatility
```

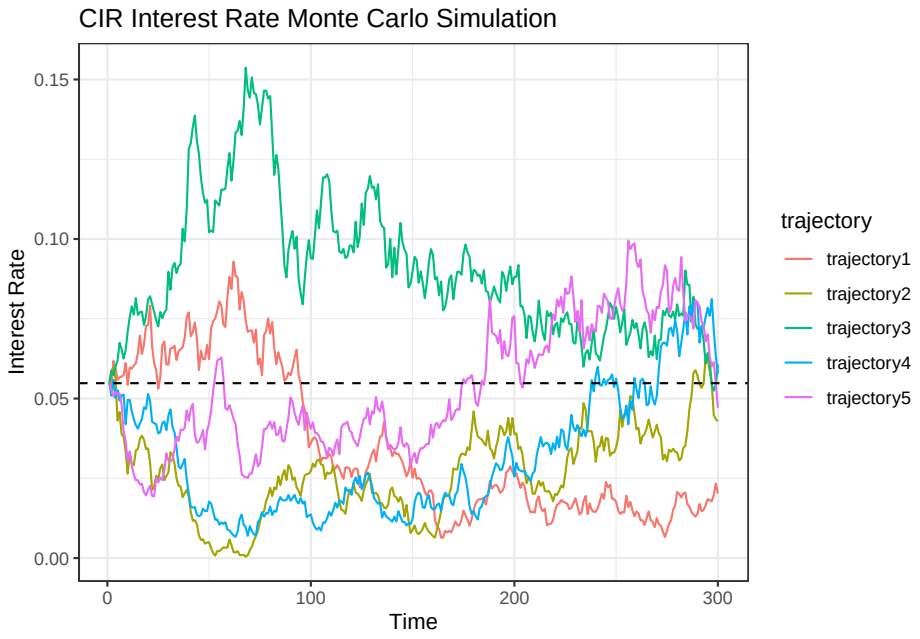
There are two ways to calculate the Log-Likelihood for the CIR model both of which are implemented in the file `functions.R`. The first involves the Bessel function, and the second involves Chi-Squared distribution.

```
# Now refine the value of parameters with ML estimation

# returns the ln Likelihood function for the CIR Model
# enter the data and time step inside the function definition
logLikelihood <- function(param) {
  result <- lnLhelper(param=param, data=r, dt=1)
  return(result)
}

# optimize using optim()
MLE <- optim(c(alpha, theta, sigma), logLikelihood)$par
```

Now that we have chosen parameters, we can generate samples.

```
# Simulate the rates
# Set number of simulation steps
steps <- 300

# choose which parameters to use
param <- MLE

# Create time vector
time <- 1:steps

# Create data frame to store trajectories
trajectories <- as.data.frame(c(time))

# Choose how many simulated trajectories to generate
N <- 5

# Run loop to generate N different simulated trajectories
for (i in 1:N) {
  trajectories <- cbind(trajectories, cir(param[1], param[2], param[3], steps))
}
colnames(trajectories) <- c('time', paste("trajectory", 1:N, sep = "")) # rename columns

# for plotting purposes in ggplot
trajectories <- melt(trajectories, id.vars = 'time', variable.name = 'trajectory')

ggplot(data = trajectories, aes(time, value)) +
  geom_line(aes(colour = trajectory)) +
  geom_hline(yintercept = param[2], linetype = "dashed") +
  xlab("Time") +
  ylab("Interest Rate") +
```

```
  ggtitle("CIR Interest Rate Monte Carlo Simulation") +
  theme_bw()
```




If the resulting samples do a good job of characterizing the true distribution of the interest rates, then I really like this approach. (This is something worth trying to prove yes or no).

Another approach is to try and make predictions. Instead of thinking about the distribution of all possible trajectories, think about what the future interest rates will be given the current and previous ones.

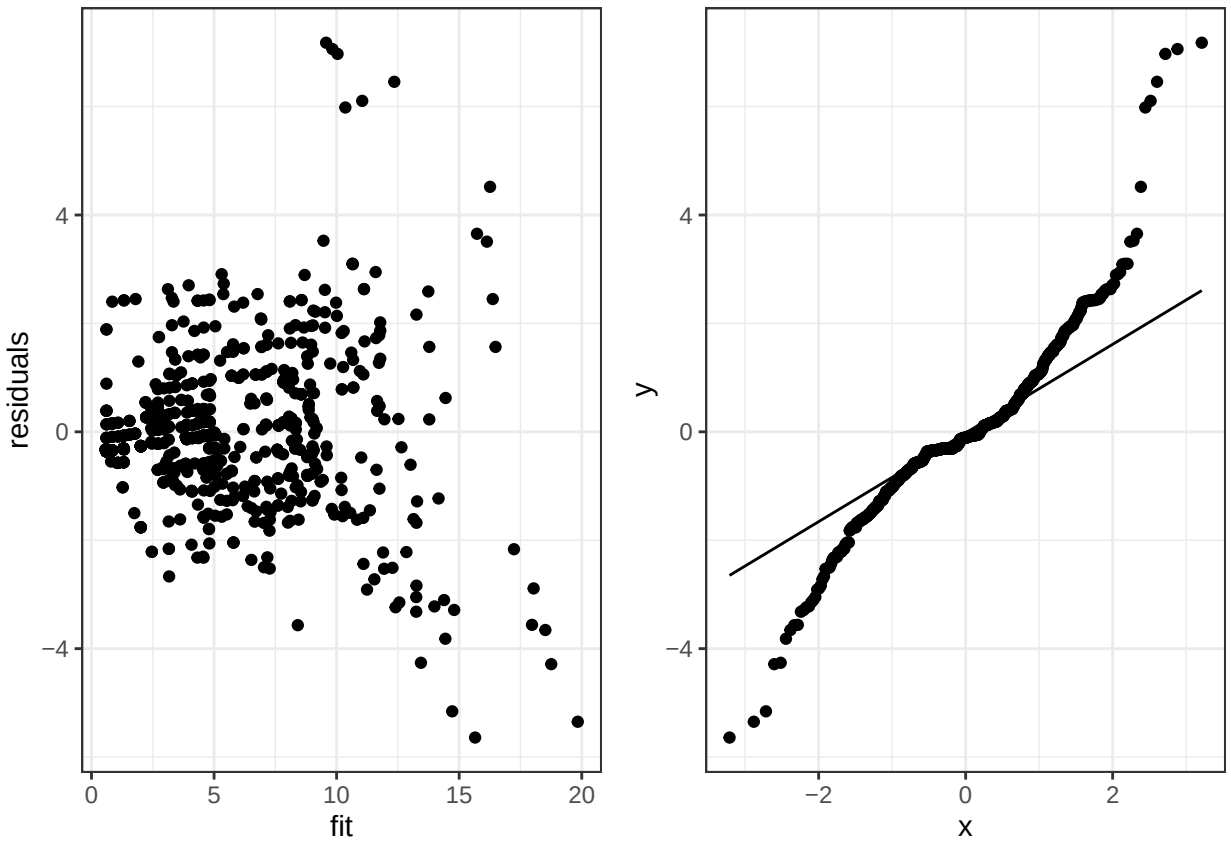
What follows are different methods for predicting interest rates from past rates.

## Approach 2 - Predicting interest rates with a linear model

Now we will fit a linear model using OLS. We will try predicting the overnight rate $t$ months from now, using the rate now, and the rates every 6 months ago until we hit $t$ months ago. For example, we might predict the rate 18 months from now using the rates now, 6 months ago, 12 months ago, and 18 months ago.

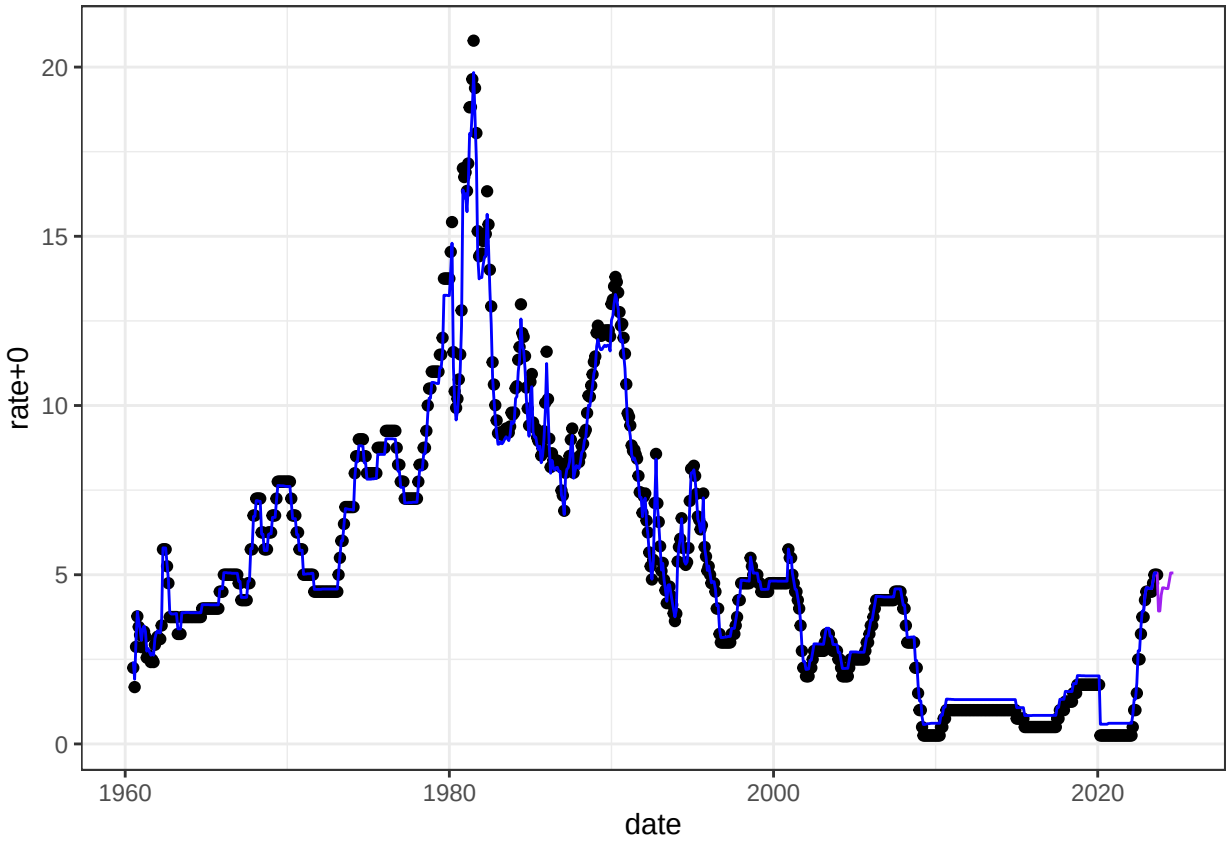
```
rate = canadarates$overnight
date = canadarates$date

# want to predict t months into future (t should be divisible by 6)
# coerce data into desired format
t <- 6
n = length(rate)
date = date[(t+1):(n-t)]
`rate+0` = rate[(t+1):(n-t)]
`rate+t` = rate[(2*t+1):n]
rates <- cbind.data.frame(date, `rate+t`, `rate+0`)
# rate-t, ... rate-(t-6), ... rate-6
for (i in 1:(t/6)) {
  rates <- cbind.data.frame(rates, rate[(1 + 6*(i-1)):(n - 2*t+6*(i-1))])
}
```

```r
colnames(rates) <- c('date', 'rate+t', 'rate+0', paste("rate-", seq(t, 6, by=-6), sep=""))

# regress `rate+t` on `rate+0`, `rate-6`, ..., `rate -(t-6), and `rate-t`
model <- lm(`rate+t` ~. - date, data = rates)

# residual and qq plots
residuals <- data.frame(fit = model$fitted.values, residuals = model$residuals)
resplot <- ggplot(data = residuals) +
  geom_point(aes(x = fit, y = residuals)) +
  theme_bw()
qqplot<- ggplot(residuals, aes(sample = residuals)) +
  stat_qq() +
  stat_qq_line() +
  theme_bw()
cowplot::plot_grid(resplot, qqplot)
```




```r
#fitted values
someRates = tail(rates, 2*t)
someDates = seq.Date(from = someRates$date[1], by ="month", length.out = 4*t)

pred = data.frame(date = someDates, predictions = predict(model, someRates), row.names = NULL)
ggplot() +
  geom_point(data = rates, mapping = aes(x = date, y = `rate+0`)) +
  geom_line(data = pred, mapping = aes(x = date, y = predictions), colour = "purple") +
  geom_line(data = data.frame(date = rates$date, pred = model$fitted.values),
```

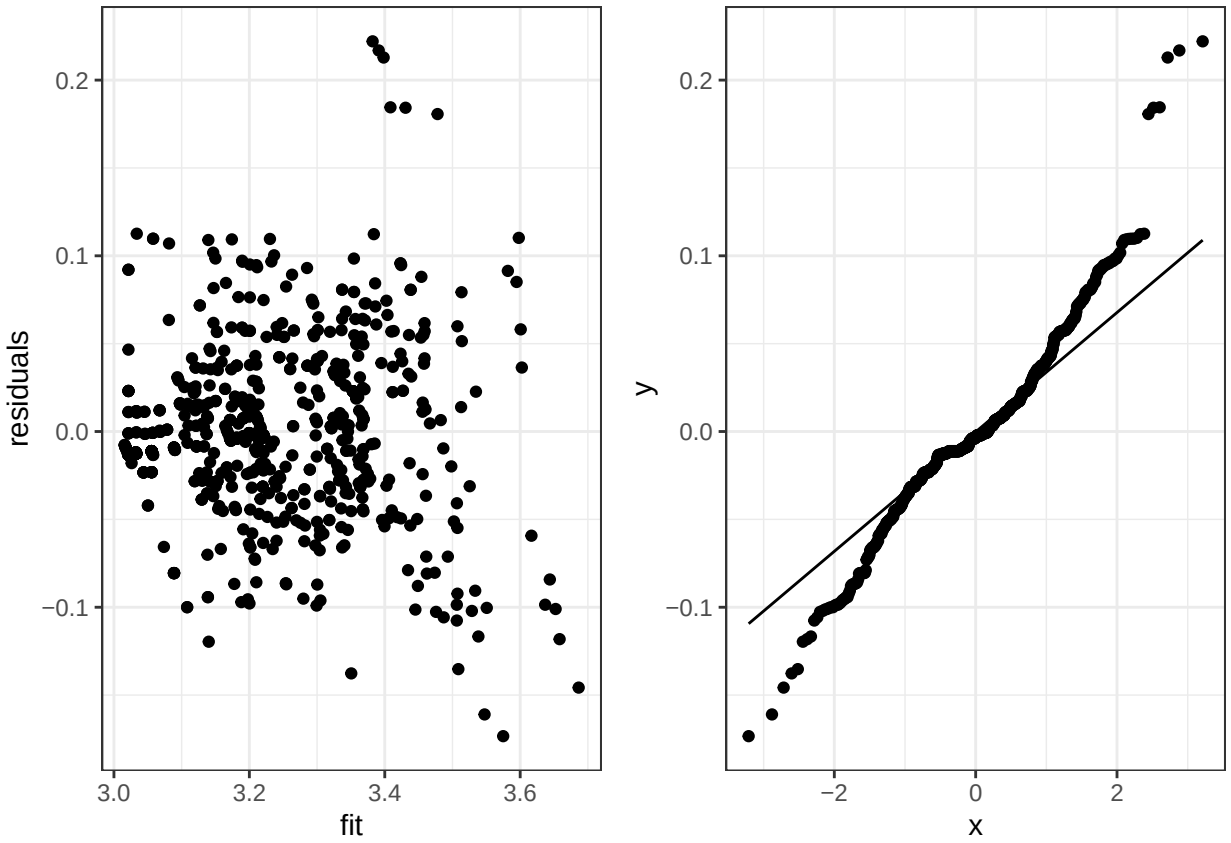
```
            mapping = aes(x = date, y = pred), colour = "blue") +
  theme_bw()
```




From the residuals and qq plots, the assumptions of a linear model do not seem to be met, and using a linear model does not seem justified. However, in all honesty the predictions aren't horrible.

But maybe if we transform our data, we can do an even better job.

```
lambda = 20
tr <- function(d) {
  log(lambda + d)
}

rates_tr <- rates %>%
  mutate(across(`rate+t`:`rate-6`, tr))

model <- lm(`rate+t` ~. -date, data = rates_tr)
residuals <- data.frame(fit = model$fitted.values, residuals = model$residuals)
resplot <- ggplot(data = residuals) +
  geom_point(aes(x = fit, y = residuals)) +
  theme_bw()
qqplot<- ggplot(residuals, aes(sample = residuals)) +
  stat_qq() +
  stat_qq_line() +
  theme_bw()
cowplot::plot_grid(resplot, qqplot)
```

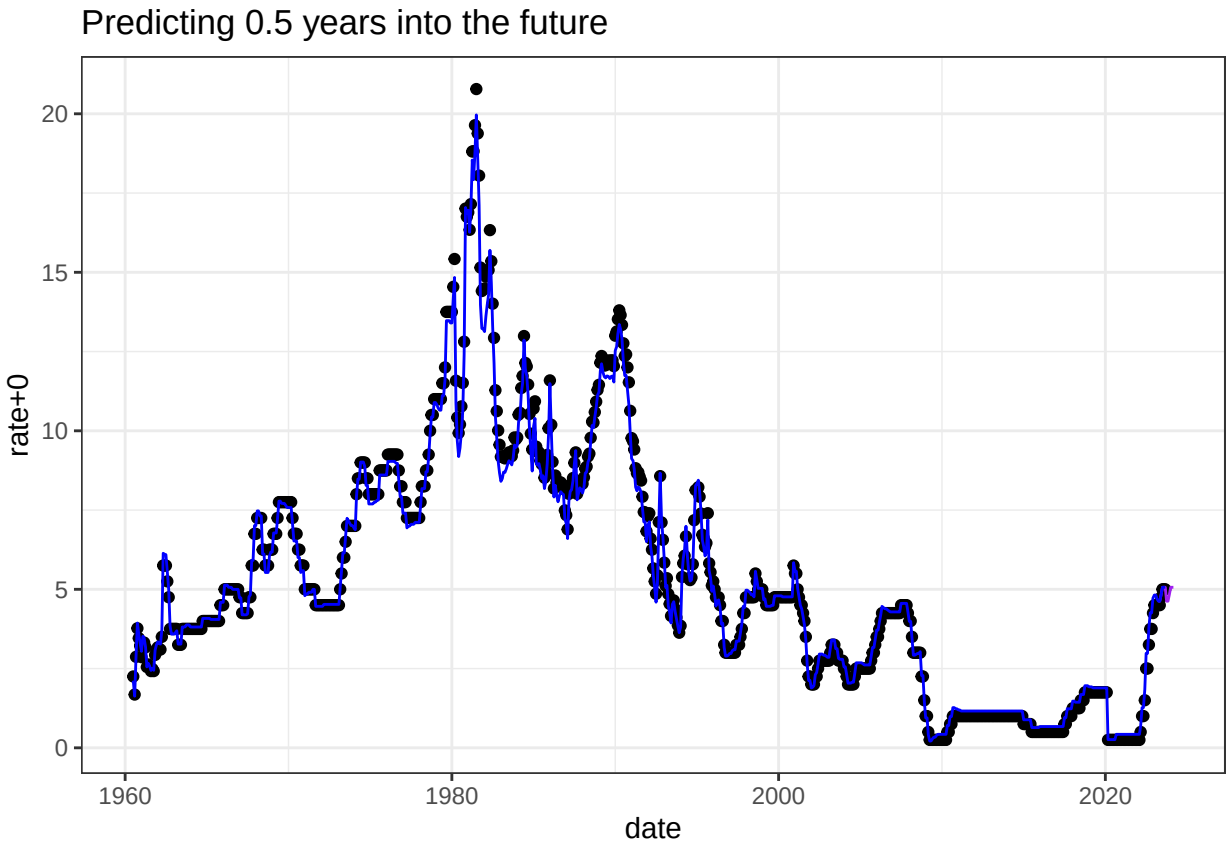



From the residual and qqplots, it seems like this made our data a little bit more suited towards a linear model. I don't mind transforming the data in this way because I am not trying to use this model for any interpretations - just making predictions. We will choose a suitable value of lambda with cross validation.

```
# find the value of lambda which results in the lowest cv score
lambda = seq(0, 20, by = 0.5)
cv <- transform_cv(rates, lambda, kfolds = 10, t = t)
lambda.min = lambda[which(cv == min(cv))]
```

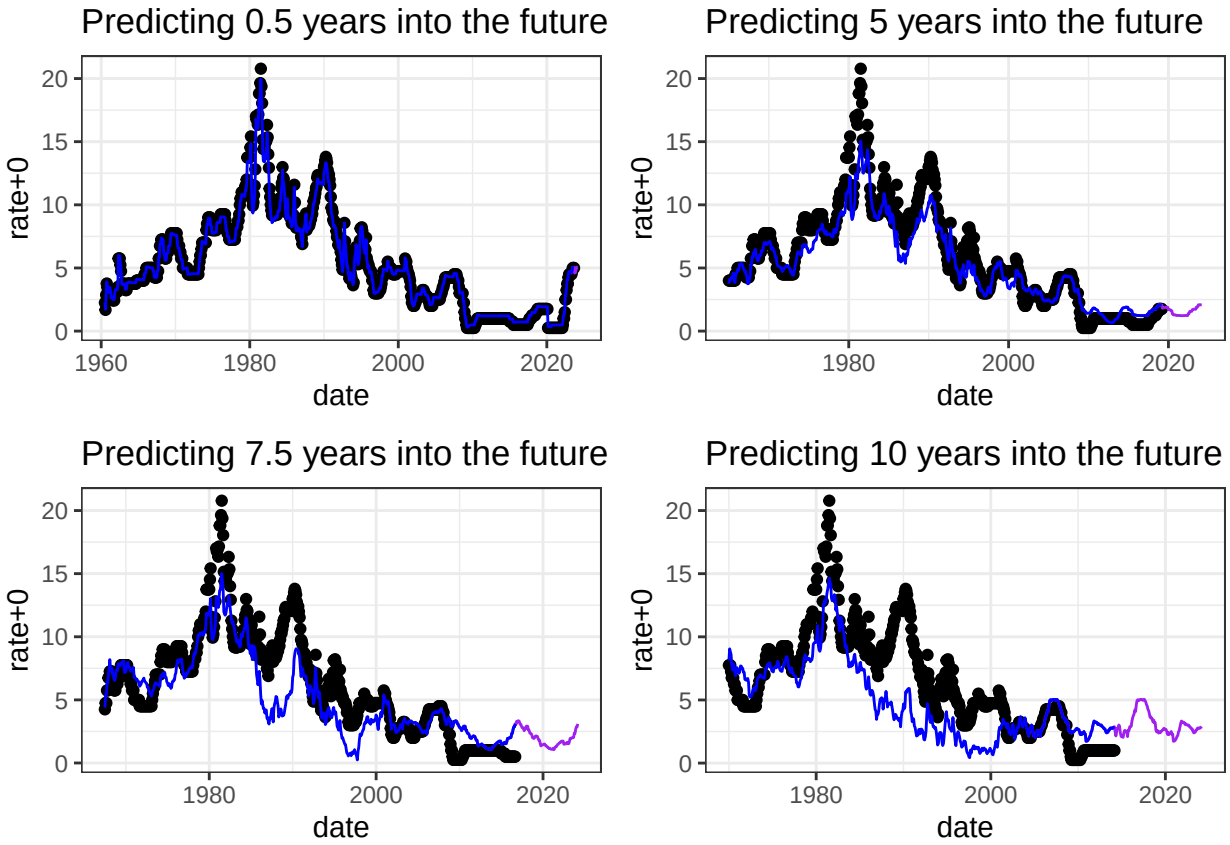
```
lambda = lambda.min
tr <- function(d) {
  log(lambda + d)
}
rates_tr <- rates %>%
  mutate(across(`rate+t`:`rate-6`, tr))
model <- lm(`rate+t` ~. -date, data = rates_tr)

# plot fitted model + future predictions + actual values
someRates = tail(rates_tr, t)
someDates = seq.Date(from = someRates$date[1], by ="month", length.out = 2*t)

pred = data.frame(date = someDates, predictions = exp(predict(model, someRates)) - lambda,
                  row.names = NULL)
ggplot() +
  geom_point(data = rates, mapping = aes(x = date, y = `rate+0`)) +
  geom_line(data = pred, mapping = aes(x = date, y = predictions), colour = "purple") +
  geom_line(data = data.frame(date = rates_tr$date,
```

```
                              pred = (exp(model$fitted.values)-lambda)),
          mapping = aes(x = date, y = pred), colour = "blue") +
  labs(title = paste("Predicting", t/12, "years into the future")) +
  theme_bw()
```




We can see that the model does a pretty nice job of fitting the data! I would say slightly better than the untransformed version. And we even get some predictions for future interest rates in purple. Unfortunately, the further out we predict the worse our fit is.

```
p1 <- OLS_t(6); p2 <- OLS_t(60); p3 <- OLS_t(90); p4 <- OLS_t(120)
cowplot::plot_grid(p1, p2, p3, p4)
```




This is unsurprising since to predict further, we need to essentially shrink our data size.

Overall I'm not a huge fan of this approach. It makes sense that subsequent interest rates would be correlated, so because of the way we set up the model, we shouldn't really make the assumption of independent predictors (which we implicitly have to by using OLS).

A better method is one that is tailored to time series data.

### Approach 3 - Predicting interest rates with Box-Jenkins

This leads us to the Box-Jenkins Method.

A disadvantage of this method is that there isn't really a programatic way of predicting future rates given some history. We need to redo the entire analysis.

### Approach 4 - Revisiting approach 1 (resampling over shorter periods)

## Results of the various approaches

## Comparing mortgaging and renting

## Conclusions