Smoothing for Time Series Regression in R

William Chiu

5/7/2022

Data and Noise

Suppose y_t is a linear function of x_t and random error:

$$y = \beta_0 + \beta_1 x_t + \epsilon$$

```
set.seed(1)

nobsFull <- 1500
nobs <- 1000

x <- log(1:nobsFull) + rnorm(nobsFull, 0, 0.25)
y <- 2 + 3 * x + rnorm(nobsFull)

df_full <- data.frame(t=1:nobsFull, y=y, x=x)

rm(x, y)
```

But also suppose that x_t is not directly observable, instead we observe its noiser cousin that contains random errors. For example, the instrument that measures x_t suffers from random imprecision.

```
df_full$x_noisy <- df_full$x + rnorm(nobsFull,0, 1)

df <- df_full[1:nobs, ]

df_test <- df_full[(nobs + 1):nobsFull,]
```

Since x_t is observed with noisy imprecision, this degrades the OLS fit.

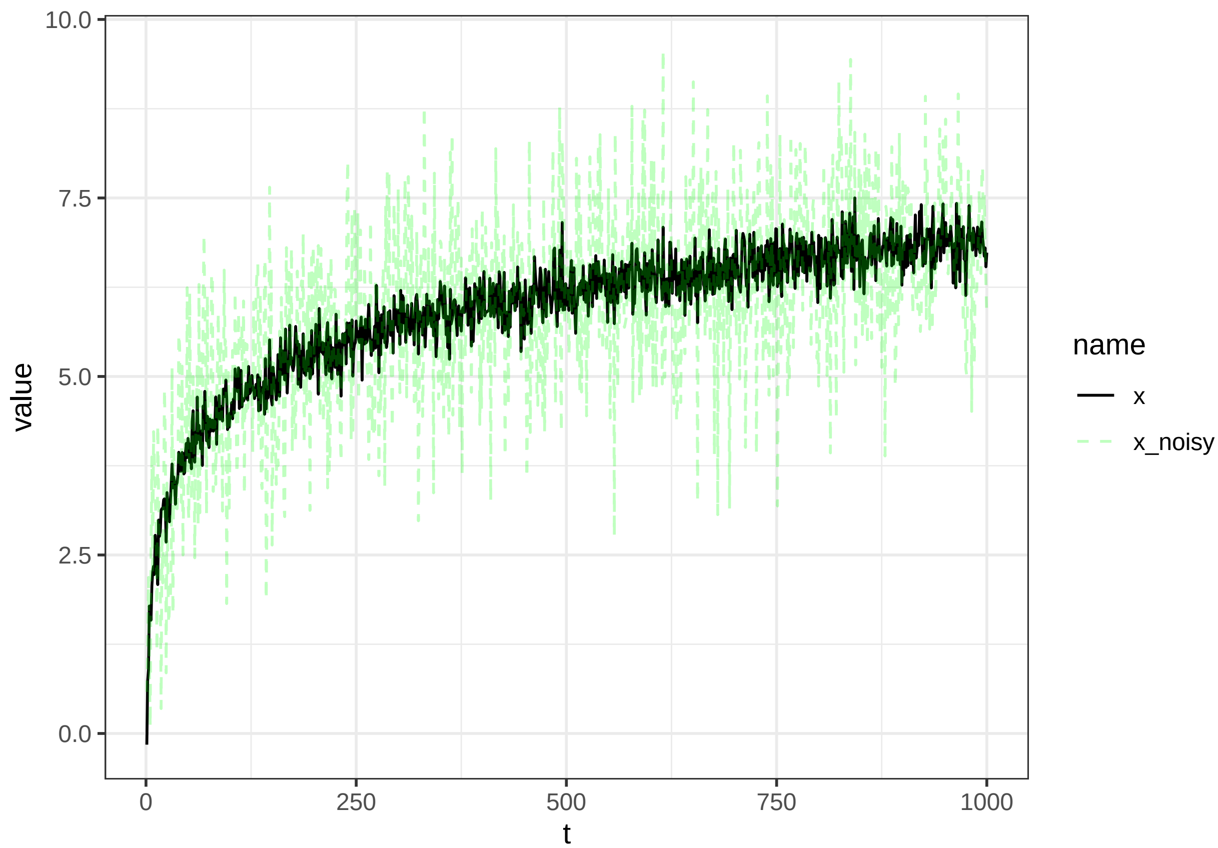
```
true_model <- lm(y ~ x, data=df)

noisy_model <- lm(y ~ x_noisy, data=df)
```

	Dependent variable:	
	y	
	(1)	(2)
Constant	1.889	11.025
	(0.198)***	(0.341)***
x	3.017	
	(0.033)***	
x_noisy		1.467
		(0.056)***
Observations	1,000	1,000
R^2	0.893	0.407
Adjusted R^2	0.893	0.407
Residual Std. Error (df = 998)	1.054	2.484
Note:	*p<0.1; **p<0.05; ***p<0.01	

Visualization

Plot x_t and x_t^{noisy} over time.

```
df_long <- pivot_longer(df, -t) %>% filter(name %in% c('x','x_noisy'))

ggplot(df_long, aes(x=t, y=value, group=name, color=name, alpha=name)) +
  geom_line(aes(linetype=name)) +
  scale_linetype_manual(values=c("solid", "dashed"))+
  scale_color_manual(values=c('black', 'green')) +
  scale_alpha_manual(values=c(1, 0.25)) +
  theme_bw()
```



Filter and Moving Averages

We could apply a filter on x_t^{noisy} to remove some of the "jumpiness". First, we apply a backward-looking moving average with a k-period window:

```
k_param <- 12

df$x_noisy_ma <- stats::filter(x=ts(df$x_noisy),
                               filter=rep(1/k_param, k_param),
                               method="convolution",
                               sides=1)

knitr::kable(head(df, k_param+5))
```

t	y	x	x_noisy	x_noisy_ma
1	2.380203	-0.1566135	0.5825015	NA
2	3.291861	0.7390580	1.1256667	NA
3	5.562697	0.8897051	2.1861023	NA
4	6.414334	1.7851146	0.9815562	NA
5	7.614397	1.6918149	0.0891892	NA
6	6.577953	1.5866424	2.5198933	NA
7	9.095070	2.0677674	3.8738567	NA
8	10.121276	2.2640227	2.2075191	NA
9	8.920044	2.3411699	4.2270812	NA

t	y	x	x_noisy	x_noisy_ma
10	9.293779	2.2262380	3.8046214	NA
11	8.527749	2.7758406	3.2781252	NA
12	9.484397	2.5823675	3.0122817	2.324033
13	8.018776	2.4096392	1.1439937	2.370824
14	8.460624	2.0853824	4.3216147	2.637153
15	10.953855	2.9892829	3.3212513	2.731749
16	10.048865	2.7613553	2.6220729	2.868458
17	10.299179	2.8291658	2.0945720	3.035574

Second, we apply a "centered" moving average that includes both past and future periods.

```
df$x_noisy_ma_ctr <- stats::filter(x=ts(df$x_noisy),
                                   filter=rep(1/k_param, k_param),
                                   method="convolution",
                                   sides=2)

knitr::kable(head(df, k_param+5))
```

t	y	x	x_noisy	x_noisy_ma	x_noisy_ma_ctr
1	2.380203	-0.1566135	0.5825015	NA	NA
2	3.291861	0.7390580	1.1256667	NA	NA
3	5.562697	0.8897051	2.1861023	NA	NA
4	6.414334	1.7851146	0.9815562	NA	NA
5	7.614397	1.6918149	0.0891892	NA	NA
6	6.577953	1.5866424	2.5198933	NA	2.324033
7	9.095070	2.0677674	3.8738567	NA	2.370824
8	10.121276	2.2640227	2.2075191	NA	2.637153
9	8.920044	2.3411699	4.2270812	NA	2.731749
10	9.293779	2.2262380	3.8046214	NA	2.868458
11	8.527749	2.7758406	3.2781252	NA	3.035574
12	9.484397	2.5823675	3.0122817	2.324033	2.854644
13	8.018776	2.4096392	1.1439937	2.370824	2.767397
14	8.460624	2.0853824	4.3216147	2.637153	2.759155
15	10.953855	2.9892829	3.3212513	2.731749	2.684317
16	10.048865	2.7613553	2.6220729	2.868458	2.769787
17	10.299179	2.8291658	2.0945720	3.035574	2.839825

The filters created `NA` values that should be removed before refitting the models.

```
df_filtered <- df %>% drop_na()

knitr::kable(head(df_filtered))
```

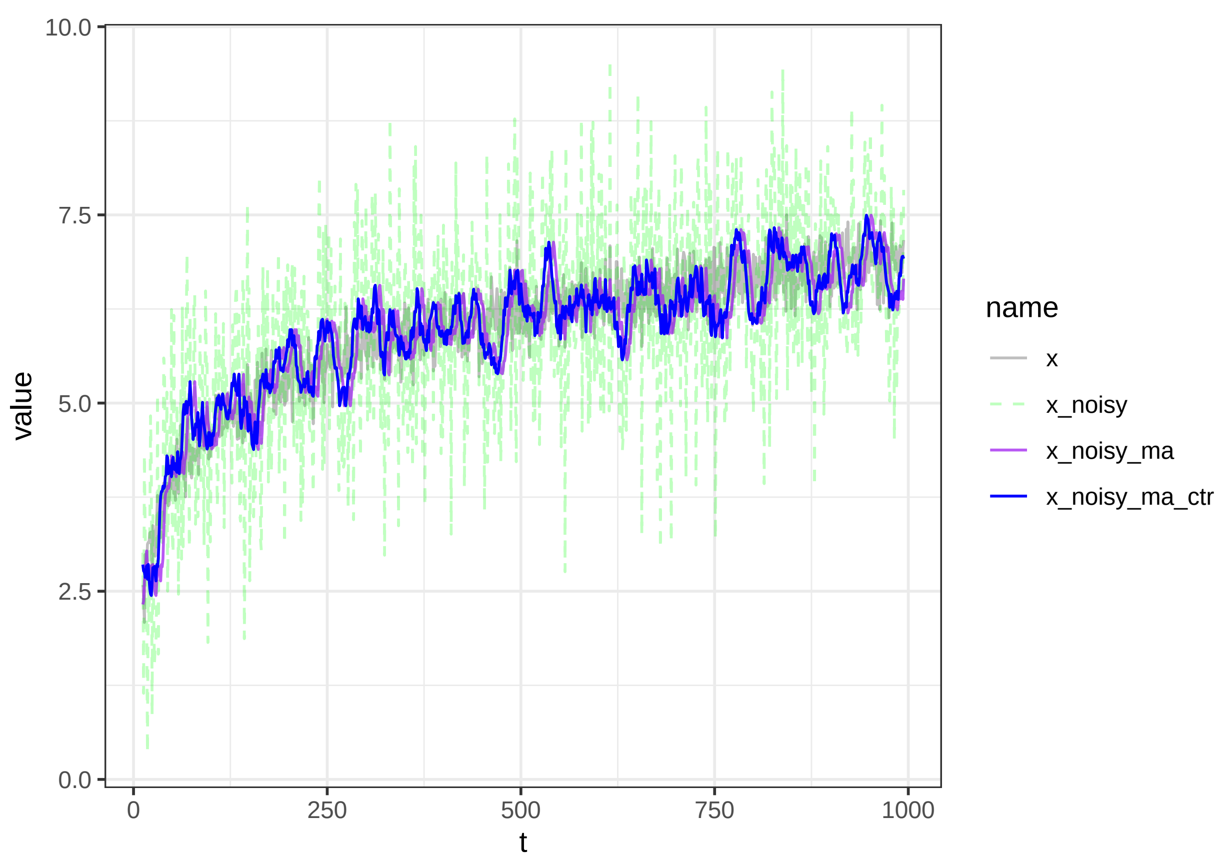
t	y	x	x_noisy	x_noisy_ma	x_noisy_ma_ctr
12	9.484397	2.582368	3.012282	2.324033	2.854644
13	8.018776	2.409639	1.143994	2.370824	2.767397
14	8.460624	2.085382	4.321615	2.637153	2.759155
15	10.953855	2.989283	3.321251	2.731749	2.684317

t	y	x	x_noisy	x_noisy_ma	x_noisy_ma_ctr
16	10.048865	2.761355	2.622073	2.868458	2.769787
17	10.299179	2.829166	2.094572	3.035574	2.839825

More Visualization

Plot x_t, x_t^{noisy}, and the filtered predictors over time.

```
df_long2 <- pivot_longer(df_filtered, -t) %>%
  filter(name %in%  c('x','x_noisy', 'x_noisy_ma', 'x_noisy_ma_ctr'))

ggplot(df_long2, aes(x=t, y=value, group=name, color=name, alpha=name)) +
  geom_line(aes(linetype=name)) +
  scale_linetype_manual(values=c("solid", "dashed", "solid", "solid"))+
  scale_color_manual(values=c('black', 'green', 'purple', 'blue')) +
  scale_alpha_manual(values = c(0.25, 0.25, 0.75, 1)) +
  theme_bw()
```



Refit models with MA predictors

```r
ma_model <- lm(y ~ x_noisy_ma, data=df_filtered)

ma_ctr_model <- lm(y ~ x_noisy_ma_ctr, data=df_filtered)
```

	Dependent variable:			
	y			
	(1)	(2)	(3)	(4)
Constant	1.889	11.025	3.727	3.226
	(0.198)***	(0.341)***	(0.336)***	(0.361)***
x	3.017			
	(0.033)***			
x_noisy		1.467		
		(0.056)***		
x_noisy_ma			2.714	
			(0.056)***	
x_noisy_ma_ctr				2.786
				(0.060)***
Observations	1,000	1,000	983	983
R^2	0.893	0.407	0.706	0.689
Adjusted R^2	0.893	0.407	0.705	0.688
Residual Std. Error	1.054 (df = 998)	2.484 (df = 998)	1.598 (df = 981)	1.644 (df = 981)
Note:				*p<0.1; **p<0.05; ***p<0.01

Optimal Filter via 5-fold CV MSE

The hyper-parameter `k_param` controls the smoothness of the `ma` predictor. We could choose the optimal `k_param` by trying different values and measuring the 5-fold cross-validation error for each value. The code below finds the optimal `k_param` for the backward-looking moving average.

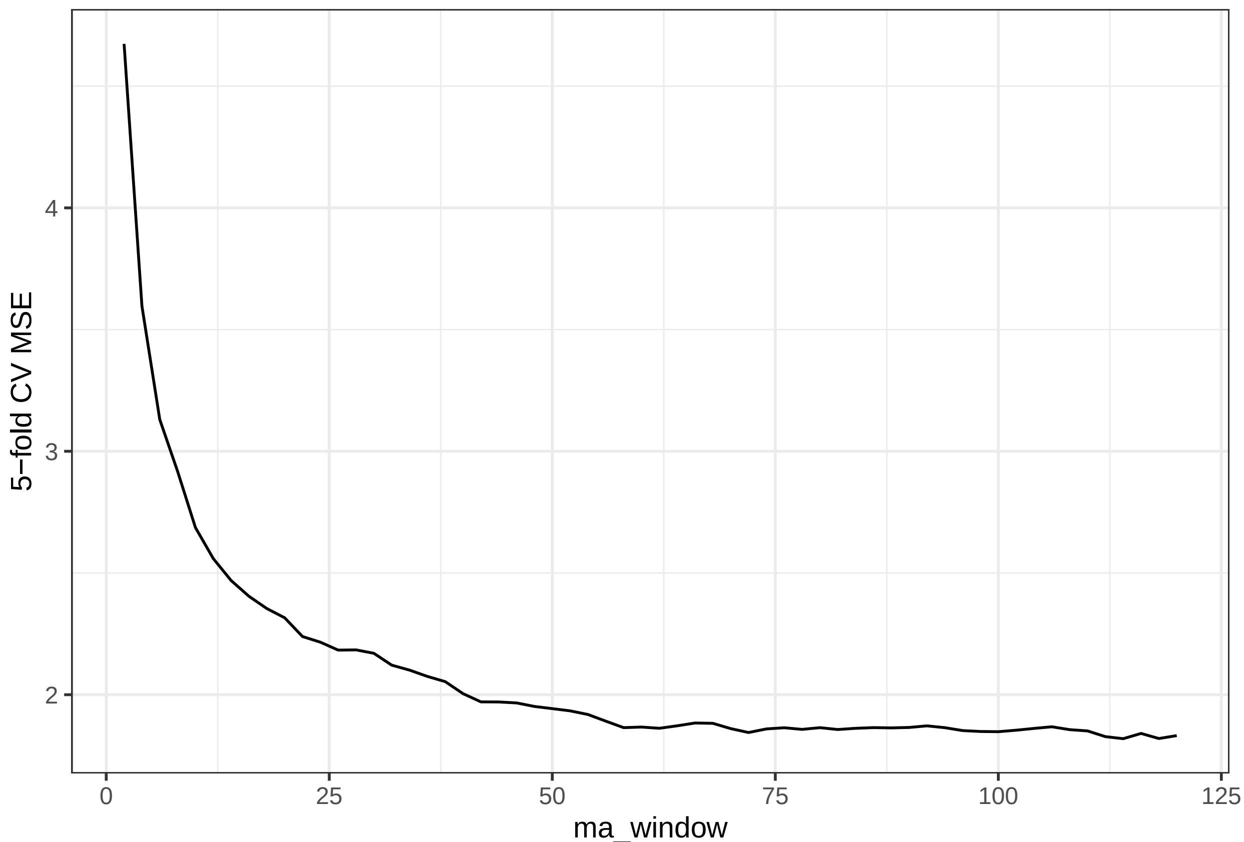
```r
cv_5_fold <- function(dataframe, ma_window){

  dataframe[,'x_noisy_ma'] <- stats::filter(x=ts(dataframe[,'x_noisy']),
                                  filter=rep(1/ma_window, ma_window),
                                  method="convolution",
                                  sides=1)

  train <- dataframe %>% drop_na()

  mod <- glm(y ~ x_noisy_ma, data=train)

  set.seed(123)

  cv_mod <- boot::cv.glm(train, mod, K=5)

  return(data.frame(ma_window=ma_window, MSE=cv_mod$delta[1]))
}
```

Let's try the moving average windows from 2 to 120 (in increments of 2):

```
tuning_ma_window <- lapply(seq(2,120,2), function(x){
  cv_5_fold(dataframe=df, ma_window=x)
})

tuning_ma_window_df <- bind_rows(tuning_ma_window)
```

Plot the window against 5-fold CV MSE:

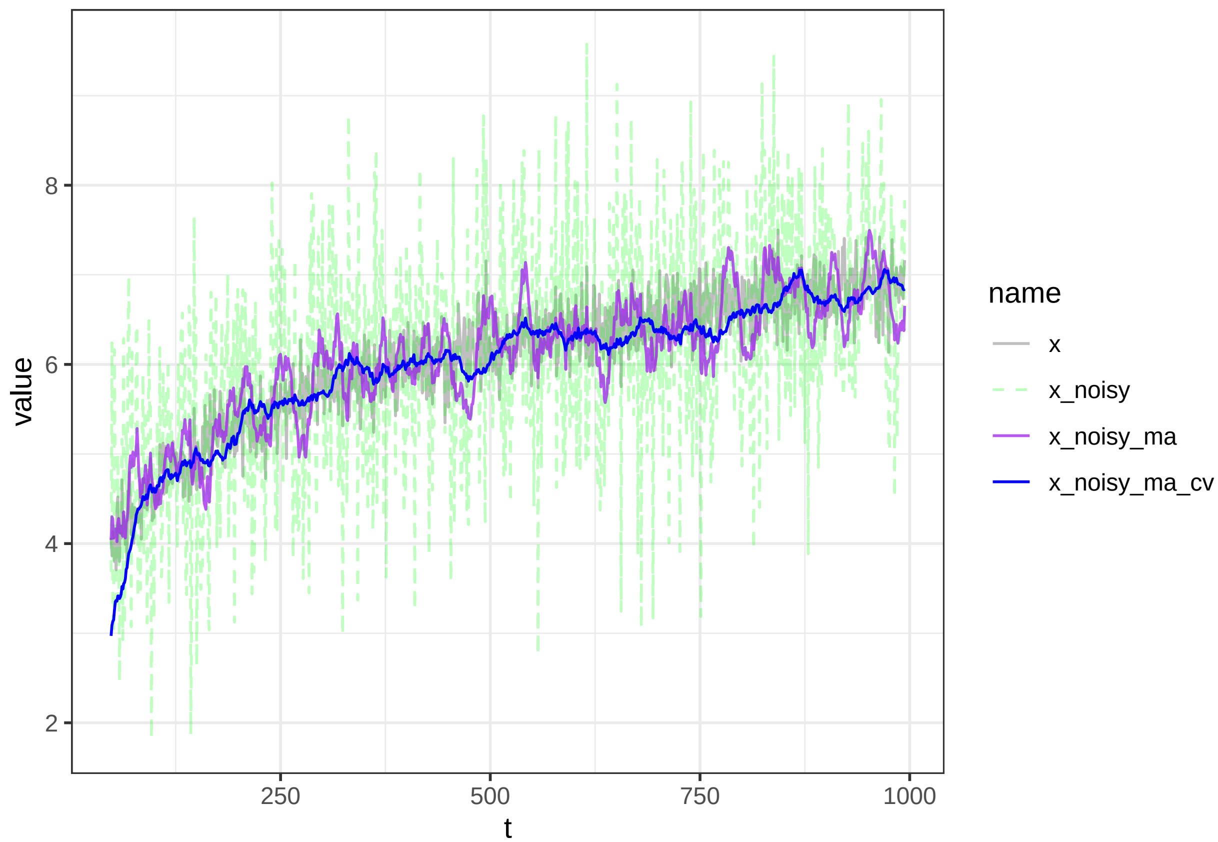
```
ggplot(tuning_ma_window_df, aes(x=ma_window,y=MSE)) +
  geom_line() +
  ylab("5-fold CV MSE") +
  theme_bw()
```



There does not appear to be a meaningful change in CV MSE after `k=48`. Hence, we choose 48 as the optimal moving average window.

```
k_param <- 48

df$x_noisy_ma_cv <- stats::filter(x=ts(df$x_noisy),
                                  filter=rep(1/k_param, k_param),
                                  method="convolution",
                                  sides=1)

df_filtered_cv <- df %>% drop_na()
```

```r
df_long_cv <- pivot_longer(df_filtered_cv, -t) %>%
  filter(name %in%  c('x','x_noisy', 'x_noisy_ma', 'x_noisy_ma_cv'))

ggplot(df_long_cv, aes(x=t, y=value, group=name, color=name, alpha=name)) +
  geom_line(aes(linetype=name)) +
  scale_linetype_manual(values=c("solid", "dashed", "solid", "solid"))+
  scale_color_manual(values=c('black', 'green', 'purple', 'blue')) +
  scale_alpha_manual(values=c(0.25, 0.25, 0.75, 1)) +
  theme_bw()
```




```r
ma_model_cv <- lm(y ~ x_noisy_ma_cv, data=df_filtered_cv)
```

	Dependent variable:			
	y			
	(1)	(2)	(3)	(4)
Constant	1.889	11.025	3.727	3.219
	(0.198)***	(0.341)***	(0.336)***	(0.366)***
x	3.017			
	(0.033)***			
x_noisy		1.467		
		(0.056)***		
x_noisy_ma			2.714	
			(0.056)***	
x_noisy_ma_cv				2.829
				(0.061)***
Observations	1,000	1,000	983	947
R^2	0.893	0.407	0.706	0.697
Adjusted R^2	0.893	0.407	0.705	0.696
Residual Std. Error	1.054 (df = 998)	2.484 (df = 998)	1.598 (df = 981)	1.397 (df = 945)
Note:				*p<0.1; **p<0.05; ***p<0.01

Filter and Exponential Smoothing

Moving averages remove jumpiness in a time series. They are easy to calculate and understand. However, they create missing values in the beginning and/or end of the time series.

Simple exponential smoothing (SES) also removes jumpiness but does not create any missing values.

Hyndman explains SES here and here. The smoothing equation is customized to our example:

$$x_t^{ses} = \alpha x_t^{noisy} + (1-\alpha)x_{t-1}^{ses}$$

If `t=1`, the value of x_0^{ses} is not obvious. Fortunately, the `ets` function estimates both x_0^{ses} and α using maximum likelihood.

```
x_ets <- ets(df$x_noisy, "ANN")

coef(x_ets)
```

```
##      alpha          l
## 0.06554357 2.48370503
```

The estimate of the smoothing parameter (α) is 0.0655436 and the estimate of x_0^{ses} is 2.483705.

Notice that the smoothed values do not contain any `NA`.

```
df$x_noisy_ses <- x_ets$states[-1,1]

knitr::kable(head(df) %>% select(-x_noisy_ma_ctr, -x_noisy_ma))
```

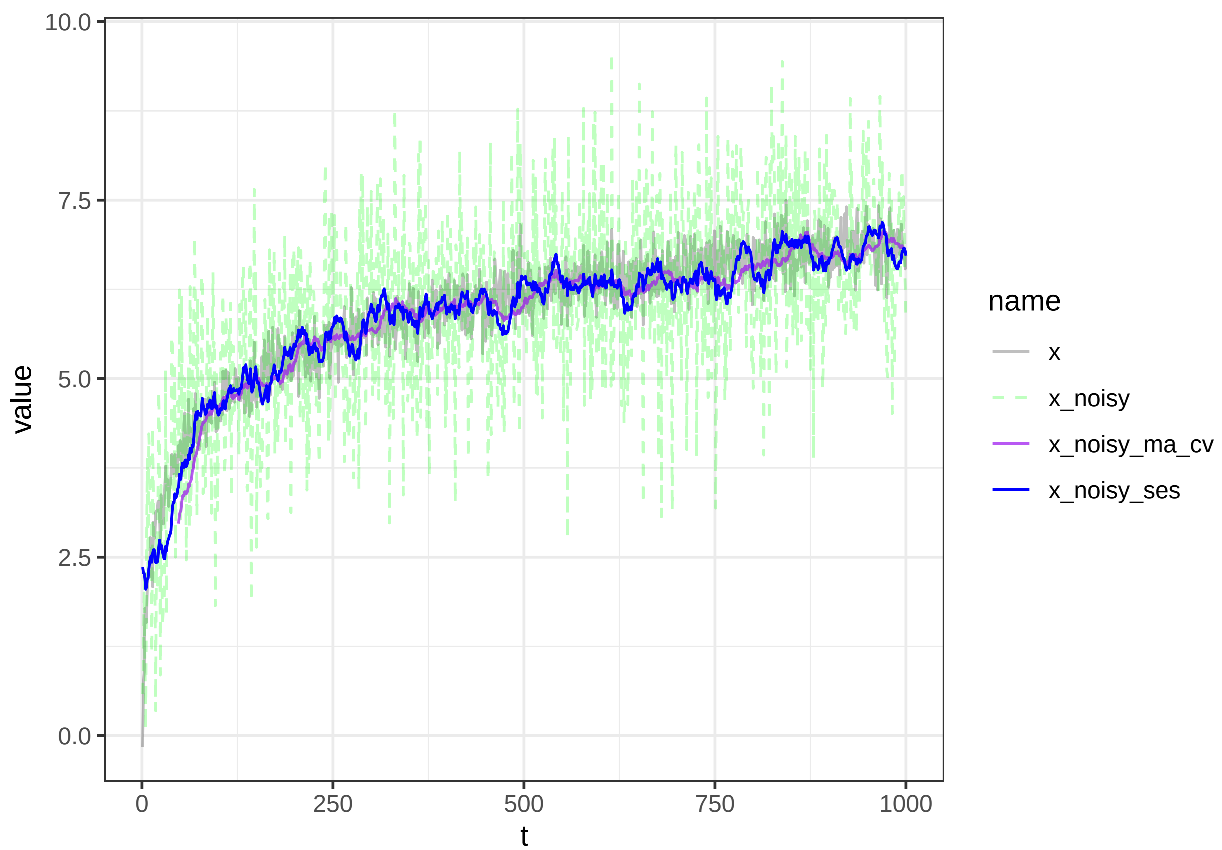
t	y	x	x_noisy	x_noisy_ma_cv	x_noisy_ses
1	2.380203	-0.1566135	0.5825015	NA	2.359093
2	3.291861	0.7390580	1.1256667	NA	2.278250

t	y	x	x_noisy	x_noisy_ma_cv	x_noisy_ses
3	5.562697	0.8897051	2.1861023	NA	2.272210
4	6.414334	1.7851146	0.9815562	NA	2.187616
5	7.614397	1.6918149	0.0891892	NA	2.050078
6	6.577953	1.5866424	2.5198933	NA	2.080871

More Visualization

```
df_long3 <- pivot_longer(df, -t) %>%
  filter(name %in%  c('x','x_noisy', 'x_noisy_ma_cv', 'x_noisy_ses'))

ggplot(df_long3, aes(x=t, y=value, group=name, color=name, alpha=name)) +
  geom_line(aes(linetype=name)) +
  scale_linetype_manual(values=c("solid", "dashed", "solid", "solid"))+
  scale_color_manual(values=c('black', 'green', 'purple', 'blue')) +
  scale_alpha_manual(values=c(0.25, 0.25, 0.75, 1)) +
  theme_bw()
```

```
## Warning: Removed 47 row(s) containing missing values (geom_path).
```



Refit Models

```
ets_model <- lm(y ~ x_noisy_ses, data=df)
```

	Dependent variable:			
	y			
	(1)	(2)	(3)	(4)
Constant	1.889	11.025	3.219	2.823
	$(0.198)^{***}$	$(0.341)^{***}$	$(0.366)^{***}$	$(0.278)^{***}$
x	3.017			
	$(0.033)^{***}$			
x_noisy		1.467		
		$(0.056)^{***}$		
x_noisy_ma_cv			2.829	
			$(0.061)^{***}$	
x_noisy_ses				2.880
				$(0.047)^{***}$
Observations	1,000	1,000	947	1,000
R^2	0.893	0.407	0.697	0.792
Adjusted R^2	0.893	0.407	0.696	0.792
Residual Std. Error	1.054 (df = 998)	2.484 (df = 998)	1.397 (df = 945)	1.470 (df = 998)
Note:				*p<0.1; **p<0.05; ***p<0.01

Optimal Filter via Time Series CV MSE

The `ets` function estimates α and x_0 but ignores the relationship between x_t and y_t. Now we will search through a grid of α values to find the optimal value that best fits several hold out data sets.

Ordinary CV is difficult in this case because:

1. The estimate of α depends on a sequence of observations, so random sampling is not appropriate. Compare this to moving averages where the weights are defined by $\frac{1}{k}$ and there are no parameters to estimate.
2. The hold out data set should not be included in the estimation of α. The smoothed values of x_t^{noisy} in the hold out set should be calculated from the the training estimate of α.

In order to solve these two problems, we use time series cross validation:

1. Partition the training set into 5 sequential slices
2. Each slice contains 150 observations for training and 50 observations for hold out
3. Estimate α for each slice and fit a linear model on the sliced training data
4. Given the estimate of α from step 3, calculate the smoothed predictors, and measure the linear model's hold out MSE
5. Repeat step 3 and 4 for each slice

Here's the function to calculate the smoothed predictors in the hold out data set. For the SES predictor, the last smoothed value from the training set is needed.

```
updateSESvalues <- function(etsobj, newdata){

  lastTrainSmoothVal <- tail(etsobj$states[,1], 1)

  alpha_est <- coef(etsobj)['alpha']

  h <- nrow(newdata)

  for(i in seq.int(h)){
    if(i==1){
      newdata[i,'x_noisy_ses'] = alpha_est*newdata[i,'x_noisy'] + (1-alpha_est)*
        lastTrainSmoothVal

    } else {
      newdata[i,'x_noisy_ses'] = alpha_est*newdata[i,'x_noisy'] + (1-alpha_est)*
        newdata[i-1,'x_noisy_ses']
    }
  }

  return(newdata$x_noisy_ses)
}
```

Create 5 slices of the training set.

```
training_slices <- list(slice1=df[1:200,],
                        slice2=df[201:400,],
                        slice3=df[401:600,],
                        slice4=df[601:800,],
                        slice5=df[801:1000,])
```

A function that does steps 2-4:

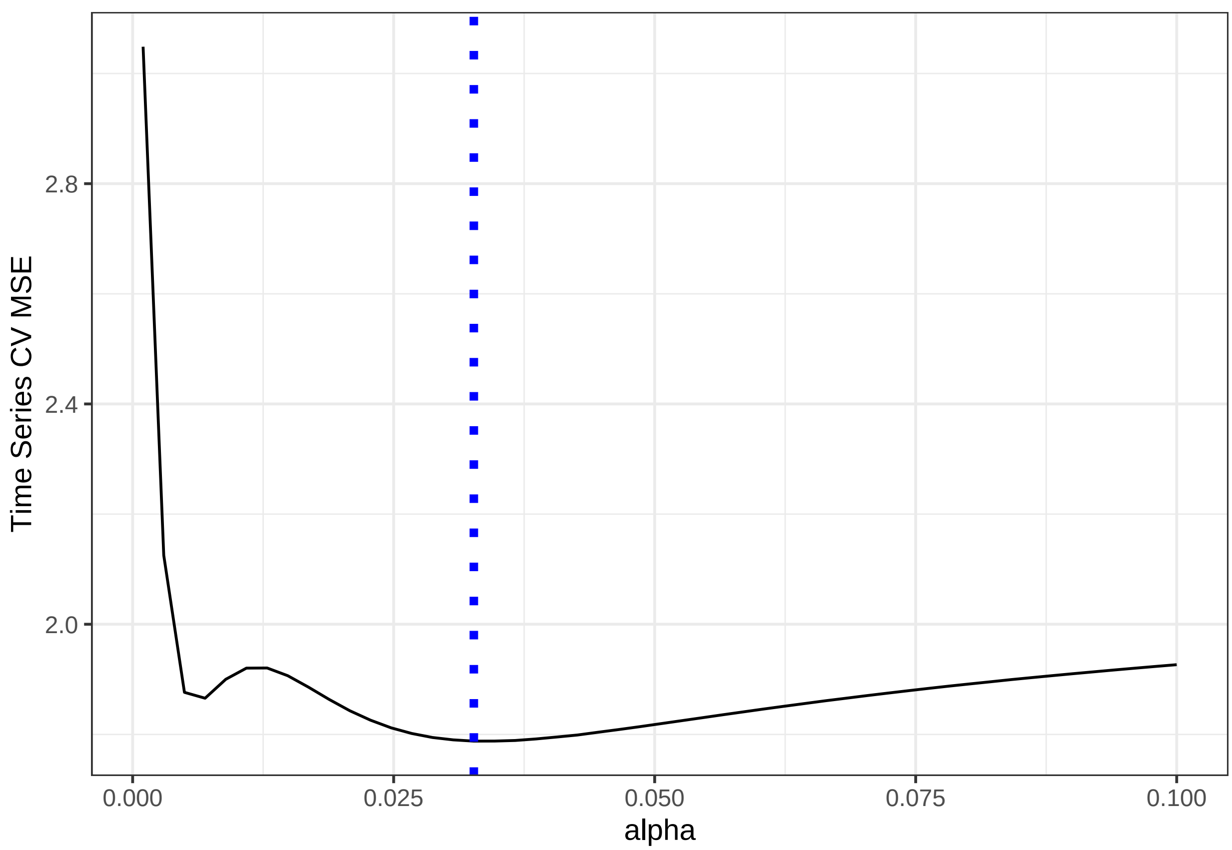
```
doOneSlice <- function(slice_df, alpha){
  train_df <- slice_df[1:150,]
  test_df <- slice_df[151:200,]

  x_ets_slice <- ets(train_df$x_noisy, "ANN", alpha=alpha)

  train_df$x_noisy_ses <- x_ets_slice$states[-1,1]

  lm_ets <- lm(y ~ x_noisy_ses, data=train_df)

  test_df$x_noisy_ses <- updateSESvalues(x_ets_slice, test_df)

  hold_out_preds <- predict(lm_ets, newdata=test_df)

  return(MSE=mse(test_df$y, hold_out_preds))
}
```

A function that runs through all 5 slices and averages the MSE.

```
doAllSlices <- function(listOfSlices, alpha){

  allSlices <- sapply(listOfSlices, doOneSlice, alpha=alpha)

  return(data.frame(alpha=alpha, MSE=mean(allSlices)))
}
```

Try α between 0.001 to 0.10 by increments of 0.00198

```
alphaGridSearch <- lapply(seq(0.001, 0.10, 0.00198), function(x){
  doAllSlices(listOfSlices = training_slices, alpha=x)
})

alphaGridSearchdf <- bind_rows(alphaGridSearch)

bestAlpha <- alphaGridSearchdf[which.min(alphaGridSearchdf$MSE), 'alpha']

ggplot(alphaGridSearchdf, aes(x=alpha,y=MSE)) +
  geom_line() +
  ylab("Time Series CV MSE") +
  theme_bw() + geom_vline(xintercept = bestAlpha, linetype="dotted",
                 color = "blue", size=1.5)
```



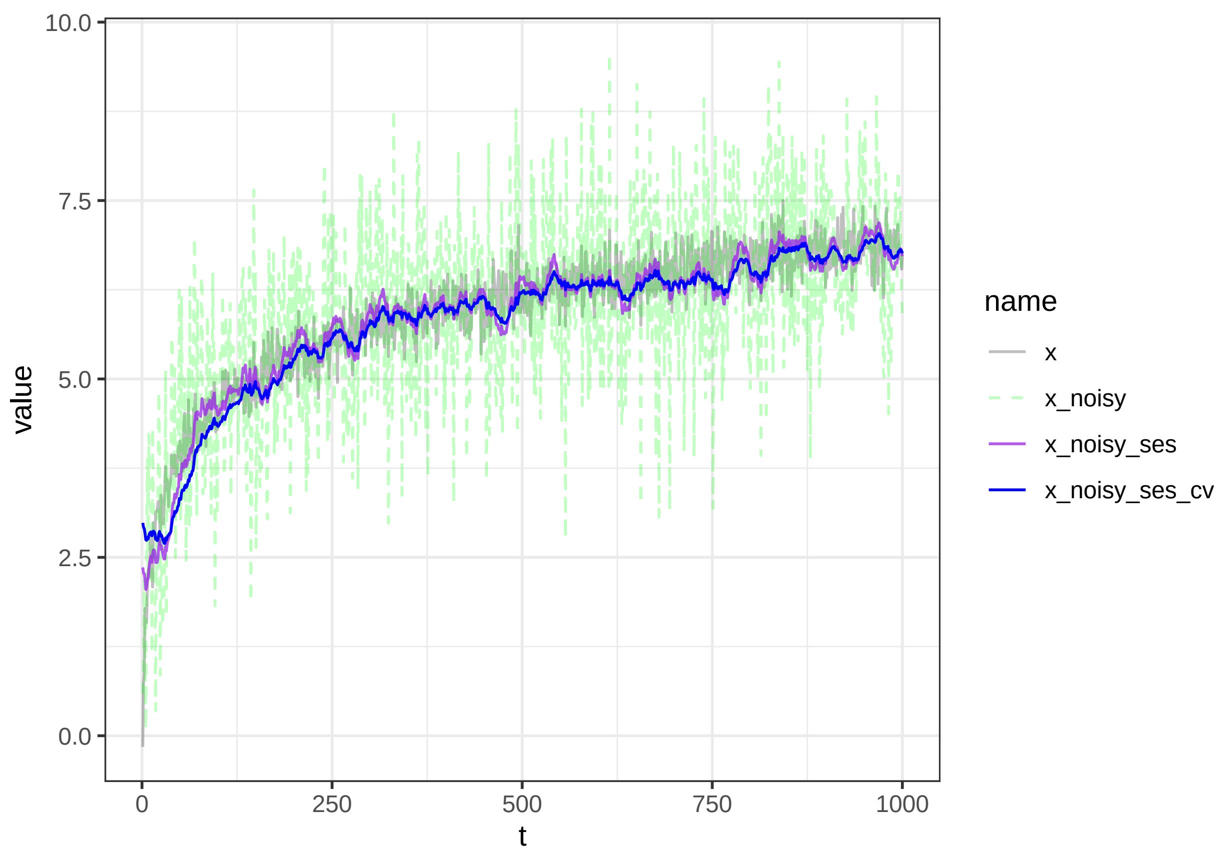
The optimal estimate of α is 0.03268.

Refit Models

```r
x_ets_cv <- ets(df$x_noisy, "ANN", alpha=bestAlpha)

coef(x_ets_cv)
```

```
##        l    alpha
## 3.064214 0.032680
```

```r
df$x_noisy_ses_cv <- x_ets_cv$states[-1,1]

df_long4 <- pivot_longer(df, -t) %>%
  filter(name %in%  c('x','x_noisy', 'x_noisy_ses', 'x_noisy_ses_cv'))

ggplot(df_long4, aes(x=t, y=value, group=name, color=name, alpha=name)) +
  geom_line(aes(linetype=name)) +
  scale_linetype_manual(values=c("solid", "dashed", "solid", "solid"))+
  scale_color_manual(values=c('black', 'green', 'purple', 'blue')) +
  scale_alpha_manual(values=c(0.25, 0.25, 0.75, 1)) +
  theme_bw()
```




```r
ets_model_cv <- lm(y ~ x_noisy_ses_cv, data=df)
```

	Dependent variable:			
	y			
	(1)	(2)	(3)	(4)
Constant	1.889	3.219	2.823	2.937
	(0.198)***	(0.366)***	(0.278)***	(0.276)***
x	3.017			
	(0.033)***			
x_noisy_ma_cv		2.829		
		(0.061)***		
x_noisy_ses			2.880	
			(0.047)***	
x_noisy_ses_cv				2.885
				(0.047)***
Observations	1,000	947	1,000	1,000
R^2	0.893	0.697	0.792	0.793
Adjusted R^2	0.893	0.696	0.792	0.792
Residual Std. Error	1.054 (df = 998)	1.397 (df = 945)	1.470 (df = 998)	1.470 (df = 998)
Note:				*p<0.1; **p<0.05; ***p<0.01

Test Set Performance

Compare the test set performance for the following models:

1. True Predictor Model
2. Noisy Predictor Model
3. Moving Average (CV) Predictor Model
4. Exponential Smoothing (CV) Predictor Model

Predictors for models 3 and 4, need to be computed using the noisy predictor and smoothing parameter estimates.

For the moving average predictor, the last 47 observations from the training set need to be appended to the top of the test set.

```
df_test_2 <- rbind(tail(df[,names(df_test)],47),
                   df_test)

df_test_2$x_noisy_ma_cv <- stats::filter(x=ts(df_test_2$x_noisy),
                                         filter=rep(1/48, 48),
                                         method="convolution",
                                         sides=1)

df_test_2 <- df_test_2 %>% drop_na()
```

For the SES predictor, the last smoothed value from the training set is needed.

```
df_test_2$x_noisy_ses_cv <- updateSESvalues(x_ets_cv, df_test)

knitr::kable(head(df_test_2))
```

t	y	x	x_noisy	x_noisy_ma_cv	x_noisy_ses_cv
1001	21.77200	7.192496	6.057866	6.737078	6.738054
1002	22.88517	7.187736	7.952293	6.768652	6.777735
1003	21.60581	6.693056	7.263767	6.781124	6.793619
1004	23.92071	6.964430	5.612736	6.758142	6.755027
1005	22.19289	6.930092	4.900206	6.708431	6.694412
1006	22.65407	6.498075	7.088554	6.693751	6.707292

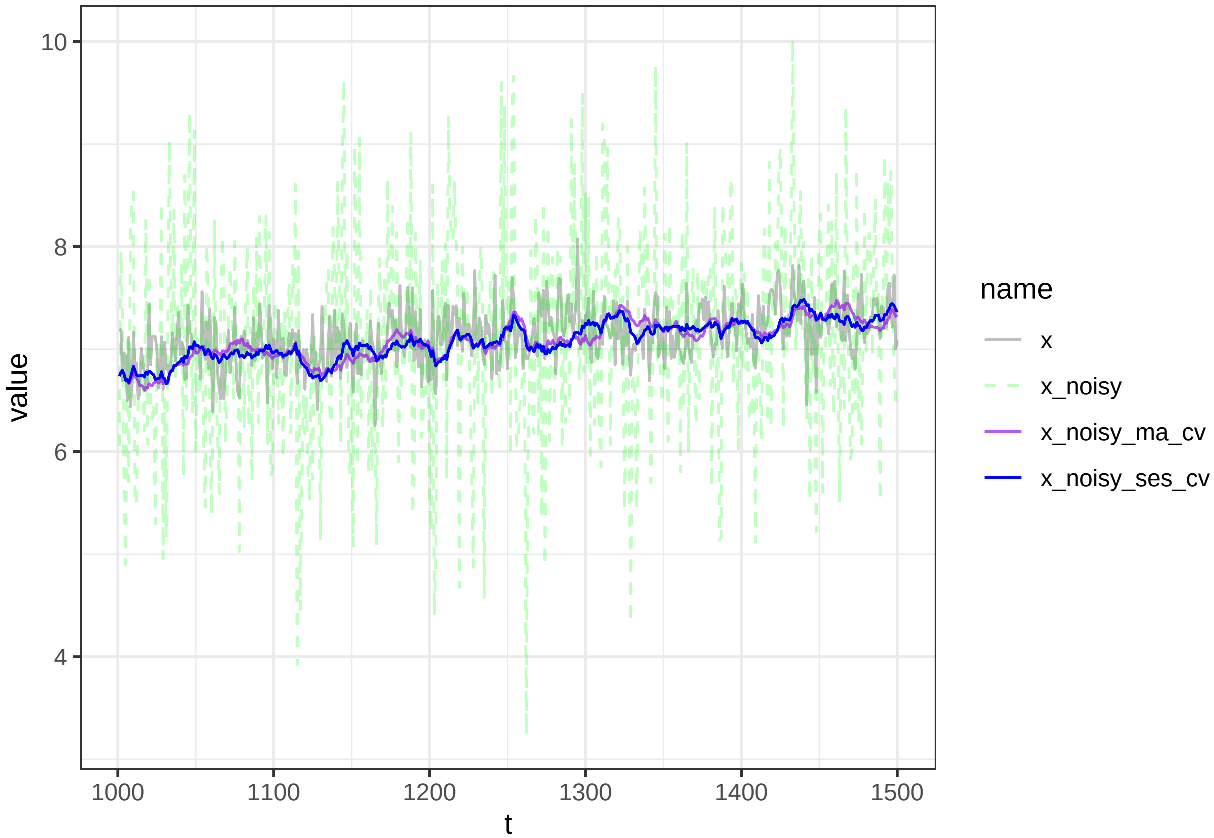
Now we can execute the predict using the 4 regression models:

```
true_x_preds <- predict(true_model, newdata=df_test_2)

noisy_x_preds <- predict(noisy_model, newdata=df_test_2)

ma_x_preds <- predict(ma_model_cv, newdata=df_test_2)

ses_x_preds <- predict(ets_model_cv, newdata=df_test_2)

all_preds <- list(true=true_x_preds,
                  noisy=noisy_x_preds,
                  ma=ma_x_preds,
                  ses=ses_x_preds)

sqrt(sapply(all_preds, function(x){
  mse(actual=df_test_2$y, predicted=x)
}))
```

```
##     true    noisy       ma      ses
## 1.024507 2.596679 1.312104 1.298950
```

Test Set Visualization

```
df_test_3 <- df_test_2

df_long_test <- pivot_longer(df_test_3, -t) %>%
  filter(name %in%  c('x','x_noisy', 'x_noisy_ma_cv', 'x_noisy_ses_cv'))

ggplot(df_long_test, aes(x=t, y=value, group=name, color=name, alpha=name)) +
  geom_line(aes(linetype=name)) +
  scale_linetype_manual(values=c("solid", "dashed", "solid", "solid"))+
  scale_color_manual(values=c('black', 'green', 'purple', 'blue')) +
  scale_alpha_manual(values=c(0.25, 0.25, 0.75, 1)) +
  theme_bw()
```



Conclusion

Filtering a noisy predictor variable (x_t^{noisy}) may improve your time series regression model. Moving averages (MA) and simple exponential smoothing (SES) remove jumpiness from time series. The amount of smoothing should be determined by cross validation.