Moving Averages in R

William Chiu

5/7/2022

Data and Noise

Suppose y is a linear function of x_1 and random error:

$$y = \beta_0 + \beta_1 x_1 + \epsilon$$

```
set.seed(1)

nobs <- 1000

x1 <- log(1:nobs) + rnorm(nobs, 0, 0.25)
y <- 2 + 3 * x1 + rnorm(nobs)

df <- data.frame(t=1:nobs, y=y, x1=x1)

rm(x1, y)
```

But also suppose that x_1 is not directly observable, instead we observe its noiser cousin that contains random errors. For example, the instrument that measures x_1 suffers from random imprecision.

```
df$x1_noisy <- df$x1 + rnorm(nobs,0)
```

Since x_1 is observed with noisy imprecision, this degrades the OLS fit.

```
true_model <- lm(y ~ x1, data=df)

noisy_model <- lm(y ~ x1_noisy, data=df)
```

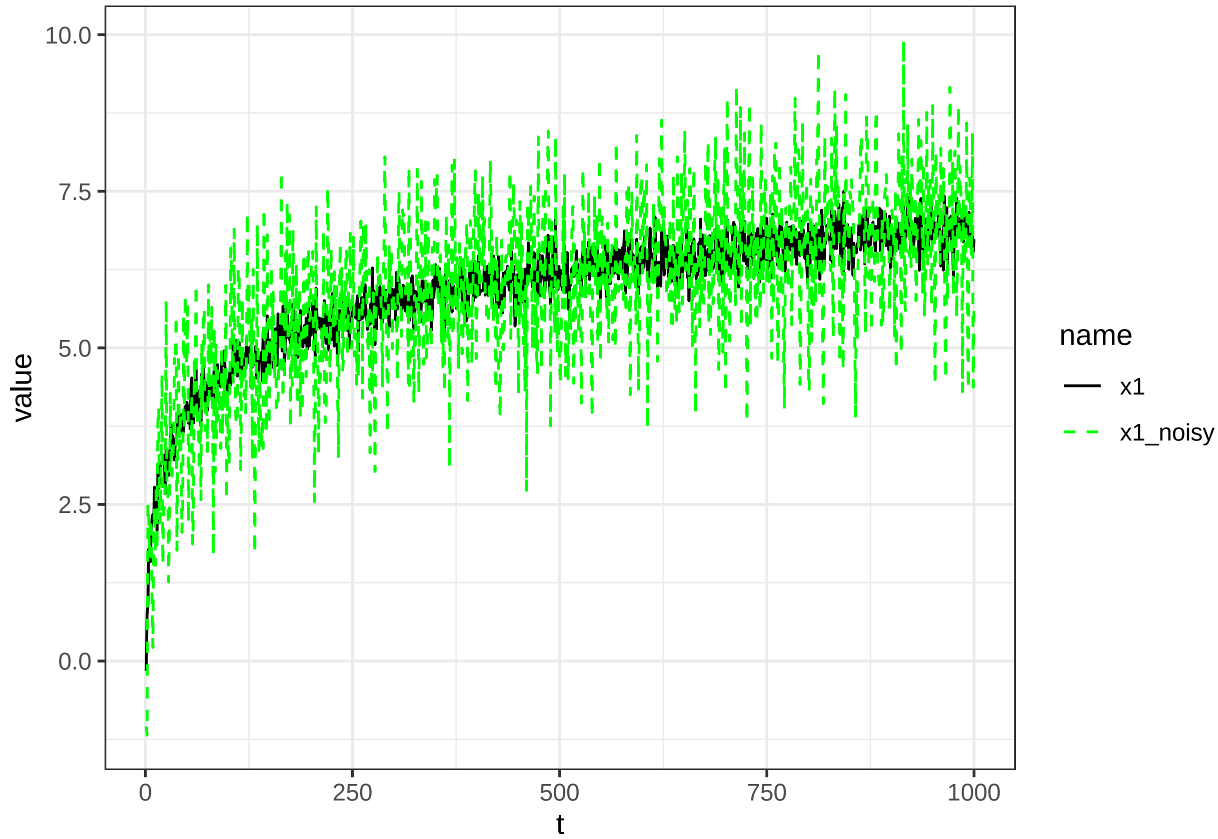
	Dependent variable:	
	y	
	(1)	(2)
Constant	1.935	10.910
	(0.195)***	(0.304)***
x1	3.008	
	(0.033)***	
x1_noisy		1.486
		(0.050)***
Observations	1,000	1,000
R^2	0.895	0.471
Adjusted R^2	0.895	0.471
Residual Std. Error (df = 998)	1.040	2.337
Note:	*p<0.1; **p<0.05; ***p<0.01	

Visualization

Plot x_1 and x_1^{noisy} over time.

```
df_long <- pivot_longer(df, -t) %>% filter(name %in% c('x1','x1_noisy'))

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



Filter and Moving Averages

We could apply a filter on x_1^{noisy} to remove some of the "jumpiness". First, we apply a backward-looking moving average with a k-period window:

```
k_param <- 12

df$x1_noisy_ma <- stats::filter(x=ts(df$x1_noisy),
                                filter=rep(1/k_param, k_param),
                                method="convolution",
                                sides=1)

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

t	y	x1	x1_noisy	x1_noisy_ma
1	2.665125	-0.1566135	-1.0427630	NA
2	5.329106	0.7390580	-1.1831969	NA
3	3.798338	0.8897051	2.5094059	NA
4	7.566075	1.7851146	2.3043845	NA
5	7.144840	1.6918149	1.6359649	NA
6	5.097278	1.5866424	2.2830600	NA
7	9.014142	2.0677674	2.1212831	NA
8	6.879722	2.2640227	0.9537392	NA

t	y	x1	x1_noisy	x1_noisy_ma
9	7.776756	2.3411699	0.2181039	NA
10	9.676868	2.2262380	2.0181594	NA
11	9.786649	2.7758406	2.4630540	NA
12	9.530727	2.5823675	1.5241317	1.317111
13	7.606980	2.4096392	2.8268628	1.639579
14	6.805183	2.0853824	1.7699308	1.885673
15	11.318758	2.9892829	3.8148321	1.994459
16	10.109519	2.7613553	4.0526274	2.140146
17	9.896069	2.8291658	2.2040608	2.187487

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

```
df$x1_noisy_ma_ctr <- stats::filter(x=ts(df$x1_noisy),
                                    filter=rep(1/k_param, k_param),
                                    method="convolution",
                                    sides=2)

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

t	y	x1	x1_noisy	x1_noisy_ma	x1_noisy_ma_ctr
1	2.665125	-0.1566135	-1.0427630	NA	NA
2	5.329106	0.7390580	-1.1831969	NA	NA
3	3.798338	0.8897051	2.5094059	NA	NA
4	7.566075	1.7851146	2.3043845	NA	NA
5	7.144840	1.6918149	1.6359649	NA	NA
6	5.097278	1.5866424	2.2830600	NA	1.317111
7	9.014142	2.0677674	2.1212831	NA	1.639579
8	6.879722	2.2640227	0.9537392	NA	1.885673
9	7.776756	2.3411699	0.2181039	NA	1.994459
10	9.676868	2.2262380	2.0181594	NA	2.140146
11	9.786649	2.7758406	2.4630540	NA	2.187487
12	9.530727	2.5823675	1.5241317	1.317111	2.184831
13	7.606980	2.4096392	2.8268628	1.639579	2.282482
14	6.805183	2.0853824	1.7699308	1.885673	2.583787
15	11.318758	2.9892829	3.8148321	1.994459	2.693905
16	10.109519	2.7613553	4.0526274	2.140146	2.729823
17	9.896069	2.8291658	2.2040608	2.187487	2.727767

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	x1	x1_noisy	x1_noisy_ma	x1_noisy_ma_ctr
12	9.530727	2.582368	1.524132	1.317111	2.184831
13	7.606980	2.409639	2.826863	1.639579	2.282482

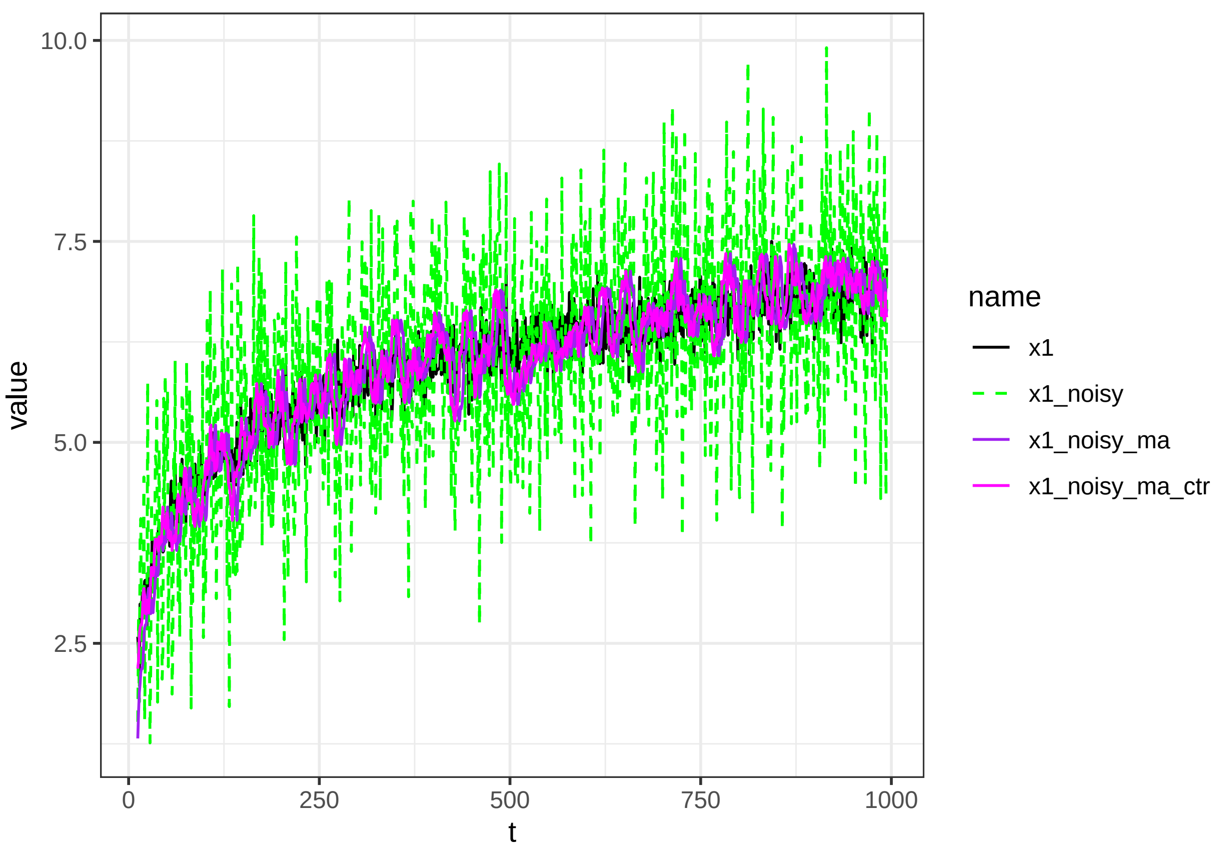
t	y	x1	x1_noisy	x1_noisy_ma	x1_noisy_ma_ctr
14	6.805183	2.085382	1.769931	1.885673	2.583787
15	11.318758	2.989283	3.814832	1.994459	2.693905
16	10.109519	2.761355	4.052627	2.140146	2.729823
17	9.896069	2.829166	2.204061	2.187487	2.727767

More Visualization

Plot x_1, x_1^{noisy}, and the filtered predictors over time.

```
df_long2 <- pivot_longer(df_filtered, -t) %>%
  filter(name %in%  c('x1','x1_noisy', 'x1_noisy_ma', 'x1_noisy_ma_ctr'))

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



Refit models with MA predictors

```
ma_model <- lm(y ~ x1_noisy_ma, data=df_filtered)

ma_ctr_model <- lm(y ~ x1_noisy_ma_ctr, data=df_filtered)
```

	Dependent variable:			
	y			
	(1)	(2)	(3)	(4)
Constant	1.935	10.910	4.930	4.229
	(0.195)***	(0.304)***	(0.285)***	(0.312)***
x1	3.008			
	(0.033)***			
x1_noisy		1.486		
		(0.050)***		
x1_noisy_ma			2.508	
			(0.047)***	
x1_noisy_ma_ctr				2.613
				(0.052)***
Observations	1,000	1,000	983	983
R^2	0.895	0.471	0.741	0.724
Adjusted R^2	0.895	0.471	0.741	0.724
Residual Std. Error	1.040 (df = 998)	2.337 (df = 998)	1.484 (df = 981)	1.533 (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.

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

  dataframe[,'x1_noisy_ma'] <- stats::filter(x=ts(dataframe[,'x1_noisy']),
                                     filter=rep(1/ma_window, ma_window),
                                     method="convolution",
                                     sides=1)

  train <- dataframe %>% drop_na()

  mod <- glm(y ~ x1_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]))
}
```

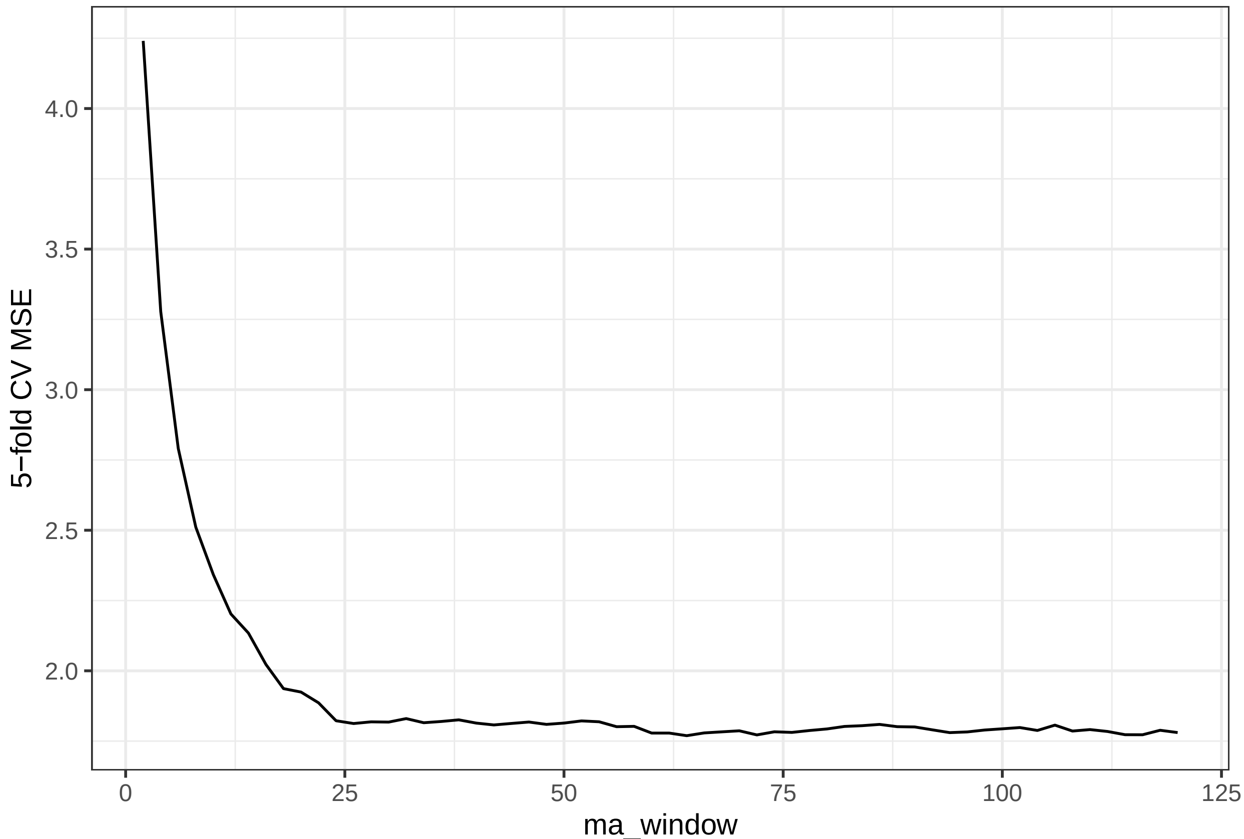
Let's try the moving average windows from 2 to 120 (in increments of 2):

```r
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:

```r
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=24`. Hence, we choose 24 as the optimal moving average window.

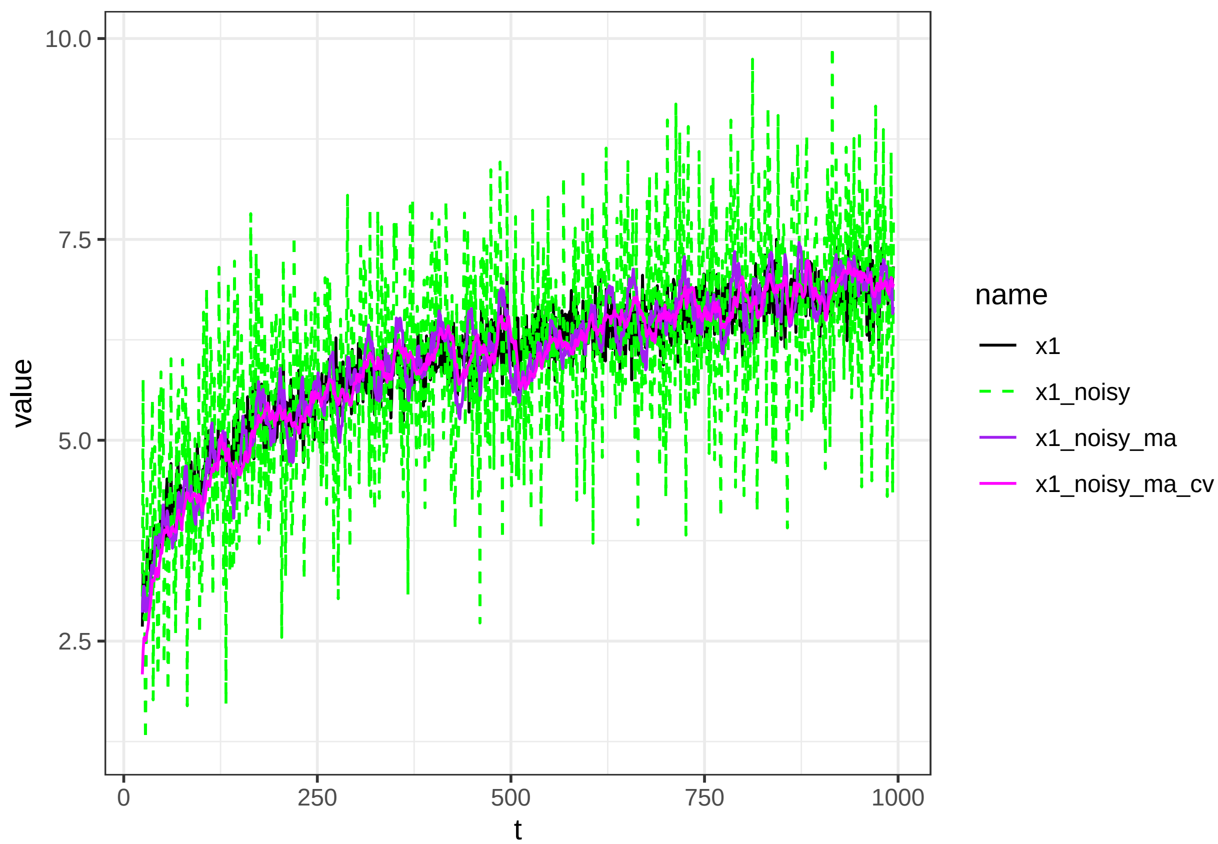
```r
k_param <- 24

df$x1_noisy_ma_cv <- stats::filter(x=ts(df$x1_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('x1','x1_noisy', 'x1_noisy_ma', 'x1_noisy_ma_cv'))

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




```r
ma_model_cv <- lm(y ~ x1_noisy_ma_cv, data=df_filtered_cv)
```

	Dependent variable:			
	y			
	(1)	(2)	(3)	(4)
Constant	1.935	10.910	4.930	4.709
	(0.195)***	(0.304)***	(0.285)***	(0.282)***
x1	3.008			
	(0.033)***			
x1_noisy		1.486		
		(0.050)***		
x1_noisy_ma			2.508	
			(0.047)***	
x1_noisy_ma_cv				2.557
				(0.047)***
Observations	1,000	1,000	983	971
R^2	0.895	0.471	0.741	0.756
Adjusted R^2	0.895	0.471	0.741	0.756
Residual Std. Error	1.040 (df = 998)	2.337 (df = 998)	1.484 (df = 981)	1.350 (df = 969)
Note:				*p<0.1; **p<0.05; ***p<0.01