# Convenience Yield: U.S. Corn

## Background

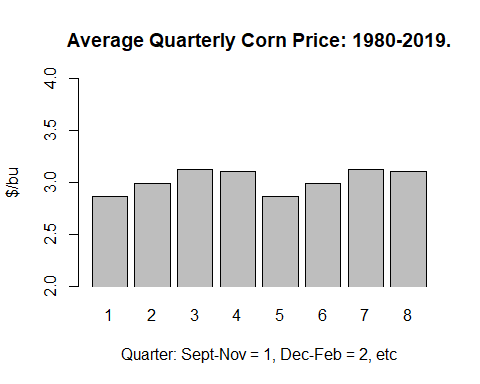
In the previous module we used a simple model which predicts a saw-toothed pattern of prices over time due to a recurring pre-harvest stock-out. Real world commodity markets seldom stock out because pipeline stocks must be carried across marketing years to ensure a continuation of commodity processing. In the U.S. corn market, stocks seldom drop below 10 percent of annual production.

In the previous module the intertemporal law-of-one price (LOP) predicts that price continues to rise over time when stocks are positive. How can we rationalize our observation that real-world stocks are always positive and the LOP which indicates that price must continue to rise over time when stocks are positive?

In this module we introduce the concept of *convenience yield* to address this inconsistency. By adding convenience yield to the simple storage model, the simulated pricing pattern for corn is reasonably close to the real-world corn pricing patterns.

The chart below shows the average quarterly price of U.S. corn over the period 1980 - 2019. The first quarter is the September to November harvesting period, the second quarter is the December to February winter period, etc. Notice that price is lowest in Q1, which is the harvesting quarter, and is highest in Q3, which is the spring quarter. In Q4, which is the pre-harvest quarter, the price begins its decline to the Q5 low point.

historic <- c(2.866, 2.993, 3.123, 3.105, 2.866, 2.993, 3.123, 3.105)  
barplot(historic,names.arg = c(1,2,3,4,5,6,7,8), main="Average Quarterly Corn Price: 1980-2019.",  
 xlab="Quarter: Sept-Nov = 1, Dec-Feb = 2, etc", ylab="$/bu",  
 beside=TRUE, ylim=c(2, 4), xpd = FALSE)



This pricing pattern has the same general shape as the saw-tooth pattern that we saw in the previous section. Specifically, price must rise over time to compensate those merchants who are bearing the cost of storing the corn rather than selling it immediately. The big difference is that price falls as the old crop year finishes and the the new crop year begins, even with positive stocks carried across years.

## Convenience Yield

Imagine in the days prior to debit and credit cards how you would decide how much money you would withdraw from the bank in order to pay for your purchases for the coming week. It is likely that you would take out money than what you expect to use in case you need to make an unexpected purchase. You are willing to give up the interest earnings on the extra amount that you withdraw in order to have the convenience of cash on hand when an unexpected purchase is necessary. The more inconvenient it is to obtain more cash, the greater the convenience yield you will obtain from having the extra cash on hand.

The situation is similar for merchants and processors. They will want to keep more inventory on hand than what they expect to use. The implicit benefit that merchants and processors receive from having inventory on hand rather than having to buy it in the spot market is called *convenience yield*. Convenience yield, which is equivalent to a negative cost of storage, is larger when stocks are scarce and smaller when stocks are plentiful. This makes sense because when stocks are scare it is more costly to search and find stocks in the spot market when an unexpected order comes along.

## Combining Convience Yield and Storage Costs

Our goal is to create an integrated model of storage costs and convenience yield. In the previous module the marginal physical cost of a unit of the stored commodity was assumed to be constant at level , and the capital cost of storage was assumed to equal where is the rate of interest and is the commodity’s price. The rate of interest is currently very low and so we will simplify by assuming . However, the assumption that the unit cost of storage does not depend on aggregate stocks in the market is not very realistic. We expect a higher unit cost storage with higher stocks because congestion will require high-cost storage options to be utilized.

With these additional assumptions, the marginal cost of storage in period can be expressed as where is a measure of aggregate stocks in the market. The marginal convenience yield can be expressed as . The negative slope of this function shows that marginal convenience yield is lower when stocks are larger. The net cost of storage is given by . We subtract because it is a benefit, which is equivalent to a negative cost. We call this net cost of storage the commodity’s *carrying cost*

Similar to the previous module, the intertemporal LOP tells us that when storage is positive then the price increase must equal the carrying cost. A merchant is indifferent between selling the commodity immediately and receiving price , or storing the commodity for one period and then selling it for a net price of . Indifference implies . We can rearrange to obtain a revised LOP expression:

Let and . If we substitute this pair of expressions together with and we obtain the final LOP expression:

## Pricing Dynamics The goal is to generate a pricing pattern similar to that shown in the diagram above. In the Q4 summer quarter the price is falling, which means that . We know that because storage costs are higher and convenience yield is lower when stocks increase. Thus, it must be the case that . In Q1 when stocks are largest and convenience yield is smallest, we see that takes on its largest value, and thus the price increases are the largest. As stocks diminish as we progress from Q1 to Q2 to Q3 the value of remains positive but it is getting smaller and smaller. Thus, the price increases are weakening. By the time we hit Q4 the convenience yield is larger than the storage costs, which means that and the price decreases instead of increases.

The LOP price equation is one of three key equations which governing how prices change over time. The general stock adjustment equation is where is the level of new production (i.e., harvest) in period and is the level of consumption in period . For the case of quarterly data, the full stock adjustment equation can be written as

In the above equation represents starting stocks, represents the level of stocks which are typically carried over from year to year (i.e., long-run carry over) and is a measure of the short run demand for year 2 stocks to be carried into year 3 net of . These three variables, , and , together with the two harvest variables, and , are exogenous in the model. An exogenous variable means that we must attach values to these variables from outside the model rather than calculating equilbrium values from within the model.

The final equation for simulating prices over the eight quarters is the inverse demand for the commodity by the processor. We will use the standard linear demand schedule, . The model has seven LOP equations, eight stock adjustment equations and eight demand equations for a total of 23 equations. There are also 23 variables to be solved for: eight quarterly prices (), eight quarterly consumption levels () and seven ending stocks (). There are only seven stock variables because as you can see above the value of is determined by . After assigning values to the model parameters and the exogenous variables we can solve this system of equations and then recover the equilibrium prices, which we are interested in analyzing.

## Data

We will simulate eight quarterly prices, which means two full years beginning with Q1 in the fall of year 1 and ending with Q8 in the summer of year 2. Data from the USDA Feed Grains Database reveals that average corn harvested acres, yield per harvested acre and beginning stocks for the most recent five crop years (2015/16 - 2019/20) was 82.91 million acres, 173.4 bushels per acre and 2.015 billion bushels, respectively.12 Multiplying the five year average acreage and yield gives five year average production of 14.38 billion bushels. If these estimates are used as the long term average it follows that for the base case. Similarly, if the five year average level of stocks is viewed as normal pipeline stocks it follows that for the base case.

The two parameters from the demand equation are set to and . These parameters ensure that the average price across all eight quarters is $3.628/bu, which is very close to the $3.648/bu average farm gate price for 2016 - 2020. Moreover, the demand elasticity, which is calculated with the simulated $3.628/bu average quarterly price and the simulated 3.595 billion bushel average quarterly consumption, is equal to -0.288. This simulated elasticity is reasonably close to the -0.2 corn demand elasticity estimate which was reported by Moschini et al. [2017].

Data on storage costs and convenience yields are not available and so it is not possible to directly estimate values for m0 and m1. The chosen values, and , are those which achieve a reasonably close match between the seasonal pattern of the simulated prices and real-world prices. More is said about this below.

Let’s formally assign these values to the model parameters and exogenous variables.

a <- 16.21  
b <- 3.50  
m0 <- -0.22  
m1 <- 0.03  
S0 <- 2.015  
H1 <- 14.38  
S\_bar <- 2.015  
v <- c(a, b, m0, m1, S0, H1, S\_bar)

## Simulation

We could tell R to solve the systems of 23 equations and 23 variables and then recover the eight equilibrium quarterly prices. However, to tie in better with the next module it is useful to specify the equilibrium prices as linear functions of and . These two variables are chosen because it is natural to view (i.e., year 2 harvest) as a random variable in Q1 through Q4, and to view (i.e., long term net demand for stocks) as a random variable in all eight quarters. The USDA provides a forecast for both of these variables and the properties of these forecasts will be used when using the model to generate random prices.

The desired equation is

In this equation the (tilde) on the and variables indicate that they are random as described above.

There are eight values for each of , and , which means that we need 24 values to generate the set of eight quarterly prices. The complex set of equations which are required to obtain the 24 values are contained in a R function titled “get\_delta” (this function is stored in file titled “price\_function.R”). We can call this function into our program, generate the 24 delta values and store these values in a matrix titled “del” as follows:

source("price\_function.R")  
del <- get\_delta(v)  
del

## del0 del1 del2  
## [1,] 9.545454 -0.4212615 0.4005162  
## [2,] 9.760180 -0.4248723 0.4039492  
## [3,] 9.919621 -0.4321249 0.4108446  
## [4,] 10.025144 -0.4430814 0.4212615  
## [5,] 10.077655 -0.4578358 0.4352893  
## [6,] 10.077603 -0.4465144 0.4530481  
## [7,] 10.024987 -0.4390203 0.4746902  
## [8,] 9.919357 -0.4352893 0.5004010

The next step is to use these 24 values together with to generate the eight quarterly prices. The following function does the necessary calculations:

get\_price <- function(H5,D) {  
 Price <- del[,1] + del[,2]\*H5 + del[,3]\*D   
}

If we assume and , which implies that year 2 harvest and long term net demand are both normal, then the prices which are generated will equal the base case prices. We first assign these values to and and then call the above pricing function:

H5 <- 14.38  
D <- 0  
P <- get\_price(H5,D)  
P

## [1] 3.487714 3.650515 3.705664 3.653634 3.493977 3.656725 3.711874 3.659897

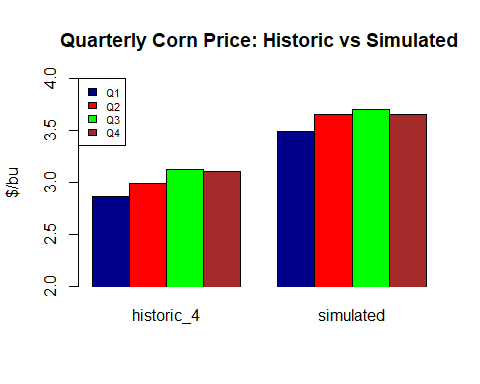
## Comparing Simulated and Real-World Corn Prices

The chart below compares the simulated prices for Q1 through Q4 with the average quarterly corn prices received by farmers for the 1980 - 2019. The similarity of the seasonal pattern in the simulated and real-world prices suggests that despite its simplicity the calibrated model is well suited to analyzing the forward curve for corn.

historic\_4 <- historic[1:4]  
simulated <- P[1:4]  
all\_price <- cbind(historic\_4,simulated)  
rownames(all\_price) <- c("Q1","Q2","Q3","Q4")  
all\_price

## historic\_4 simulated  
## Q1 2.866 3.487714  
## Q2 2.993 3.650515  
## Q3 3.123 3.705664  
## Q4 3.105 3.653634

barplot(all\_price, main="Quarterly Corn Price: Historic vs Simulated",ylab="$/bu",  
 col=c("darkblue","red", "green", "brown"),  
 legend = rownames(all\_price), args.legend = list(x = "topleft", cex = .7), beside=TRUE, ylim=c(2, 4), xpd = FALSE)



It should be obvious from this chart that the simple model we are using does a reasonably good job capturing the seasonality in real-world corn prices. This is especially important when when we examine hedging in a future module.

## Verifying the LOP

It is a good idea to verify that the model is working as intended. First, we should check that the LOP equation, , holds for all eight quarters. Second, we should check that consumption of the inventory across all eight quarters is equal to . In other words, all stocks are fully accounted for. Both of these verifications are conducted in the Appendix of this module.

## What If Analysis

Now that we have the model built we can do “what if” analysis. An obvious “what if” is how does the set of 8 prices respond to a change in the size of the year 2 harvest (e.g., a smaller value for due to fewer planted acres)? For example, suppose , which is below the level of year 1 harvest, . Generating a new set of eight quarterly prices requires using the pricing function, “get\_price”, with the revised value for .

H5 <- 13  
D <- 0  
P\_rev <- get\_price(H5,D)  
P\_rev

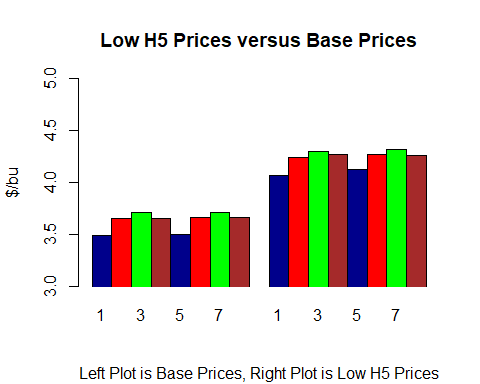
## [1] 4.069054 4.236839 4.301997 4.265086 4.125790 4.272915 4.317722 4.260596

To graph this revised set of prices together with the base set of prices we first combine the two price series into a single matrix and then generate a side-by-side bar chart similar to what what shown in the previous section.

all\_price2 <- cbind(P,P\_rev)  
all\_price2

## P P\_rev  
## [1,] 3.487714 4.069054  
## [2,] 3.650515 4.236839  
## [3,] 3.705664 4.301997  
## [4,] 3.653634 4.265086  
## [5,] 3.493977 4.125790  
## [6,] 3.656725 4.272915  
## [7,] 3.711874 4.317722  
## [8,] 3.659897 4.260596

barplot(all\_price2, main="Low H5 Prices versus Base Prices",ylab="$/bu", sub="Left Plot is Base Prices, Right Plot is Low H5 Prices", names.arg=c(1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8),  
 col=c("darkblue","red", "green", "brown"),  
 beside=TRUE, ylim=c(3, 5), xpd = FALSE)



The chart shows that the lower year 2 harvest raises all eight prices by roughly the same amount. This is an important pricing property of storable commodities. A supply or demand shock in the current year will impact prices in future years by roughly the same amount. This is because merchants are continually shifting the level of stocks being carried through time in order to maximize the average selling price of the commodity.

When prices adjust in response to a supply or demand shock such as the one implied by the previous chart the impact on the eight prices is similar but not identical. This is because the shock will typically change the level of stocks, which in turn affects storage costs and convenient yield. The pricing impacts are calculated as follows:

impacts <- P\_rev - P  
impacts

## [1] 0.5813409 0.5863238 0.5963324 0.6114524 0.6318134 0.6161899 0.6058481  
## [8] 0.6006992

## Appendix

The purpose of this Appendix is to demonstrate the LOP and the stock adjustment equation hold for the base case set of eight similated prices. The verification begins by substituting the equilibrium price into the inverse demand schedule to obtain the quarterly consumption, , Then substitute quarterly consumption into the stock adjustment equation to derive the quarterly stock levels, . Finally, substitute the derived values of and the equilibrium prices into the LOP equation, , to verify that the equation holds.

H5 <- 14.38  
D <- 0  
X <- a/b - 1/b\*P  
S <- rep(0, times = 8)  
S[1] <- S0 + H1 - X[1]  
S[2] <- S[1] - X[2]  
S[3] <- S[2] - X[3]  
S[4] <- S[3] - X[4]  
S[5] <- S[4] + H5 - X[5]  
S[6] <- S[5] - X[6]  
S[7] <- S[6] - X[7]  
S[8] <- S[7] - X[8]  
S

## [1] 12.760061 9.171637 5.598970 2.011436 12.758287 9.171637 5.600744  
## [8] 2.015000

lop <- rep(0, times = 8)  
for(i in 1:7)  
{  
 lop[i+1] <- P[i+1]-P[i]-(m0+m1\*S[i])  
}  
lop

## [1] 0.000000e+00 -3.247402e-15 -1.193490e-15 3.635980e-15 -4.996004e-15  
## [6] 8.049117e-16 5.828671e-16 2.137179e-15