# Consumers' model

This model is taken from the paper: Smallwood and Conlisk, 1979, *Product Quality in Markets where Consumers are Imperfectly Informed*, Quarterly Journal of Economics, 93-1.

Consumers are, in the most relevant markets, no expert about the quality of the products they buy. However, people try to find some clues about which is the best product. A typical rule is to trust more products that are more common among other consumers, as if the popularity of a product could signal its quality.

Is it justified this assumption? Can consumers, unable individually to evaluate qualities, assess them collectively?

## Consumers' model

The model makes extreme (unrealistic) assumptions in provide a clear answer. In particular:

- the price is identical for all products;
- no differentiation in distribution;
- no advertisment;
- no access to experts' information.

The model simply implement products/firms with different (hidden) qualities, and consumers retain their currently owned product if it works, or buy a new one if not. To choose a product, consumer choose randomly with probabilities proportional to the current market shares.

## Consumers' model

Let's see the main equations. Consumers keep the current product if it keeps on working, and choose a new product if the product is broken:

$$\text{Current Product} = \begin{cases} \text{Current product} & \text{, if it is not broken} \\ \text{New product} & \text{, if breaks down} \end{cases}$$

We consider that each product when used has a probability of "breaking down", meaning it does not satisfy any longer the consumer.

$$\text{Broken} = \begin{cases} Yes & \text{with probability} = BD \\ No & \text{with probability} = 1 - BD \end{cases}$$

## Consumers' model

When a consumer needs to buy a product, she does not know the probabilities BD (i.e. the only quality that matters). But the consumer can observe the other consumers' products. We assume that the consumer can choose any of the products currently available, with probabilities proportional to the *shares* of consumers currently owning the products.

$$p_i = \frac{ms_i^\alpha}{\sum_{j=1}^{n} ms_j^\alpha}$$

## Consumers' model

The value of $\alpha$ affects the concentration of probabilities in favor of the larger firms. For example:

- $\alpha$=0: all $p_i = 1$, irrespective of market shares
- $\alpha \uparrow$: larger firms have increasingly higher probabilities than smaller ones
- $\alpha \simeq \infty$: the largest firm has $p_i \simeq 1$ and all others, even close, have $p_j \simeq 0$

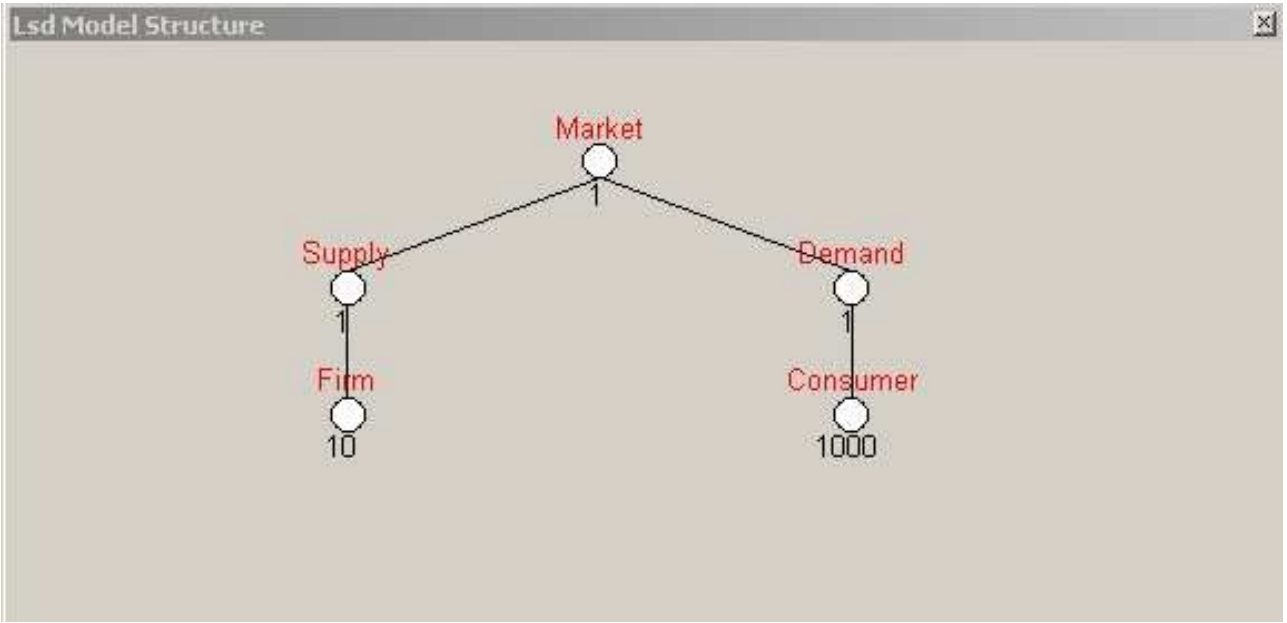
Thus, we may take $\alpha$ as a proxy for the trust consumers have of the market to select the best product, i.e. assuming that the best selling product has better quality.

## Consumers' model

Let's start a new model as usual. Use **LMM / Browse Models** to create a new model called **Consumer Model** in directory **SC**. Firstly, we consider the model structure, without implementing it, just to "place" the equations in the different objects. Then we look at all the equations, understanding what they do. After the implementation of the equations we will compile the model program and, with that, and create the model configuration.

# Model structure

The structure of the model is straightforward, made by **Supply** containing **Firm**s and **Demand** with **Consumer**s.



## Consumers' model

We assign to each firm a parameter, **IdFirm**, taking different values for each firm, and obviously a variable of market share. Since we are dealing with a semi-durable market, there exist two distinct types of market shares:

- **ms_sales**: sales of the firm over total sales.
- **ms_users**: number of users over total number of users.

The second index is clearly better to compute the probabilities $p_i$ which, essentially, refers to the visibility of different brands, and consumers are more likely to monitor shares of owned products, not of sales.

## Consumers' model

We create a variable indicating the visibility of a product according to the market shares and the trust of consumers, indicated by $\alpha$:

$$Visibility_t = ms\_user_t^{\alpha}$$

## Consumers' model

We then define a variable **NumUsers** for each firm, whose equation is:

$$NumUsers_t = NumUsers_{t-1} + Sales_t - NumLost_t$$

where **Sales** and **NumLost** indicate respectively the new consumers purchasing a product, and previous users abandoning the product.

## Consumers' model

**Sales** and **NumLost** are obviously variables, since they change at each time step. However, their equation is difficult to design, since they depend on the decisions of a selected group of consumers.

Thus, we define them technically as LSD parameters, and then we will ensure that consumers' action update their value as necessary:

- consumers with a broken product update **NumLost** that sold the broken product;
- consumers purchasing a product increase the **Sales** of the firm selling the new product.

## Consumers' model

Variable **ProdUsed** indicates the product used by the consumer, indicated by the **IdFirm** of the firm who sold it.

```
EQUATION("ProdUsed")
/*
Determine the product used by the consumer at each time step
*/
v[0]=V("IsBroken"); //breaks ?
if(v[0]==1)
  v[1]=V("Purchase"); //yes, buy a new produc
else
  v[1]=VL("ProdUsed",1); //no, keep on using the previous one
RESULT(v[1])
```

# Consumers' model

Let's start by analyzing the code for **Purchase**, which should return the id of a firm chosen proportionally to **Visibility**.

```
EQUATION("Purchase")
/*
Make a purchase for the calling object (supposedly a consumer).
*/
cur1=RNDDRAW("Firm","Visibility");  //choose randomly one of the products
v[0]=VS(cur1,"IdFirm"); //return the ID of the chosen firm
V("InitTrade"); //ensure that firms are ready to sell
INCRS(cur1,"Sales",1); //increase the Sales of the chosen firm
RESULT(v[0] )
```

## Consumers' model

The LSD command `RNDDRAW("Firm","Visibility")` returns an object with the label **Firm** with probabilities proportional to the normalized values of **Visibility**.

The equation then asks for the **IdFirm** of the chosen firm, which is returned as result of the equation.

Moreover, the code also uses `INCRS(cur1,"Sales",1);`, which changes the value of **Sales** contained in object `cur1` adding 1 to the previous value.

## Consumers' model

The command `RNDDRAW(...)` (like `CYCLE...`) must necessarily be located in an object containing the objects to analyze, in our case **Firm**. Therefore, the element **Purchase** must be located necessarily in **Supply**, containing the firms of the model.

## Consumers' model

Moreover, **Purchase** cannot be a variable: it is a single copy, but all consumers making a purchase need a freshly computed value from **Purchase**. If it were a variable, its equation would be computed once at a time step, and all subsequent consumers asking for a value would obtain the same result.

Thus, **Purchase** must be defined as a function placed in **Supply**.

## Consumers' model

The computation `V("InitTrade");` does not produce any value, but it serves as *semaphore*. The following line increases the sales of the chosen firm, but it risks adding 1 to the sales from the previous time step.

We want to ensure that the level of **Sales** is reset to 0 *before* the first purchase takes place, and then cumulate all sales.

## Consumers' model

The following equation produces this result. Defined as a variable, **InitTrade** placed in **Supply** is computed once and only once at each $t$. Being called before any sale is registered, it ensures that cumulated sales are not mistakenly transferred from one period to the following.

```
EQUATION("InitTrade")
/*
Initialize the trading period. For each firm set to 0 the
parameter Sales and NumLost
*/
CYCLE(cur, "Firm")
 { //for all firms set to 0 Sales and NumLost
  WRITES(cur,"Sales",0);
  WRITES(cur,"NumLost",0);
 }
RESULT(1 )
```

## Consumers' model

**IsBroken** needs to use the probability of breaking down from the firm that sold the product currently used by the consumer, and drawing a random event with that probability. It will then return 0 for not broken and 1 for broken.

# IsBroken

```
EQUATION("IsBroken")
/*
Check whether the product breaks down or not.
Return 1 if the product breaks or 0 if not.
*/
v[0]=VL("ProdUsed",1); //product owned by the consumer at t-1
cur=SEARCH_CND("IdFirm",v[0]); //search the object with IdFirm
v[2]=VS(cur,"BD"); //read the value of the prob. to break down
if(RND<v[2]) //
 {v[1]=1; //product broken
  V("InitTrade"); //just to ensure NumLost is reset
  INCRS(cur,"NumLost",1);
 }
else
 v[1]=0; //product not broken
RESULT(v[1] )
```

## Consumers' model

The command `SEARCH_CND("IdFirm",v[0])` scans the whole model searching for the copy of **IdFirm** having exactly the value contained in `v[0]`.

`RND` is a (pseudo-)random variable returning a value in (0;1) with uniform probability. The IF-ELSE construct ensures the desired result.

Notice that the repetition of call to **InitTrade** is not a problem. Being a variable is always computed no more than once at every time step.

## Consumers' model

We can now write the equation for **NumUsers**. It needs to use **Sales** and **NumLost** updated by consumers actions. The only problem is to ensure that all consumers made their purchase, because if **NumUsers** is computed before some consumers acted the value in **NumUsers** would be wrong. Thus we use another semaphore.

```
EQUATION("NumUsers")
/*
Number of users, computed, after the end of the trading period,
by summing to the previous users the new sales and
removing the lost users
*/
V("EndTrade"); //ensure that buyers finished the shopping
v[0]=VL("NumUsers",1); //former number of users
v[1]=V("Sales"); //sales at this time
v[2]=V("NumLost"); //lost users at this time
v[4]=v[0]+v[1]-v[2];
RESULT(v[4] )
```

## Consumers' model

The semaphore **EndTrade** must be located in **Demand** because it cycles through all consumers.

```
EQUATION("EndTrade")
/*
For each consumer ensures that the variable ProdUsed
is updated so that Sales and NumLost are filled with
 the correct values
*/
CYCLE(cur, "Consumer")
 VS(cur,"ProdUsed");
RESULT( 1)
```

## Consumers' model

The equation for **ms_users** and **Visibility** are trivial.

```
EQUATION("ms_user")
/*
Market shares of users, computed as the ratio
of users over the total number of users
*/
v[0]=V("TotalUsers");
v[1]=V("NumUsers");
RESULT(v[1]/v[0] )
```

# Consumers' model

```
EQUATION("Visibility")
/*
Visibility, implemented to avoid the math error
of a power of 0.
*/
v[0]=V("alpha");
v[1]=VL("ms_user",1);
if(v[1]==0)
 v[2]=0;
else
 v[2]=pow(v[1],v[0]);
RESULT(v[2])
```

Notice that we use past shares to avoid a dead-lock error.

## Consumers' model

The model is completed and functioning for the generic time step $t$ on the basis of the correct values at $t-1$. In particular, it is necessary that at any moment there are exactly **NumUsers** number of consumers having **ProdUsed** set to the id of any firm.

How can we ensure this condition is respected at the very first time step t=1? We can write an initialization function ensuring the coherence of the model at $t = 1$. The variable must be placed in **Market** to ensure its working, as it needs to affect both firms and consumers.

# Consumers' model

```
EQUATION("Init")
/*
Initialize demand. It assumes that initially there
is only 1 consumer. ms_users[0] are the prob of first purchase
*/
cur=SEARCH("Demand");//search the object Demand
v[0]=V("TotalUsers");//number of consumers
ADDNOBJS(cur,"Consumer",v[0]-1);//create new objects
CYCLES(cur,cur1, "Consumer")
 {
  v[1]=V("Purchase");
  WRITELS(cur1,"ProdUsed",v[1], t-1);//replace the existing value
 }
PARAMETER //transform the variable into a param
RESULT(1)
```

## Consumers' model

The command `SEARCH("Demand")` returns the object indicated to a pointer.

The command `ADDNOBJS(cur,"Consumer",v[0]-1)` creates `v[0]-1` new copies of **Consumer** placing them in `cur`.

The command `WRITELS(cur1,"ProdUsed",v[1], t-1)` overwrites the content of **ProdUsed** with `v[1]` contained into `cur1`, cheating the system to believe that the imposed value was the result of a computation made at time $t-1$.

`PARAMETER` effectively prevents the variable **Init** from being computed again, turning it into a parameter which are never computed .

## Consumers' model

Now we have the equations ready. Implement and compile the model and fix the possible compilation errors.

When the Lsd model program runs, create the model structure, composed by following elements and initializations.

## Model Structure

- Object **Market** contains: variable **Init**; parameter **TotalUsers**; objects **Demand**, **Supply**
- Object **Supply** contains: variable **InitTrade**; function **Purchase**; parameter **alpha**; object **Firm**.
- Object **Firm** contains: parameters **IdFirm**, **BD**, **Sales**, **NumLost**; variables **NumUsers**(1), **ms_users**(1), **Visibility**.
- Object **Demand** contains: variable **EndTrade**; object **Consumer**.
- Object **Consumer** contains: variable **ProdUsed**(1), function **IsBroken**

## Model Initialization

- Object **Market** (1 copy): **TotalUsers**=10000
- Object **Supply** (1 copy): **alpha**=0.5
- Object **Firm** (10 copies): **IdFirm**=1,2,3,...; **BD**=0.10, 0.11, 0.12, ...; **Sales**=0, **NumLost**=0; **NumUsers**=0, **ms_users**=0.1
- Object **Demand** (1 copy).
- Object **Consumer** (1 copy): variable **ProdUsed**=0

## Simulation settings

Use the following simulation settings:

- Num. Simulations = 1;
- Initial Seed = 1
- Simulation Step = 500
- Insert debugger at = 0
- Mark the **Save** option for **ms_user**, **NumUsers**, **Sales**, **NumLost**:

## Consumers' model

Do you have any expectation on what will happen? And why?

Run the simulation and investigate the model's behaviour. Test different seeds and different values for **alpha**.

Plot the series of **ms_users** and explain their behavior for various values of **alpha**

## Optimization

LSD models can be optimized to reduce the steps the system has to take to fulfill commands and the number of elements.

For example, **IsBroken** is present in 10,000 copies, but it contains no data worth saving. It can be turned into a function placed in **Supply**, so each consumer can use its code.

But doing so would not work. The problem is that the code for **IsBroken** would not be able to access the data from the consumer asking for its computation. Here is the solution.

# IsBroken

```
EQUATION("IsBroken")
/*
Check whether the product breaks down or not.
Return 1 if the product breaks or 0 if not.
*/
v[0]=VLS(c,"ProdUsed",1); //product owned by the consumer at t-1
cur=SEARCH_CND("IdFirm",v[0]); //search the object with IdFirm
v[2]=VS(cur,"BD"); //read the value of the prob. to break down
if(RND<v[2]) //
 {v[1]=1; //product broken
  V("InitTrade"); //just to ensure NumLost is reset
  INCRS(cur,"NumLost",1);
 }
else
 v[1]=0; //product not broken
RESULT(v[1] )
```

## Optimization

The LSD element `c` is a pointer, like `cur`, but it assigned by the system to contain the object whose equation requested the computation of the variable.

Thus, the equation for **IsBroken** can now safely access the specific consumer asking for its computation.

## Optimization

A second type of optimization consists in reducing the need for the system to scan sets of objects searching for those with specific properties.

The command `SEARCH_CND("IdFirm",v[0])` replicates this search every time step, even though the object returned is the same until a new one is purchased. Here is how we can exploit an optimization tools in LSD.

# Optimization

Let's use **Purchase** to assign a special field available in any Lsd object, called `hook`. This is a physical link to connect the object to another one.

```
EQUATION("Purchase")
/*
Make a purchase for the calling object (supposedly a consumer).
*/
cur1=RNDDRAW("Firm","Visibility");  //choose randomly one of the products
v[0]=VS(cur1,"IdFirm"); //return the ID of the chosen firm
V("InitTrade"); //ensure that firms are ready to sell
INCRS(cur1,"Sales",1); //increase the Sales of the chosen firm
c->hook=cur1; //Attach to the consumer's hook the newly purchased product
RESULT(v[0] )
```

## Optimization

If the `hook` is correctly set, we don't need to do any search, because each consumer is associated to the product used. Thus, **IsBroken** can be simplified as follows.

# Optimization

```
EQUATION("IsBroken")
/*
Check whether the product breaks down or not.
Return 1 if the product breaks or 0 if not.
*/
// USELESS v[0]=VLS(c,"ProdUsed",1); //product owned by the consumer at t-1
// USELESS cur=SEARCH_CND("IdFirm",v[0]); //search the object with IdFirm
v[2]=VS(c->hook,"BD"); //read the value of the prob. to break down
if(RND<v[2]) //
 {v[1]=1; //product broken
  V("InitTrade"); //just to ensure NumLost is reset
  INCRS(c->hook,"NumLost",1);
 }
else
 v[1]=0; //product not broken
RESULT(v[1] )
```