[http://book.xogeny.com/_static/images/TitleHeading.png](http://book.xogeny.com/)

Release: v0.3.6-1-gc28d7f1-Early Access

[Basic Equations](http://book.xogeny.com/behavior/equations/)

**Lotka-Volterra Systems**

Table Of Contents

* [Basic Equations](http://book.xogeny.com/behavior/equations/)
  + [Lotka-Volterra Systems](http://book.xogeny.com/behavior/equations/population/)
    - [Classic Lotka-Volterra](http://book.xogeny.com/behavior/equations/population/#classic-lotka-volterra)
    - [Steady State Initialization](http://book.xogeny.com/behavior/equations/population/#steady-state-initialization)
    - [Avoiding Repetition](http://book.xogeny.com/behavior/equations/population/#avoiding-repetition)

Lotka-Volterra Systems

So far, we’ve seen thermal, electrical and mechanical examples. In effect, these have all been engineering examples. However, Modelica is not limited strictly to engineering disciplines. To reinforce this point, this section will present a common ecological system dynamics model based on the relationship between predator and prey species. The equations we will be using are the [[Lotka]](http://book.xogeny.com/behavior/equations/population/#lotka)-[[Volterra]](http://book.xogeny.com/behavior/equations/population/#volterra) equations.

Classic Lotka-Volterra

The [classic Lotka-Volterra model](http://en.wikipedia.org/wiki/Lotka-Volterra_equation) involves two species. One species is the “prey” species. In this section, the population of the prey species will be represented by x. The other species is the “predator” species whose population will be represented by y.

There are three important effects in a Lotka-Volterra system. The first is reproduction of the “prey” species. It is assumed that reproduction is proportional to the population. If you are familiar with chemical reactions, this is conceptually the same as the [Law of Mass Action](http://en.wikipedia.org/wiki/Law_of_mass_action) [[Guldberg]](http://book.xogeny.com/behavior/equations/population/#guldberg). If you aren’t familiar with the Law of Mass Action, just consider that the more potential mates are present in the environment, the more likely reproduction is to occur. We can represent this mathematically as:

x˙r=αx

where x is the prey population, x˙r is the change in prey population *due to reproduction* and α is the proportionality constant capturing the likelihood of successful reproduction.

The next effect to consider is starvation of the predator species. If there aren’t enough “prey” around to eat, the predator species will die off. When modeling starvation, it is important to consider the effect of competition. We again have a proportionality relationship, but this time it works in reverse because, unlike with prey reproduction, the more predators around the more likely starvation is. This is expressed mathematically in much the same way as reproduction:

y˙s=−γy

where y is the predator population, y˙s is the change in predator population *due to starvation* and γ is the proportionality constant capturing the likelihood of starvation.

Finally, the last effect we need to consider is “predation”, *i.e.*, the consumption of the prey species by the predator species. Without predators, the prey species would (at least mathematically) grow exponentially. So predation is the effect that keeps the prey species population in check. Similarly, without any prey, the predator species would simply die off. So predation is what balances out this effect and keeps the predator population from going to zero. Again, we have a proportionality relationship. But this time, it is actually a bilinear relationship that is, again, conceptually similar to the Law of Mass Action. This relationship is simply capturing, mathematically, the fact that the chance that a predator will find and consume some prey is proportional to both the population of the prey and the predators. Since this particular effect requires both species to be involved, this mathematical relationship has a slightly different structure than reproduction and starvation, *i.e.,*

x˙py˙p=−βxy=δxy

where x˙p is the decline in the prey population *due to predation*, y˙p is the increase in the predator population *due to predation*, β is the proportionality constant representing the likelihood of prey consumption and δ is the proportionality constant representing the likelihood that the predator will have sufficient extra nutrition to support reproduction.

Taking the various effects into account, the overall change in each population can be represented by the following two equations:

x˙y˙=x˙r+x˙p=y˙p+y˙s

Using the previous relationships, we can expand each of the right hand side terms in these two equations into:

x˙y˙=x(α−βy)=y(δx−γ)

Using what we’ve learned in this chapter so far, translating these equations into Modelica should be pretty straightforward:

**model** **ClassicModel** "This is the typical equation-oriented model"

**parameter** Real alpha=0.1 "Reproduction rate of prey";

**parameter** Real beta=0.02 "Mortality rate of predator per prey";

**parameter** Real gamma=0.4 "Mortality rate of predator";

**parameter** Real delta=0.02 "Reproduction rate of predator per prey";

**parameter** Real x0=10 "Start value of prey population";

**parameter** Real y0=10 "Start value of predator population";

Real x(start=x0) "Prey population";

Real y(start=y0) "Predator population";

**equation**

der(x) = x\*(alpha-beta\*y);

der(y) = y\*(delta\*x-gamma);

**end** **ClassicModel**;

At this point, there is only one thing we haven’t discussed yet and that is the presence of the start attribute on x and y. As we saw in theNewtonCoolingWithUnits example in the previous section titled [*Getting Physical*](http://book.xogeny.com/behavior/equations/physical/#getting-physical), variables have various attributes that we can specify (for a detailed discussion of available attributes, see the upcoming section on [*Built-In Types*](http://book.xogeny.com/behavior/equations/variables/#builtin-types)). We previously discussed the unit attribute, but this is the first time we are seeing the start attribute.

The observant reader may have noticed the presence of the x0 and y0 parameter variables and the fact that they represent the initial populations. Based on previous examples, one might have expected these initial conditions to be captured in the model as follows:

**model** **ClassicModelInitialEquations** "This is the typical equation-oriented model"

**parameter** Real alpha=0.1 "Reproduction rate of prey";

**parameter** Real beta=0.02 "Mortality rate of predator per prey";

**parameter** Real gamma=0.4 "Mortality rate of predator";

**parameter** Real delta=0.02 "Reproduction rate of predator per prey";

**parameter** Real x0=10 "Initial prey population";

**parameter** Real y0=10 "Initial predator population";

Real x(start=x0) "Prey population";

Real y(start=y0) "Predator population";

**initial equation**

x = x0;

y = y0;

**equation**

der(x) = x\*(alpha-beta\*y);

der(y) = y\*(delta\*x-gamma);

**end** **ClassicModelInitialEquations**;

However, for the ClassicModel example we took a small shortcut. As will be discussed shortly in the section on [*Initialization*](http://book.xogeny.com/behavior/equations/initialization/#initialization), we can specify initial conditions by specifying the value of the start attribute directly on the variable.

It is worth noting that this approach has both advantages and disadvantages. The advantage is one of flexibility. The start attribute is actually more of a hint than a binding relationship. If the Modelica compiler identifies a particular variable as a state (*i.e.*, a variable that requires an initial condition) **and**there are insufficient initial conditions already explicitly specified in the model via initial equation sections then it can substitute the start attribute as an initial condition for the variable it is associated with. In other words, you can think of the start attribute as a “fallback initial condition” if an initial condition is needed.

There are a couple of disadvantages to the start attribute that you need to watch out for. First, it is only a hint and tools may completely ignore it. Next, whether it will be ignored is also hard to predict since different tools may make different choices about which variables to treat as states.

One way to avoid both of these disadvantages is to use the fixed attribute (also discussed in the section on [*Built-In Types*](http://book.xogeny.com/behavior/equations/variables/#builtin-types)). The fixed attribute can be used to tell the compiler that the start attribute **must** be used as an initial condition. In other words, an initial equation like this:

Real x;

**initial equation**

x = 5;

is equivalent to the following declaration utilizing the start and fixed attributes:

Real x(start=5, fixed=**true**);

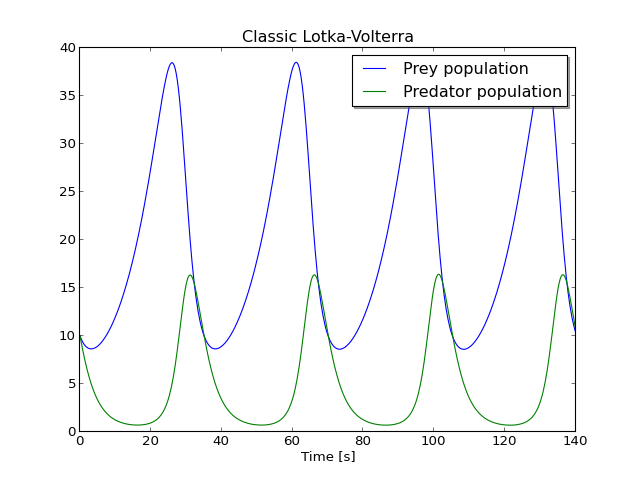
Finally, one additional complication is that the start attribute is also “overloaded”. This means that it is actually used for two different things. If the variable in question is not a state, but is instead an “iteration variable” (*i.e.*, a variable whose solution depends on a non-linear system of equations), then the startattribute may be used by a Modelica compiler as an initial guess (*i.e.*, the value used for the variable during the initial iteration of the non-linear solver).

Whether to specify a start attribute or not depends on how strictly you want a given initial condition to be enforced. Knowing that is something that takes experience working with the language and is beyond the scope of this chapter. However, it is worth at least pointing out that there are different options along with a basic explanation of the trade-offs.

Using either initialization method, the results for these models will be the same. The typical behavior for the Lotka-Volterra system can be seen in this plot:

([Source code](http://book.xogeny.com/plots/LVCM.py))

 Simulate ClassicModel in your browser



Note the cyclical behavior of each population. Initially, there are more predators than can be supported by the existing food supply. Those predators that are present consume whatever prey the can find. Nevertheless, some starvation occurs and the predator population declines. The rate at which predators consume the prey species is so high during this period that the rate at which the prey species reproduces is not sufficient to make up for those lost to predation so the prey population declines as well.

At some point, the predator population gets so low that the rate of reproduction in the prey species is larger than the rate of prey consumption by the predators and the prey species begins to rebound. Because the predator species population takes longer to rebound, the prey species experiences growth that is, for the moment, virtually unchecked by predation. Eventually, the predator population begins to rebound due to the abundance of prey until the system returns to the original predator and prey populations **and the entire cycle then repeats itself** *ad infinitum*.

The fact that the system returns again and again to the same initial conditions (ignoring numerical error, of course) is one of the most interesting things about the system. This is even more remarkable given the fact that the Lotka-Volterra system of equations is actually non-linear.

Steady State Initialization

Let’s imagine that these extreme swings in species population had some undesirable ecological consequences. In such a case, it would be useful to understand what might reduce or even eliminate these fluctuations. A simple approach would be to keep the populations in a state of equilibrium. But how can we use these models to help use determine such a “quiescent” state?

The answer lies in the initial conditions. Instead of specifying an initial population for both the predator and prey species, we might instead chose to initialize the system with some other equations that somehow capture the fact that the system is in equilibrium (you may remember this trick from theFirstOrderSteady model [*discussed previously*](http://book.xogeny.com/behavior/equations/first_order/#ex-simpleexample-firstordersteady)). Fortunately, Modelica’s approach to initialization is rich enough to allow us to specify this (and many other) useful types of initial conditions.

To ensure that our system starts in equilibrium, we simply need to define what equilibrium is. Mathematically speaking, the system is in equilibrium if the following two conditions are met:

x˙=0

y˙=0

To capture this in our Modelica model, all we need to do is use these equations in our initial equation section, like this:

**model** **QuiescentModel** "Find steady state solutions to LotkaVolterra equations"

**parameter** Real alpha=0.1 "Reproduction rate of prey";

**parameter** Real beta=0.02 "Mortality rate of predator per prey";

**parameter** Real gamma=0.4 "Mortality rate of predator";

**parameter** Real delta=0.02 "Reproduction rate of predator per prey";

Real x "Prey population";

Real y "Predator population";

**initial equation**

der(x) = 0;

der(y) = 0;

**equation**

der(x) = x\*(alpha-beta\*y);

der(y) = y\*(delta\*x-gamma);

**end** **QuiescentModel**;

The main difference between this and our previous model is the presence of the highlighted initial equations. Looking at this model, you might wonder exactly what those initial equations mean. After all, what we need to solve for are x and y. But those variables don’t even appear in our initial equations. So how are they solved for?

The answer lies in understanding that the functions x(t) and y(t) are solved for by integrating the differential equations starting from some initial equations. During the simulation, we see that x and x˙ are “coupled” by the following equations:

x(t)=∫tft0x˙ dx+x(t0)

(and, of course, a similar relationship exists between y and y˙)

**However**, during initialization of the system (*i.e.*, when solving for the initial conditions) this relationship doesn’t hold. So there is no “coupling” between xand x˙ in that case (nor for y: and y˙). In other words, knowing x or y doesn’t give you any clue as to how to compute x˙ or y˙. The net result is that for the initialization problem we can think of x, y, x˙ and y˙ as four independent variables.

Said another way, while simulating, we solve for x by integrating x˙. So that integral equation is the equation used to solve for x. But during initialization, we cannot use that equation so we need an additional equation (for each integration that we would otherwise perform during simulation).

In any case, the bottom line is that during initialization we require four different equations to arrive at a unique solution. In the case of our QuiescentModel, those four equations are:

x˙=0

x˙=x (α−βy)

y˙=0

y˙=y (δx−γ)

It is very important to understand that these equations **do not contradict each other**. Especially if you come from a programming background you might look at the first two equations and think “Well what is x˙? Is it zero or is it x(α−βy)?” The answer is **both**. There is no reason that both equations cannot be true!

The essential thing to remember here is that these are **equations not assignment statements**. The following system of equations is mathematically identical and demonstrates more clearly how x and y could be solved:

x˙=0

y˙=0

x(α−βy) =x˙

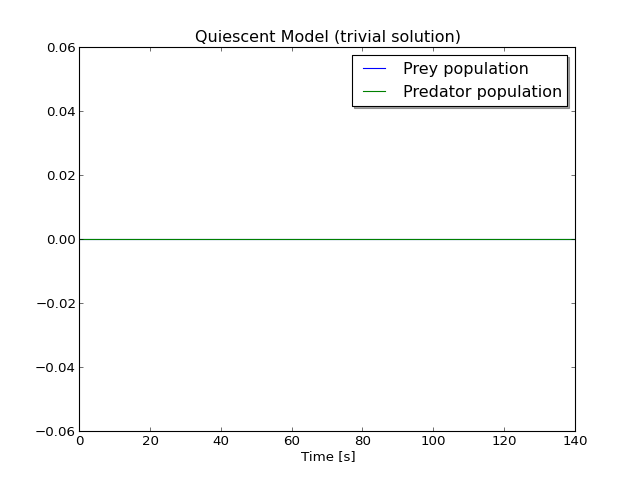
y(δx−γ)=y˙

In this form, it is a bit easier to recognize how we could arrive at values of x and y. The first thing to note is that we cannot solve explicitly for x and y. In other words, we cannot rearrange these equations into the form x=... without having x also appear on the right hand side. So we have to deal with the fact that **this is a simultaneous system of equations** involving both x and y.

But the situation is further complicated by the fact that this system is non-linear (which is precisely why we cannot use linear algebra to arrive at a set of explicit equations). In fact, if we study these equations carefully we can spot the fact that there exist two potential solutions. One solution is trivial (x=0;y=0) and the other is not.

So what happens if we try to simulate our QuiescentModel? The answer is pretty obvious in the plot below:

([Source code](http://book.xogeny.com/plots/LVQM.py))



We ended up with the trivial solution where the prey and predator populations are zero. In this case, we have no reproduction, predation or starvation because all these effects are proportional to the populations (*i.e.*, zero) so nothing changes. But this isn’t a very interesting solution.

There are two solutions to this system of equations because it is non-linear. How can we steer the non-linear solver away from this trivial solution? If you were paying attention during the discussion of the [*Classic Lotka-Volterra*](http://book.xogeny.com/behavior/equations/population/#classic-lotka-volterra) model, then you’ve already been given a hint about the answer.

Recall that the start attribute is overloaded. During our discussion of the [*Classic Lotka-Volterra*](http://book.xogeny.com/behavior/equations/population/#classic-lotka-volterra) model, it was pointed out that one of the purposes of thestart attribute was to provide an initial guess if the variable with the start attribute was chosen as an iteration variable. Well, our QuiescentModel happens to be a case where x and y are, in fact, iteration variables because they must be solved using a system of non-linear equations. This means that if we want to avoid the trivial solution, we need to specify values for the start attribute on both x and y that are “far away” from the trivial solution we are trying to avoid (or at least closer to the non-trivial solution we seek). For example:

**model** **QuiescentModelUsingStart** "Find steady state solutions to LotkaVolterra equations"

**parameter** Real alpha=0.1 "Reproduction rate of prey";

**parameter** Real beta=0.02 "Mortality rate of predator per prey";

**parameter** Real gamma=0.4 "Mortality rate of predator";

**parameter** Real delta=0.02 "Reproduction rate of predator per prey";

Real x(start=10) "Prey population";

Real y(start=10) "Predator population";

**initial equation**

der(x) = 0;

der(y) = 0;

**equation**

der(x) = x\*(alpha-beta\*y);

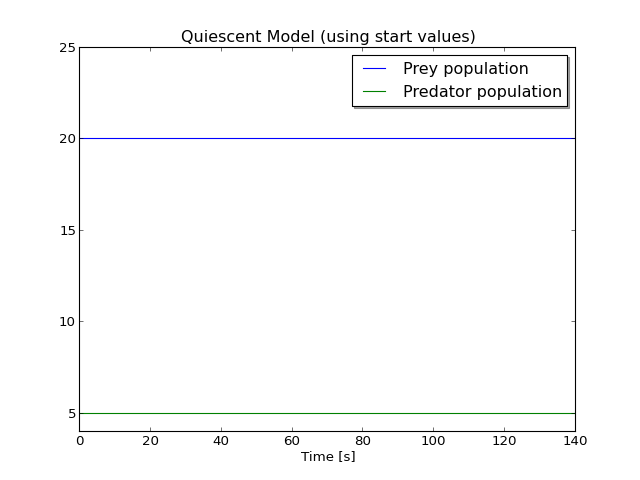
der(y) = y\*(delta\*x-gamma);

**end** **QuiescentModelUsingStart**;

This model leads us to a set of initial conditions that is more inline with what we were originally looking for (*i.e.*, one with non-zero populations for both the predator and prey species).

([Source code](http://book.xogeny.com/plots/LVQMUS.py))

 Simulate QuiescentModelUsingStart in your browser



It is worth pointing out (as we will do shortly in the section on [*Built-In Types*](http://book.xogeny.com/behavior/equations/variables/#builtin-types)), that **the default value of the ``start`` attribute is zero**. This is why when we simulated our original QuiescentModel we happened to land exactly on the trivial solution...because it was our initial guess and it happened to be an exact solution so no other solution or iterating was required.

Avoiding Repetition

We’ve already seen several different models (ClassicModel, QuiescentModel and QuiescentModelUsingStart) based on the Lotka-Volterra equations. Have you noticed something they all have in common? If you look closely, you will see that they have almost **everything** in common and that there are actually hardly any **differences** between them!

In software engineering, there is a saying that “Redundancy is the root of all evil”. Well the situation is no different here (in no small part because Modelica code really is software). The code we have written so far would be very annoying to maintain. This is because any bugs we found would have to be fixed in each model. Furthermore, any improvements we made would also have to be applied to each model. So far, we are only dealing with a relatively small number of models. But this kind of “copy and paste” approach to model development will result in a significant proliferation of models with only slight differences between them.

So what can be done about this? In object-oriented programming languages there are basically two mechanisms that exist to reduce the amount of redundant code. They are *composition* (which we will address in the future chapter on [*Components*](http://book.xogeny.com/components/components/#components)) and *inheritance* which we will briefly introduce here.

If we look closely at the QuiescentModelUsingStart example, we see that there are almost no differences between it and our original ClassicModel version. In fact, the only real differences are shown here:

**model** **ClassicModel** "This is the typical equation-oriented model"

**parameter** Real alpha=0.1 "Reproduction rate of prey";

**parameter** Real beta=0.02 "Mortality rate of predator per prey";

**parameter** Real gamma=0.4 "Mortality rate of predator";

**parameter** Real delta=0.02 "Reproduction rate of predator per prey";

**parameter** Real x0=10 "Start value of prey population";

**parameter** Real y0=10 "Start value of predator population";

Real x(start=x0) "Prey population";

Real y(start=y0) "Predator population";

**equation**

der(x) = x\*(alpha-beta\*y);

der(y) = y\*(delta\*x-gamma);

**end** **ClassicModel**;

**model** **QuiescentModelUsingStart** "Find steady state solutions to LotkaVolterra equations"

**parameter** Real alpha=0.1 "Reproduction rate of prey";

**parameter** Real beta=0.02 "Mortality rate of predator per prey";

**parameter** Real gamma=0.4 "Mortality rate of predator";

**parameter** Real delta=0.02 "Reproduction rate of predator per prey";

Real x(start=10) "Prey population";

Real y(start=10) "Predator population";

**initial equation**

der(x) = 0;

der(y) = 0;

**equation**

der(x) = x\*(alpha-beta\*y);

der(y) = y\*(delta\*x-gamma);

**end** **QuiescentModelUsingStart**;

In other words, the only real difference is the addition of the initial equation section (the original ClassicModel already contained non-zero start values for our two variables, x and y). Ideally, we could avoid having any redundant code by simply defining a model in terms of the differences between it and another model. As it turns out, this is exactly what the extends keyword allows us to do. Consider the following alternative to theQuiescentModelUsingStart model:

**model** **QuiescentModelWithInheritance** "Steady state model with inheritance"

**extends** ClassicModel;

**initial equation**

der(x) = 0;

der(y) = 0;

**end** **QuiescentModelWithInheritance**;

Note the presence of the extends keyword. Conceptually, this “extends clause” simply asks the compiler to insert the contents of another model (ClassicModel in this case) into the model being defined. In this way, we copy (or “inherit”) everything from ClassicModel without having to repeat its contents. As a result, the QuiescentModelWithInheritance is the same as the ClassicModel with an additional set of initial equations inserted.

But what about a case where we don’t want **exactly** what is in the model we are inheriting from? For example, what if we wanted to change the values of the gamma and delta parameters?

Modelica handles this by allowing us to include a set of “modifications” when we use extends. These modifications come after the name of the model being inherited from as shown below:

**model** **QuiescentModelWithModifications** "Steady state model with modifications"

**extends** QuiescentModelWithInheritance(gamma=0.3, delta=0.01);

**end** **QuiescentModelWithModifications**;

Also note that we could have inherited from ClassicModel, but then we would have had to repeat the initial equations in order to have quiescent initial conditions. But by instead inheriting from QuiescentModelWithModifications, we reuse the content from **two** different models and avoid repeating ourselves even once.

**More population dynamics**

This concludes the set of examples for this chapter. If you’d like to explore the Lotka-Volterra equations in greater depth, an upcoming section titled[*Lotka-Volterra Equations Revisited*](http://book.xogeny.com/components/components/population/#population-components) demonstrates how to build complex models of population dynamics using graphical components that are dropped onto a schematic and connected together.

|  |  |
| --- | --- |
| [[Lotka]](http://book.xogeny.com/behavior/equations/population/#id2) | Lotka, A.J., “Contribution to the Theory of Periodic Reaction”, J. Phys. Chem., 14 (3), pp 271–274 (1910) |

|  |  |
| --- | --- |
| [[Volterra]](http://book.xogeny.com/behavior/equations/population/#id3) | Volterra, V., Variations and fluctuations of the number of individuals in animal species living together in Animal Ecology, Chapman, R.N. (ed), McGraw–Hill, (1931) |

|  |  |
| --- | --- |
| [[Guldberg]](http://book.xogeny.com/behavior/equations/population/#id5) | C.M. Guldberg and P. Waage,”Studies Concerning Affinity” C. M. Forhandlinger: Videnskabs-Selskabet i Christiana (1864), 35 |