Vensim 5 Modeling Guide

1 System Dynamics Overview

This modeling guide is intended to introduce some of the basic concepts of building and using models and provide a number of examples to improve understanding. If you are new to Vensim you should first work through the *Vensim User's Guide.*.

Each chapter in this *Guide* contains a model, or set of models, which you can build as you work through the chapter. Finished models for each section are included with the software in the *mguide* subdirectory of *models* (normally *c:\Program Files\Vensim\models\mguide*). Each chapter has a subdirectory that begins with the chapter number. The name of the model appears in the applicable title or subtitle of this guide. If you want to build the models yourself, you should use a different name or work in a different directory.

There are very few mechanical instructions for building models in this guide. If you are having trouble, you might want to go back to the *Tutorial* and find the instructions for building a similar structure.

Events, Behavior and Structure

Our lives are filled with events: birthday parties, graduations, job starts, product launches, retirements, arguments, agreements and storms. Because of their prevalence events tend to fill our discussions. In terms of understanding our world, however, events turn out to have limited usefulness. These limitations are well recognized in the physical sciences. If a teacher were to stand in front of a class and drop a piece of chalk and ask the class "why did the chalk hit the ground?" the response, "because you let go of it" would generate a chuckle and quickly be dismissed. If, however, you were to put the same student into a suit and ask of her "why did stock prices fall?" the response "because the Federal Reserve announced that it was increasing interest rates" would be considered serious and correct.

One step back from events is the idea of behavior patterns. A behavior pattern is something that connects together a long series of events over time. The American revolution was an event. The extent of suppression, resentment and taxation in the decades preceding the revolution were patterns of behavior. Once you step away from events and begin considering patterns of behavior questions such as "what caused ..." are given a different and much deeper meaning. We are no longer searching for an event that precedes or corresponded with another event. Rather we are looking for sources of pressure and imbalance that cause things to change.

Structure is the set of physical and information interconnections that generate behavior. Inventory is the accumulation of production less shipments. Workforce changes with hires and attrition and hiring is based on the targeting of production to meet demand and correct inventory imbalances. The result of this is that the inventory level moves up and down (behavior) and we now have so much inventory that we are not profitable and the CEO has been fired (an event). Structure determines behavior and events are snapshots of that behavior.

The event—behavior—structure distinction is an important tool for understanding and working with problems. Ultimately, successful policies and interventions need to be changes to structure, so that behavior is improved and bad events become less frequent. System dynamics and Vensim, provide you with tools to represent structure, and understand how it determines behavior.

The System Dynamics Process

Though there is no universally accepted process for developing and using good quality system dynamics models there are some basic practices that are quite commonly used. A more complete description of the modeling process is contained in "Guidelines for Model Conceptualization," by

Jørgen Randers in Jørgen Randers (ed.), Elements of the System Dynamics Method, MIT Press, Cambridge, MA 1980 pp. 117-138 (now available from Pegasus Communications).

The following steps are a useful guideline.

Issue statement. The issue statement is simply a statement of the problem that makes it clear what the purpose of the model will be. Clarity of purpose is essential to effective model development. Developing a model of a system or process without specifying how the system needs to be improved or what specific behavior is problematic is difficult. Having a clear problem in mind makes it easier to develop models with good practical applicability.

Variable Identification. Identify some key quantities that will need to be included in the model for the model to be able to address the issues at hand. Usually a number of these are very obvious. It can sometimes be useful just to write down all of the variables that might be important and try to rank them in order to identify the most important ones.

Reference modes. A reference mode is a pattern of behavior over time. Reference modes are drawn as graphs over time for key variables, but are not necessarily graphs of observed behavior. Rather, reference modes are cartoons that show a particular characteristic of behavior that is interesting. For example, a company's sales history may be growing but bumpy, and the reference mode may be the up and down movement around the growth trend. Reference modes can refer to past behavior, or future behavior. They can represent what you expect to have happen, what you fear will happen and what you hope will happen. Reference modes should be drawn with an explicitly labeled time axis to help refine, clarify and bound a problem statement.

Reality Check. Define some Reality Check statements about how things must interrelate. These include a basic understanding of what actors are involved and how they interact, along with the consequences for some variables of significant changes in other variables. Reality Check information is often simply recorded as notes (often mental notes) about what connections need to exist. It is based on knowledge of the system being modeled. In Chapter 9 we will make this information explicit using the Reality Check functionality in Vensim.

Dynamic hypotheses. A dynamic hypothesis is a theory about what structure exists that generates the reference modes. A dynamics hypothesis can be stated verbally, as a causal loop diagram, or as a stock and flow diagram. The dynamic hypotheses you generate can be used to determine what will be kept in models, and what will be excluded. Like all hypotheses, dynamic hypotheses are not always right. Refinement and revision is an important part of developing good models.

Simulation Model. A simulation model is the refinement and closure of a set of dynamic hypotheses to an explicit set of mathematical relationships. Simulation models generate behavior through simulation. A simulation model provides a laboratory in which you can experiment to understand how different elements of structure determine behavior.

The above process is iterative and flexible. As you continue to work with a problem you will gain understanding that changes the way you need to think about the things you have done before. Later chapters in this guide go through the process of model development from a number of different perspectives placing emphasis on different parts of the above process.

Vensim provides explicit support for naming variables, writing Reality Check information, developing dynamic hypotheses and building simulation models. Creating good issue statements and developing reference modes can easily be done with pencil and paper or using other technologies. Dynamic hypotheses can be developed as visual models in Vensim, or simply sketched out with pencil and paper. Simulation is the one stage where it would be impractical to dispense with the computer altogether.

Fundamental Structures and Behaviors

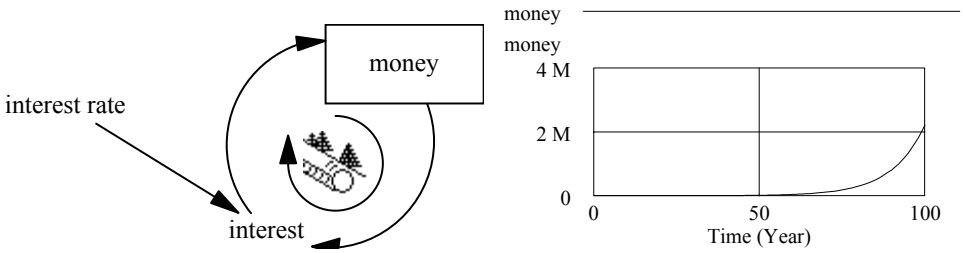
As discussed above the process of building models requires the generation of an hypothesis about what kind of structure might be responsible for the behavior in the reference mode. This can be quite

difficult, but it does become easier as you gain experience with model building. One reason it becomes easier is that you will gain experience with different structures and the behavior they generate. Studying the simplest and most fundamental patterns of behavior and the simplest structure that can generate the behavior is a useful way to get started on gaining this experience.

In the following sections we percent growth, decay, adjustment and oscillation and some simple structures that generate them. The models are presented in causal loop form to emphasize feedback which is an essential part of generating a dynamic hypothesis.

Exponential Growth (money.mdl)

Suppose that you deposit $100 in the bank. If the interest rate is 10% per year (compounded daily) and you wait 100 years what will happen? This is an example of a first order positive feedback loop.

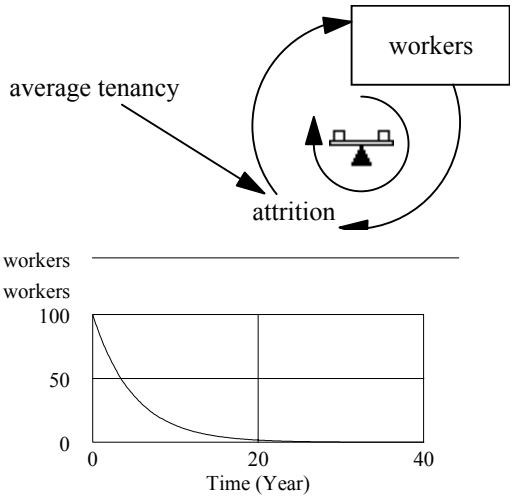


At the end of 100 years there would be over two million dollars in the bank. Exponential growth is interesting because it demonstrates a constant doubling time. If it takes, as it does in this example, about 7 years to go from 100 to 200 dollars, it will also take about 7 years to go from 1 million to 2 million dollars. It is a useful exercise to explore the relationship between the interest rate and the time it takes the money to double.

Note that for this example you can either select TIME STEP to be small enough (about .125) so that it makes no difference or choose Runge-Kutta integration.

Exponential Decay(workers.mdl)

Suppose that you have 100 people working for you and you decide never to hire anyone again. Your average worker hangs around for 10 years. What will happen to your workforce? This is an example of a first order negative feedback loop.



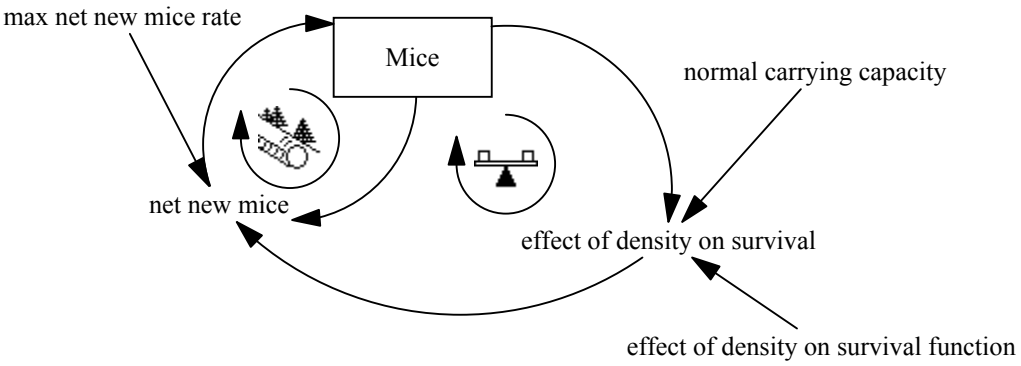
The number of workers will decrease, very quickly at first and then more slowly. This is the opposite of exponential growth where changes was slow at first and then quick and there is a similar constancy in exponential decay. If it takes, as it does here, about seven years for half the workers to leave, it will

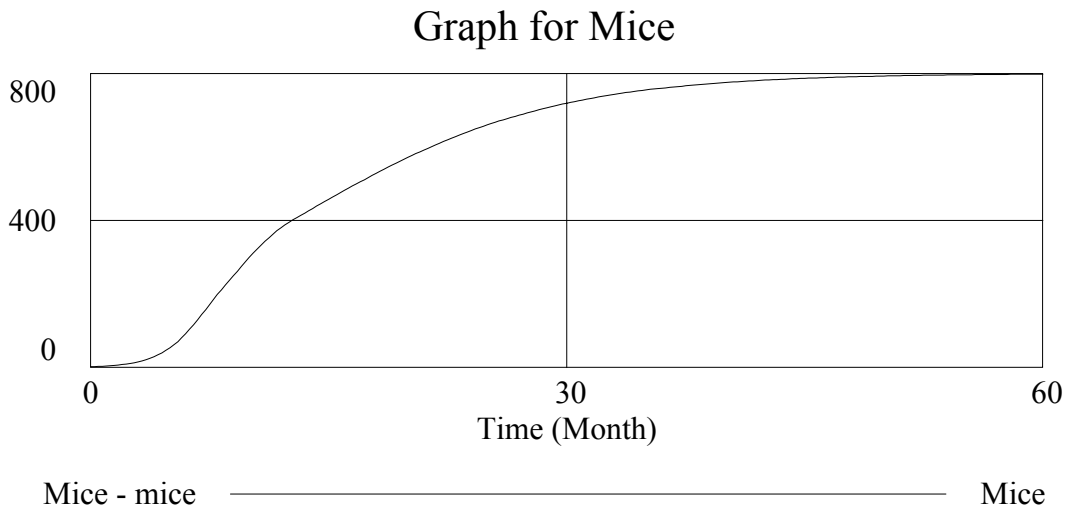
take another seven for half of those that remain to leave. It is interesting to experiment and see the relationship between *average tenancy* and the time it takes half the workers to leave.

The important thing about exponential decay is that if you move the goal away from zero, you get the same behavior. Suppose you have a house at 10 degrees and you put it into a 40 degree neighborhood, what happens to the house temperature?

S Shaped Growth (mice.mdl)

Suppose that you let some mice loose in your house and don't try to kill them. What will happen to the mouse population? At first glance this sounds very much like the exponential growth example, and for the first little time it is. Try as they might, though, the mice will never get 2 million of themselves into your house.



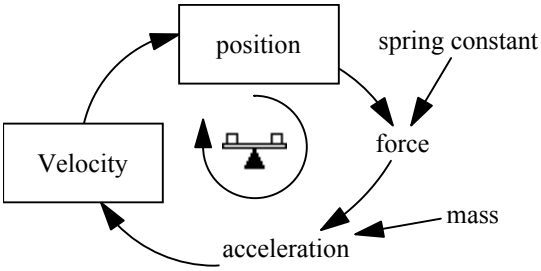


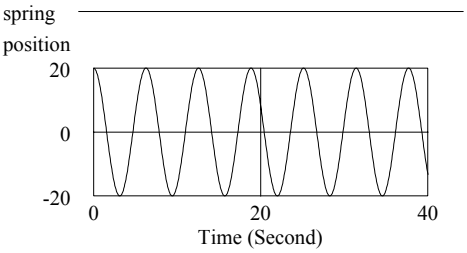
The beginning of this does indeed look like exponential growth while the end looks like exponential adjustment. To get this kind of transition there has to be more than a single feedback loop operating. Often, and in the case of this simple model, there are both a positive and negative feedback loop with the positive loop dominate in the beginning and the negative loop dominant in the end.

It is interesting to experiment with the Constant *max net new mice rate* to see the effect this has. Watching the way *net new mice behaves* as you change values is especially interesting. The process of adjustment in this model is almost entirely dependent on the shape of the Lookup *effect of density on survival function*. Experimenting with different shapes for this function can also be interesting.

Oscillation (spring.mdl)

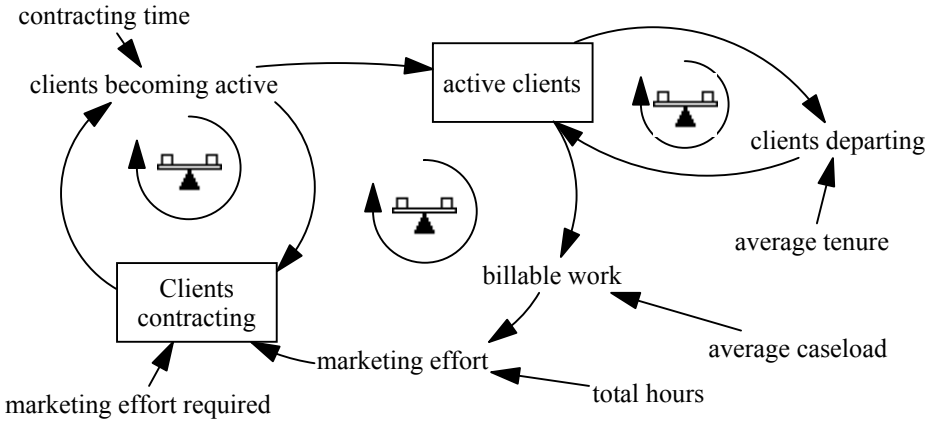
Consider the problem of a spring pushing a weight on a frictionless horizontal surface.



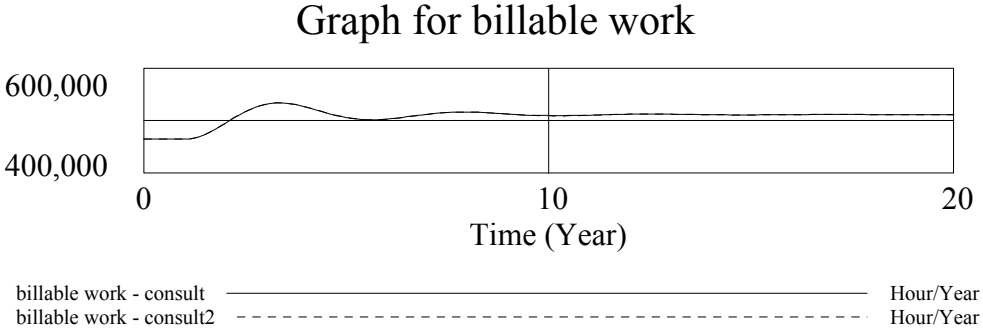


You need to be sure to use Runge-Kutta integration when simulating this model to get the correct result. This is the simplest structure that can generate oscillations, and the oscillations are undamped.

Consider the closely related problem of a constancy that uses time not spent doing billing work to bring in new clients (*consult.mdl*):



This model is driven by a step increase in total hours. It also has a somewhat involved set of initialization equations and is shown with these links hidden.



The striking thing in comparing these two oscillating models is the addition of the two extra loops in the second model. It is actually quite rare to get a single loop with more than a single level in business systems. The rule is that there are likely to be a number of minor negative loops and these, in general, will be damping.

It is a useful exercise to experiment with changing constants in these to models and understanding how they influence behavior. The Workforce Inventory model in Chapter 2 and the discussion in Chapter 8 go further into oscillation.

2 Workforce, Inventory and Oscillation

This chapter is focused on the conceptual issues in the development, analysis and use of a model. After working through this chapter you should have a better understanding of how to think about, and work through, a dynamic model with Vensim.

Background

You are involved in the production and sale of prefabricated window frames. Overall your company is doing quite well, but you often go through periods of low capacity utilization followed by production ramp up and added shifts. While all of this is normally blamed on market demand and the condition of the economy, you have your doubts. Looking back at sales and production over the last 8 years it seems that sales is more stable than production. Your goal is to determine why this might be, and what you can do about it.

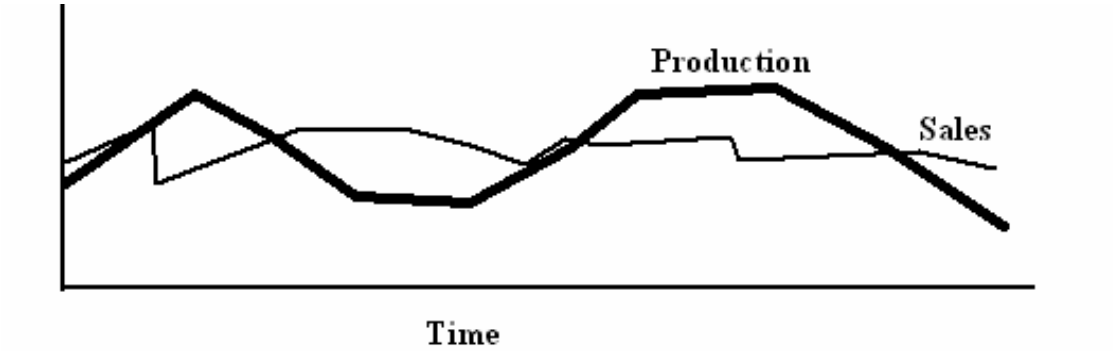
In attacking this problem you want to simplify as much as possible your current situation. There are a number of reasons for this simplification:

- It is easier to understand a simple model.
- You can get results quickly and decide if you are on the right track.
- It is more effective to start with a simple model and add detail, than to build a complex model and attempt to extract insights from it after it is complete.
- Using a simple model forces you to take an overview which is usually useful in the initial modeling phases.

It is not always true that you want to start building simple models. In many cases the behavior you see is the result of complexity, and Vensim provides a very rich set of tools for dealing with complexity. Until you have had substantial experience, however, simplicity is highly recommended. While it is not uncommon to discover that what you have developed is not rich enough for the problem you wish to address, it is rare not to gain understanding in the process. Conversely, large complex models can become significant resource drains, providing no payoff for a very long time.

Reference Modes

A reference mode is a graphical statement about a problem. Verbally, the problem was stated as "production is less stable than sales." Graphically we might draw:



This reference mode is a sketch of behavior we might expect a model to produce. It might be real data from your records, or your expectation of what might happen in a new situation. The reference mode

is used to focus activity. Having mapped out one or more reference modes the goal is to define the simplest structure that is sensible and capable of generating patterns of behavior that qualitatively resemble the reference modes. If appropriate, such a structure can also be refined in order to develop a model that can be validated quantitatively against the available data.

Reality Check

Lets put down some common sense statements about how the business works.

- Without any workers there is no production.
- Without any inventory we can't ship.
- If sales go up for a sustained period we will try to expand production.
- With no production inventory will never go up.

Reality Check information in this form should be kept in the back of your mind as you develop dynamics hypotheses and build a simulation model.

Dynamic Hypothesis

A dynamic hypothesis is an idea about what structure might be capable of generating behavior like that in the reference modes. For this example we can formulate a dynamic hypothesis simply by thinking about how the two variables in the reference mode are connected — that is by specifying the set of policies (or rules) that determine *production* given *sales*. The dynamic hypothesis for this firm is that a manager is setting production based on current sales, but is amplifying the amount resulting in higher (or lower) production than is necessary. The reference mode supplies us with two variables — *production* and *sales* — that we will want to include in the model. This is a reasonably good basis on which to begin a sketch, so let us put these variables down to start the model.

Workforce / Inventory Model (wfinv1.mdl)

Now the question arises, how are *production* and *sales* related. Clearly there is a close relationship, since it is necessary to produce something before it is sold. Sales and production are related in two ways:

Physical: production is required to produce goods to sell

Information: managers base production decisions on current or recent sales

We will start the model with the physical side. When production occurs, goods are not immediately sold. Instead, they are stored in an inventory until a sale occurs, at which point they are removed from inventory. In general, there will be an inventory, or some combination of inventory and backlog, separating production from sales. If a backlog is used in the model, it is useful to consider orders and shipments instead of simply sales. In this model, we will just use an inventory.

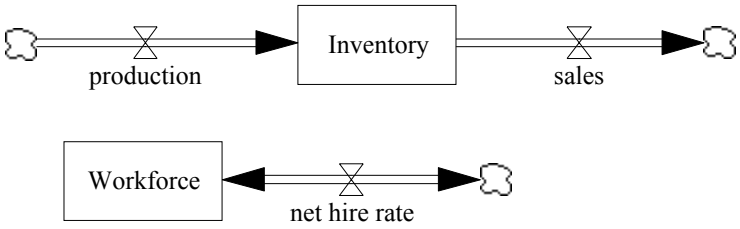
We construct *Inventory* as a Level, then add a rate flowing in and a rate flowing out. Next, we use the variable merge tool to drag our two existing variables, *production* and *sales* onto the valves.

It is worth noting at this point that we could have created the same diagram by first entering the level containing *Inventory*, then adding and naming the two rates. The reason we chose to add *Inventory* then attach the existing variables as rates was to work through the problem as it came to light, rather than working out the problem first then putting it on the sketch.

Workforce

Now we need to figure out how *production* gets determined. Over the long term investment and capacity are clearly important, but these have been extremely stable. In the shorter term more people are hired and, if necessary, an additional production shift added. There is quite a bit of complicated stuff going on when shifts are added: management changes, maintenance scheduling problems arise, and so on. However, as a first approximation, more people make more products, and this is a good starting point, so we add the level *Workforce*.

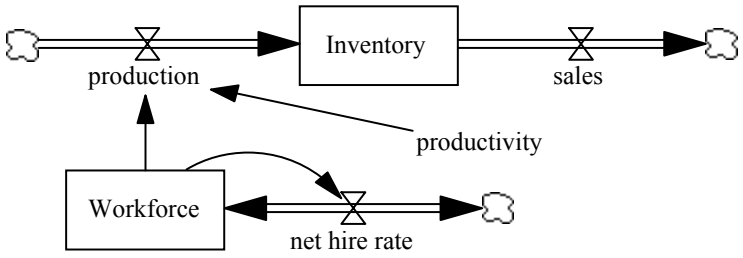
The things that change workforce are hiring, layoffs, firings and retirements. Again, for simplicity we combine all of these into a composite concept — the *net hire rate*. Note that net hire rate can either increase or decrease the workforce.



Behavioral Relationships

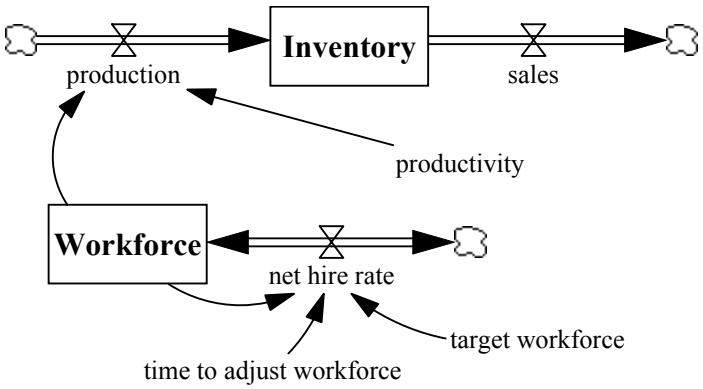
This is the physical part of the problem. Now it is necessary to make some of the behavioral (information) connections. Putting down the important physical stocks and flows is often a good starting point in developing a model. It lets you make part of the system concrete, and this can simplify the conceptualization of other parts of the system. Alternative approaches include building a causal loop diagram and converting that to stock and flow form, or writing equations directly from causal loop diagrams. You might also want to draw a causal loop diagram, then start over again with a stock and flow diagram. What works best varies by individual and by problem, so we try to present some alternative approaches in different chapters in this guide.

In completing the information connection, we will try to keep things as simple as possible. Starting with production we want to remove all the complexities of adding shifts and mothballing equipment and simply state that *production* is proportional to *Workforce*. We add the proportionality constant *productivity*. Also, *net hire rate* is dependent on the value of *Workforce*. This gives us:

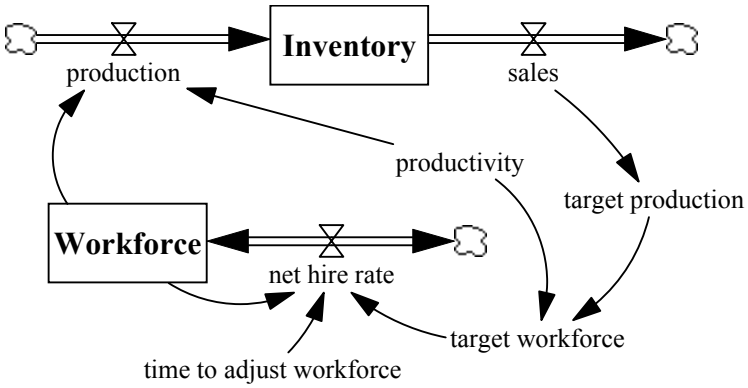


The net hire rate is the net number of people hired. The most straightforward way to formulate this is as a stock adjustment process. In a stock adjustment process you take an existing value of a variable (usually a stock) and compare it to some target or desired level, then take an action based on the difference between the two. For example if you are driving a car at 40 MPH and wish to be going 50 MPH you would depress the accelerator. Your car's speed (a stock) will increase at a rate that depends on how far you depress the accelerator and the car you are driving.

To capture this stock adjustment process we add in the variables *target workforce* and *time to adjust workforce* and connect them as shown :



time to adjust workforce represents the time required for management to agree on a change in the workforce level and screen potential applicants or notify workers to be laid off. *target workforce* is the number of people you need to produce the amount you want to produce. The Level *Workforce* is initialized at this value. Now we add the concept of *target production*, and connect it to *target workforce*. We will set *target production* on the basis of *sales*.



This is a complete model, though it does have a critical error of omission which will be brought to light during simulation. The next step is to take the conceptual model and turn it into a simulation model.

Equation Set wfinv1.vmf

The full equation set for this model follows:

```
FINAL TIME  = 100
Units: Month

INITIAL TIME  = 0
Units: Month

TIME STEP  = 0.25
Units: Month

SAVEPER  = TIME STEP
Units: Month

Inventory = INTEG(
   production-sales,
         300)
Units: Frame

net hire rate = (target workforce-Workforce)/time to adjust
```

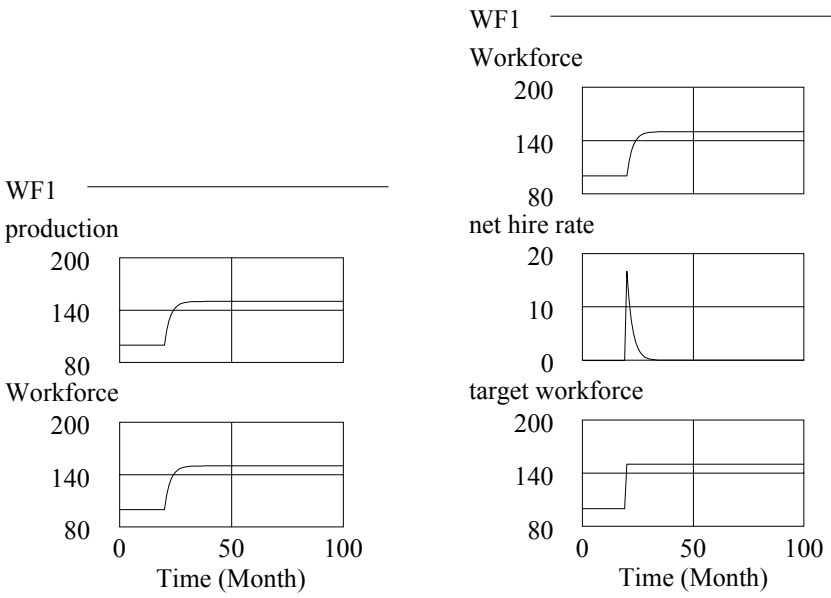
```
   workforce
Units: Person/Month

production = Workforce*productivity
Units: Frame/Month

productivity = 1
Units: Frame/Month/Person

sales = 100 + STEP(50,20)
Units: Frame/Month

target production = sales
Units: Frame/Month

target workforce = target production/productivity
Units: Person

time to adjust workforce = 3
Units: Month

Workforce = INTEG(
   net hire rate,
         target workforce)
Units: Person
```

Each equation is consistent with discussion during the initial conceptualization. The equation for *sales* has *sales* steady at 100 until time 20, when *sales* step up to, and thereafter remain at, 150. This input pattern is used to test that the system is indeed at an equilibrium, and then check the adjustment to a new operating condition. This step input pattern is the cleanest pattern available for looking at the internally generated dynamics of a system such as this. It allows you to observe how a single shock propagates through the system without further external disturbance.

The model is complete. A simulation is performed with the name WF1.

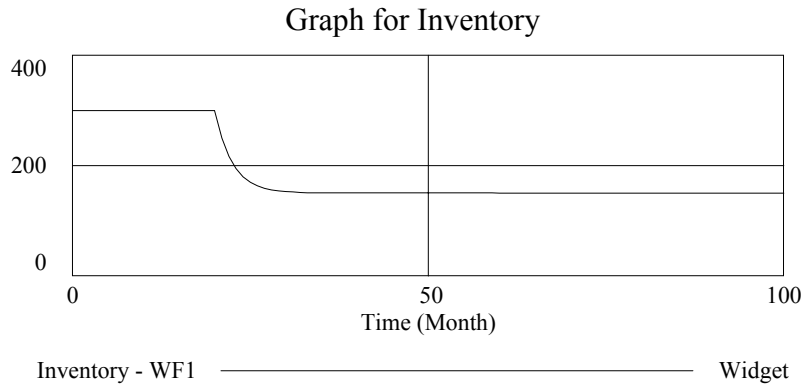
Analysis

A Causes Strip graph for *production* and causes strip of *Workforce* show the dynamics.



It is immediately obvious that there is not a lot of variability in *production* or *Workforce*. There is very smooth adjustment from the initial 100 Widgets/Month to 150 Widgets/Month. Unlike our

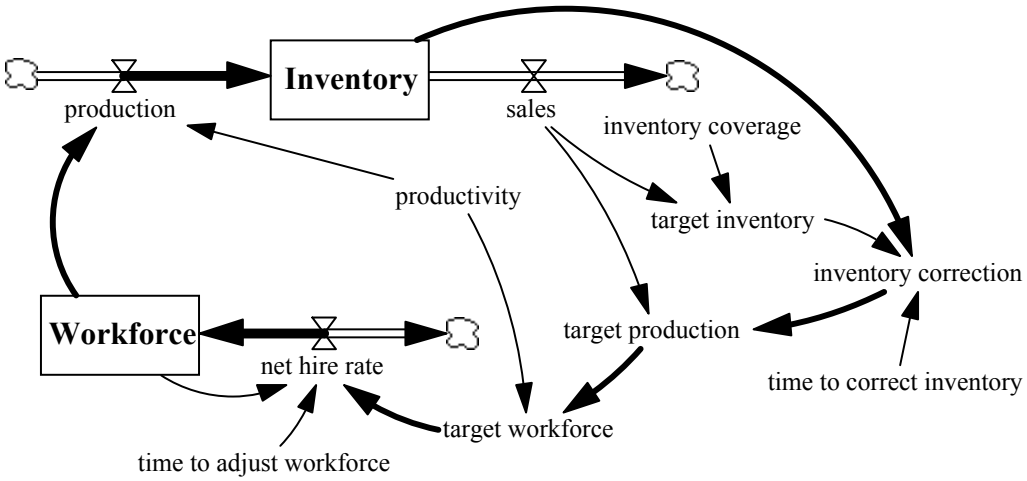
reference mode, the model does not appear to generate more variability in production than sales. At this point it is worth taking a look at *Inventory*.



Inventory falls smoothly from its initial value of 300, to about 150. Since the purpose of holding an inventory is to be sure that the right product configuration is available for customers, there is clearly something wrong. Any discrepancy in *Inventory* from the level necessary to meet product mix requirements and have a comfortable safety stock needs to be corrected, and this model does not make that correction.

Model Refinement (wfinv2.vmf)

In order to refine the model we introduce *target inventory*, *inventory correction* and two additional Constants. The idea is simple — *target inventory* is the amount of stock that should be held based on expectations about sales. The *inventory correction* is the correction for a deviation of *Inventory* from its target. A new loop has been introduced and is highlighted.



Additional Equations

```
target production = sales + inventory correction
Units: Frame/Month

inventory correction = (target inventory - Inventory)/
                TIME TO CORRECT INVENTORY
Units: Frame/Month
```

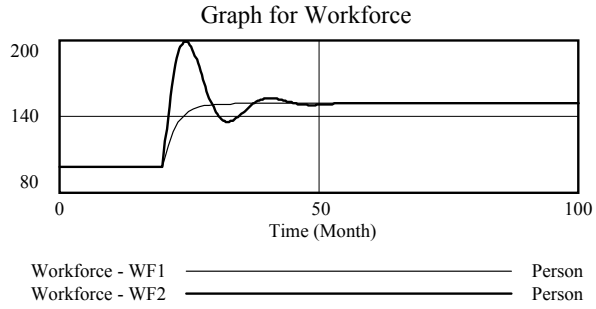
```
TIME TO CORRECT INVENTORY = 2
Units: Month

target inventory = sales * INVENTORY COVERAGE
Units: Frame

INVENTORY COVERAGE = 3
Units: Month
```

inventory correction is a stock adjustment formulation, just as *net hire rate* was. The *time to correct inventory* represents the time required to notice significant changes in inventory and schedule corrections in production.

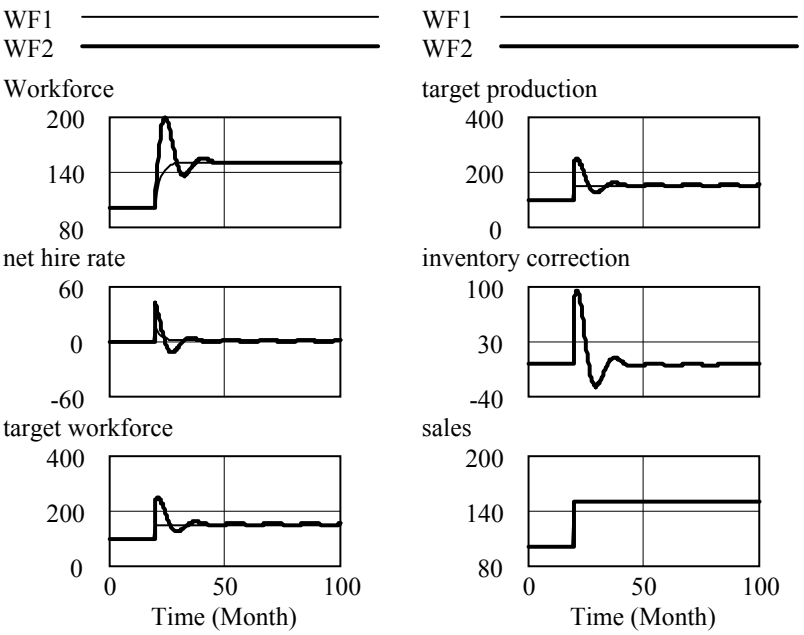
The important difference between this formulation and that of *net hire rate* is that the *net hire rate* directly influences the stock it is attempting to adjust (*Workforce*) whereas *inventory correction* influences *target production*, *net hire rate*, *Workforce*, *production* and finally *inventory*. This connection has an intervening level, *Workforce*, which has important dynamic consequences.

Refined Model Behavior

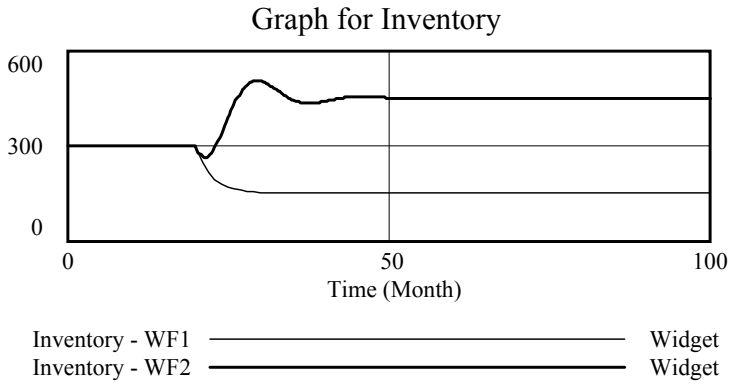
The model is simulated and the run named WF2. First generate a graph of behavior for *Workforce* with datasets from both runs loaded WF1 and WF2.



The behavior is dramatically different from the earlier model. *Workforce* is less stable and there is oscillation. Causal tracing is performed on *Workforce*, and on *target workforce* (graph not shown) and *target production*, and the output shown below:

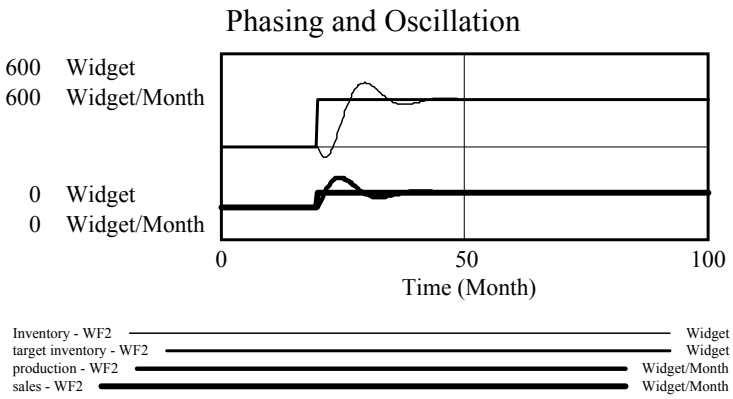


inventory correction was not in the first model. Therefore there is no plot from WF1 for *inventory correction*. All the oscillatory behavior in *target production* is due to *inventory correction*. A graph of *Inventory* shows similar oscillation, but a different final value from the first run. When sales increase, inventory now eventually adjusts to a new, higher desired level, rather than falling as before.

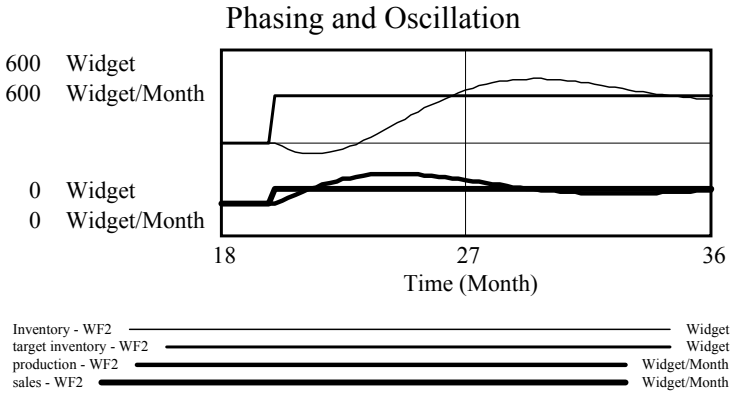


Phasing and Oscillation

To get some useful insight into this model, and oscillations in general, use the Graphs tab in the Control Panel to create a graph containing *Inventory*, *target inventory*, *production* and *sales*. Set the scales to zero minimum and 600 maximum values for all variables. You will also need to go the Datasets tab of the Control Panel and put WF2 first.



We want to focus in on the timing of changes just after the jump in sales. Hold down the shift key and dragging across the portion of the graph of interest, or go to the Time Axis tab of the Control panel, to set the time focus from approximately 18 months to 36 months. Now the custom graph generates this output:



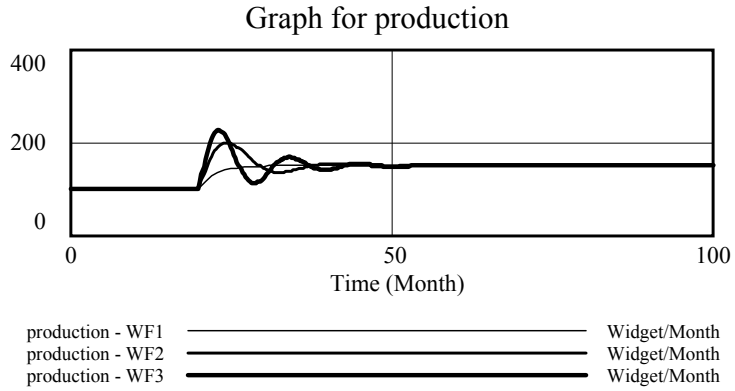
Initially, when *sales* step up, inventory begins to fall because *sales* exceed *production*. *production* gradually increases because *target production* increases with *sales* and *inventory correction* increases as inventory falls. However, there is a delay in this process due to the time required to hire new workers. By month 21.3, *production* is equal to *sales*, so *inventory* stops falling and then begins to grow.

While *Inventory* is increasing, so is *production*. As long as *inventory* is below *target inventory*, there is pressure to increase *production*, even when *production* is already above *sales*. In fact, *production* needs to get sufficiently above *sales* so that the ongoing difference balances the pressure from *inventory correction* before *net hire rate* will go negative. Note that when *Inventory* is equal to *target inventory*, *production* is still higher than *sales* because the workforce is excessive, causing an ongoing increase in *Inventory*. The variability in *production* is now greater than that in *sales*; the model is exhibiting the kind of amplification we identified in the reference mode.

In fact, the amplification of variability from sales to production is physically inevitable. The change in production must exceed the change in sales for some time in order to replace inventory lost before production can adjust and to adjust inventory to the new, higher target level.

Sensitivity

It is interesting to experiment with the model by changing different parameters. One possible policy to improve the company's performance would be to correct deviations in inventory more aggressively. We can simulate this by using a decreased value for *time to correct inventory*. Reducing *time to correct inventory* from 2 months to 1 month, we get the simulation output for *production*:



Here we find that reducing the delay in correcting inventory actually increases the amplitude (and decreases the period) of the oscillations, not necessarily a desirable thing for a firm to experience!

Having worked through the modeling exercises the explanation for this response is quite intuitive. The system needs to overshoot in order to build inventory. Trying to rebuild inventory more quickly will increase the size of that overshoot. Because of the delays involved that overshoot will also lead to an overbuilding of inventory which will require a decrease in production. Because the overshoot is bigger, the decrease will also be bigger.

Extensions and Exercises

If you look at the model structure we have developed you will see that it fails the Reality Check "Without Inventory we can't ship." Try to develop a structure to correct this flaw. You will need to create an explicit shipment variable and use a nonlinear function (Lookups are often used here) to constrain shipments.

3 Project Dynamics

A variety of activities, from writing term papers to building nuclear power plants, have very similar dynamic characteristics. These projects have an initial goal, a lot of work towards the goal, and more or less successful completion of the goal. Projects often fail to achieve the expected goal, and cost overruns, time overruns, and poor quality, are common.

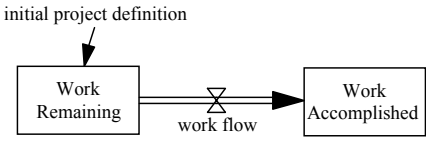
In this chapter, we will develop a model to help understand the processes involved in getting a project done. This project model focuses on the design of a new building, although the model is directly applicable to other activities such as designing a new product, preparing a presentation, developing software, or building a space shuttle.

In conceptualizing and creating this model, we will use an iterative approach. We will start with the simplest structure that is relevant to this problem, and continually refine it. The iterative technique is useful because it prevents a situation in which you have completed a large, complex model, but the simulation results do not make sense. You will be simulating at every step, and seeing the effects of new model structure *as it is added*.

In developing this model, we will be depending on the computer to continually give us feedback on the consequences of changing structure. While the computer is good at this, it is also important to think about what we are doing. Before any simulation experiment is run you should ask yourself what you expect the results to be. If you are surprised, find out why. If you are right, make sure it is for the right reasons. To keep yourself honest, it is helpful to write down predictions about each run beforehand; many experienced modelers keep a notebook documenting model changes, surprising behavior, and important insights as they work.

Task Accomplishment (project1.mdl)

The most fundamental characteristic of any project is that there is something to do and it gets done, at least partially. We start with two Levels and the flow between them:



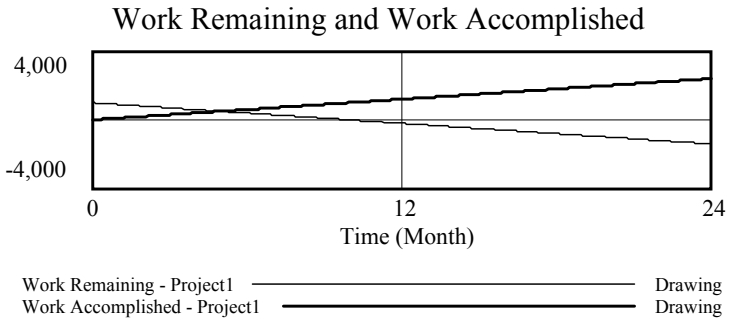
This is a complete model. We set *initial project definition* at 1000 and *work flow* at 100. This is a project that should take 10 months. Just to make sure we have time to get done we run the project for 24 months with TIME STEP at .0625 (this is 1/16th of a month and should be sufficiently fine to trap changes in project activity). The equations are:

```
Work Remaining = INTEG(
   -work flow,
         initial project definition)
Units: Drawing

Work Accomplished = INTEG(
   work flow,
         0)
Units: Drawing

initial project definition = 1000
Units: Drawing
```

```
work flow = 100
Units: Drawing/Month
```

The units of measure — *Drawing* — are appropriate for the design of a building. Tasks, lines of code, or other measures could also be used. It is important to remember that we are dealing with an aggregate representation of a project. In reality a project is not made up of a bunch of equally sized components, but a combination of big and little things. We are representing *averages* in our equations. It is very important to keep this in mind, especially when you are working on a real problem.

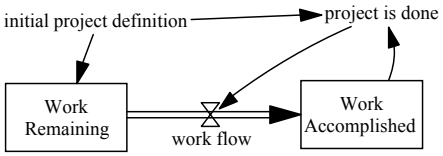
When we simulate this model we see the following:



After 10 months *Work Remaining* is 0, but things continue to get done. There is no logic for shutting the project down.

Stopping Work (project2.mdl)

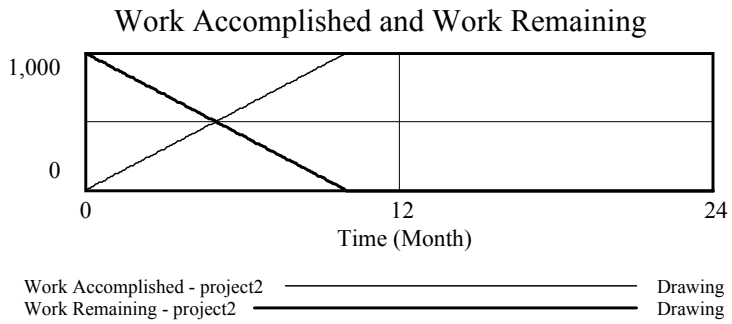
There are two reasonable ways to shut down the project model. One is simply to stop simulating when the project is done. You can do this by writing a different equation for FINAL TIME. The approach we will take here is to stop activity on project completion. We add in the concept *project is done* and make a connection from that to *work flow*.



The dependence of *project is done* on *Work Accomplished rather than Work Remaining* will provide us with a mechanism for descoping the project if there are schedule or budget problems. We add the equations:

```
project is done = IF THEN ELSE(Work Accomplished >=
        initial project definition,1,0)
Units: Dmnl

work flow = IF THEN ELSE(project is done,0,100)
Units: Drawing/Month
```

This yields the results:



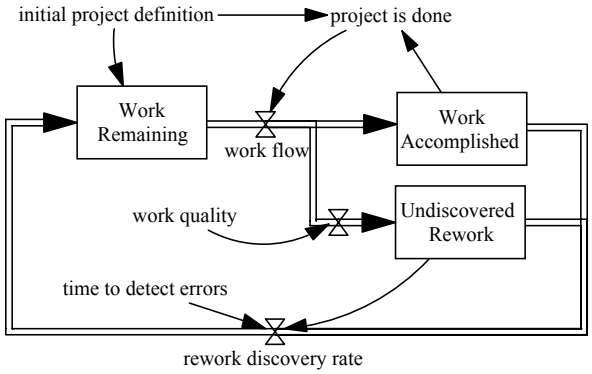
Finished on time, and on budget. A project manager's dream, and the basic model used in most project management software.

Integration Techniques

Before we proceed, a note on integration techniques is in order. In this model we have introduced an abrupt work stop. It is best, with project models, to stick to Euler integration. A consequence of this, which we will see later in this model, is that there will usually be some overshoot. We might do 101% of the work, for example. These overshoots are not problematic, and are of little conceptual significance, though for presentation reasons it is sometimes desirable to correct them. We will, in this chapter, focus on the important conceptual problems, not the details of computation. See Chapter 8 for more detail on integration methods.

Errors and Rework (project3.mdl)

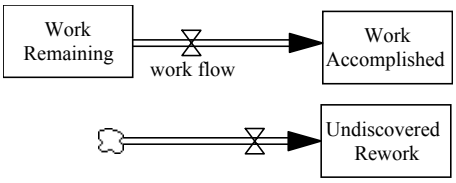
So far we have assumed that the work being done is being done without error. In general this is not true. There are a number of places where errors can occur, including miscommunication among personnel, technical oversights and just plain mistakes. When errors occur they are not, however, immediately discovered. Errors remain undetected until there is a review or integration activity that brings them to light.



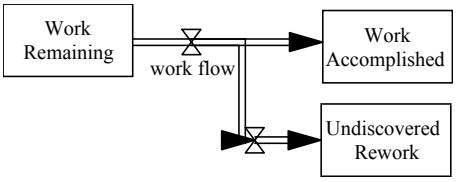
We have set up the diagram to show a parallel flow of work with errors accumulating into *Undiscovered Rework* and a flow back from *Work Accomplished* into *Work Remaining.* (An alternative formulation would be to split the workflow into good and bad and then have the bad work flow back to the work to be done. If we did this, the Work Accomplished concept would need to be replaced by the sum of the good and bad work.)

Drawing the Diagram

To draw the above diagram you need to go through several steps. Working only on the Levels and flows start with:

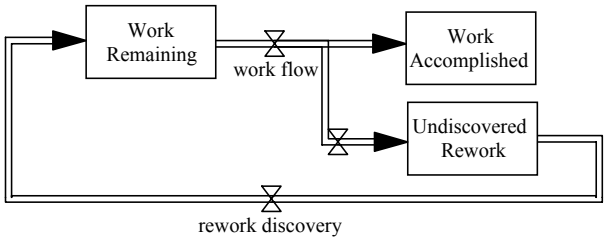


To create the unnamed valve just press the Esc key instead of typing in a name for it. Use the delete tool to remove the cloud from the diagram. You may also need to move the unnamed valve to the right of where it is first placed as shown above. Now select the Rate tool and click on the valve labeled *work flow*. Move to the right (but not down) and, holding down the shift key, click again, now move straight down to the same vertical position as the unnamed valve and, holding down the shift key, click again. Finally release the shift key and click on the unnamed valve. You should have:



What you have may not look quite so bad depending on the relative placement of the two valves. To remove the unwanted arrowhead Right-Click or Control-Click on the arrowhead and uncheck the Arrowhead checkbox in the Arrow Options dialog.

Now draw the rate from *Undiscovered Rework* to *Work Remaining* by clicking on *Undiscovered Rework* then, holding down the Shift Key clicking straight to the right, and again to the right and down. Move left to the position you want the *rework discovery* rate to appear and Click while holding down the Control key. Now move straight to the left and, holding down the Shift key, Click. Move straight up to the position of *Work Remaining* and, holding down the Shift Key, click again. Finally, let go of the Shift key and click on *Work Remaining*. Name the valve *rework discovery*. You should have:



To create the final pipe use the rate Tool to start from *Work Accomplished*. Move directly right to the same horizontal position that the existing pipe turns at and, holding down the shift key, click. Now move down to the bottom turn of the existing pipe and, holding down the shift key, click Finally release the Shift key and click on the valve above *rework discovery*.

You will probably need to select the Move/Size tool and drag the pipes around a little bit to get them to line up properly.

NOTE When you draw the last pipe from *Work Accomplished* to *rework discovery* Vensim, detecting a flow from a Level to a Rate, reverses the direction of causality and removes the arrowhead. Thus, you could have gotten the same thing by starting at *rework discovery* then unchecking the Arrowhead box on the Arrow Options dialog.

Putting it Together

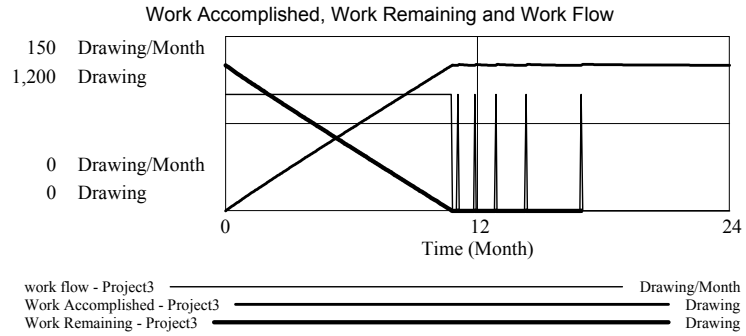
With the diagram in place we need to modify a number of equations, and add some others. The changed and new equations are:

```
rework discovery rate = Undiscovered Rework/time to detect errors
Units: Drawing/Month

time to detect errors = 3
Units: Month
```

The selection of a value for *time to detect errors* requires consideration of the different types of errors that occur and who is likely to find them. This is unlikely to be truly constant over the whole project, and we will address this later on. For the time being, an average of 2 to 3 months seems reasonable.

```
Undiscovered Rework = INTEG(
   work flow*(1-WORK QUALITY)- rework discovery rate,
         0)
Units: Drawing

Work Accomplished = INTEG(
   work flow - rework discovery rate,
         0)
Units: Drawing

WORK QUALITY = 0.9
Units: Dmnl

Work Remaining = INTEG(
   rework discovery rate - work flow,
         initial project definition)
Units: Drawing
```

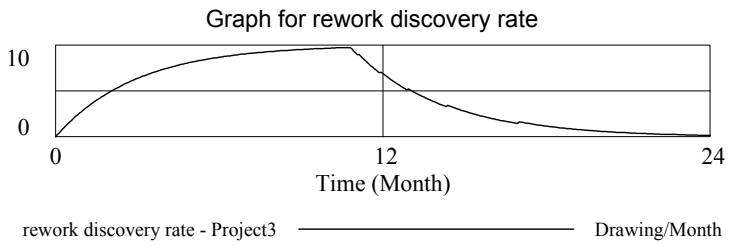
With these additions we get the following behavior:



There are two things to notice. First the project takes a little longer, because not all work is done correctly the first time. Secondly, after the project ends there are a number of repeated resurgences of activity as undiscovered rework comes to light and needs to be redone.

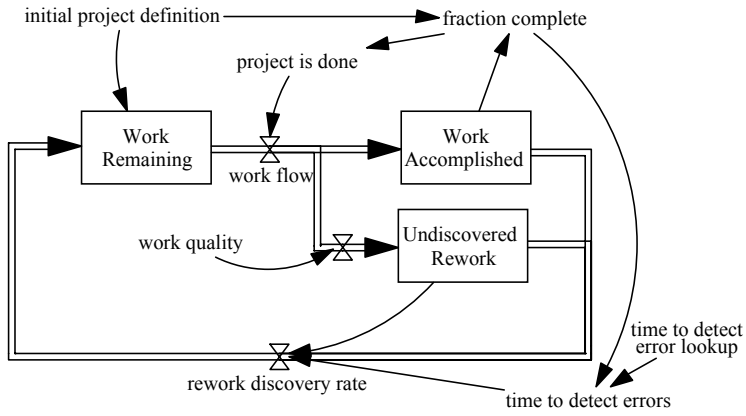
Rework Discovery (project4.mdl)

If we look at `Undiscovered Rework`, we see that it is peaking as the project ends:



While *Undiscovered Rework* is inherently unobservable, it is true that the final stages of a project tend to see a big increase in rework discovery. It is very much like finally putting the pieces of a puzzle together: At the end it becomes quite obvious which pieces are missing or the wrong shape. Examples of problems that can occur are plumbing systems that depend on nonexistent access corridors and ventilation shafts that are the wrong size for the planned equipment.

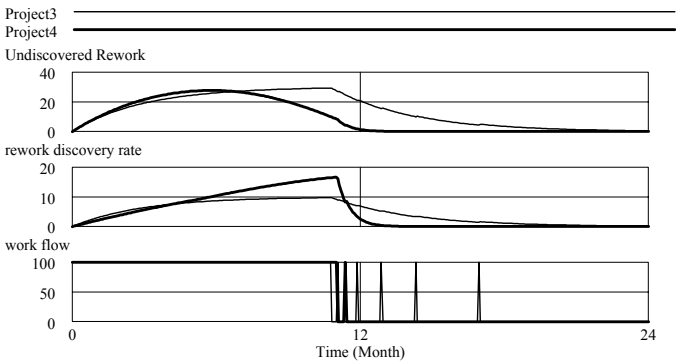
We can represent these concepts simply by making *time to detect errors* depend on the state of completion of the project.



Here we have also added in the variable *fraction complete* and changed *project is done* so that it depends on the new variable. The new and changed equations are:

```
fraction complete = Work Accomplished/initial project definition
Units: Dmnl

project is done = IF THEN ELSE(fraction complete >= 1,1,0)
Units: Dmnl

time to detect errors = time to detect error lookup(
        fraction complete)
Units: Month

time to detect error lookup ((0,5),(0.5,3),(1,0.5))
Units: Month
```

We are using a lookup function on fraction complete to drive *time to detect errors* to a much smaller value toward the end of the project. Simulating now gives us:



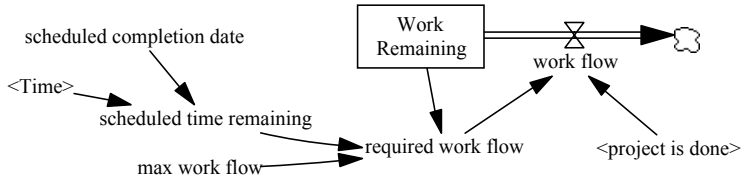
The project ends slightly later, but there is very little rework remaining at the time of completion.

You will notice during simulation that you receive warning messages about being above *time to detect error lookup* computing *time to detect errors*. This happens because the amount of work done slightly exceeds the original amount of work to do. This is something that was expected because of the discontinuous way the project is stopped. This not a conceptual problem so we leave it alone for now.

Schedule (project5.mdl)

So far we have been representing *work flow* as a constant with no feedback from schedule. The purpose of project management is to keep projects on schedule. To do this it is necessary to know what the schedule is, and adjust resources to meet that schedule.

If we have a completion date scheduled, we can compute the time we have left. Since we know the amount of work remaining, we can further determine how fast we need to work to meet the schedule. For compactness and clarity, we will represent only the structure that is changing from now on.



The equations for this are:

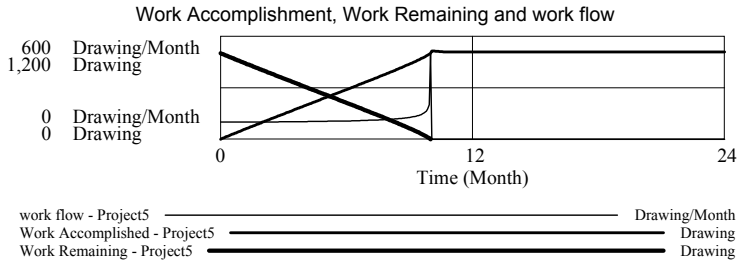
```
required work flow = MIN(max work flow,
       XIDZ(Work Remaining,scheduled time remaining,max work flow))
Units: Drawing/Month
```

This formulation keeps `required work flow` less then *max work flow* using the MIN (minimum) function. The XIDZ (X If Divide by Zero) function divides *Work Remaining* by *scheduled time remaining* unless *scheduled time remaining* is 0, when it returns *max work flow*. It is important to use this function since dividing by zero makes no sense and will be reported as an error by Vensim.

```
scheduled time remaining = MAX(0,scheduled completion date - Time)
Units: Month

scheduled completion date = 10
Units: Month

max work flow = 500
Units: Drawing/Month

work flow = IF THEN ELSE(project is done,0,required work flow)
```

```
Units: Drawing/Month
```

Simulation of the model now gives:

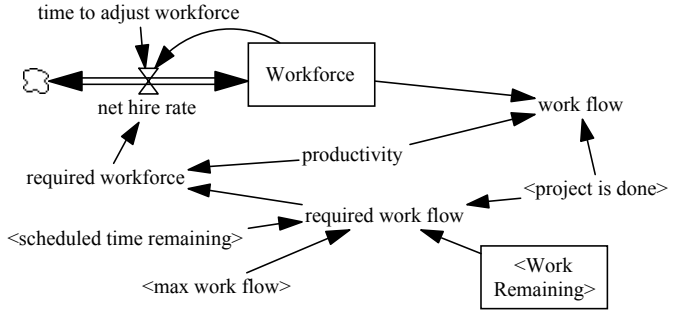


The project is completed on time. *work flow* drifts slowly upward, then shows a sudden jump right at the end to finish the project on time.

Workforce and Hiring (project6.mdl)

So far we have treated *work flow* as whatever is necessary to get the work done, subject to limitations. In order to get work done, however, resources are necessary. In this model we are focusing on effort, and resources are people. Whether these people are hired for the job, or assigned internally within an organization, they are required to get the job done.

The simplest formulation for workforce is the same as that used in the Workforce-Inventory model. In this case the desired amount of production is that required to complete the project on time:



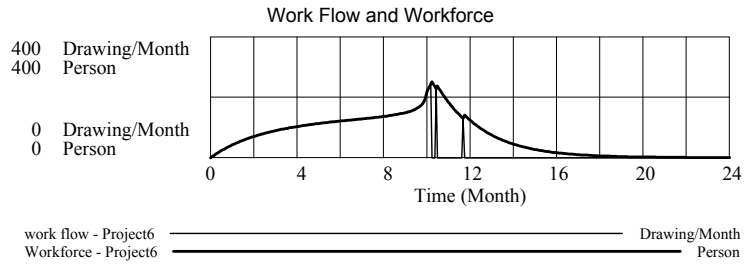
The formulation for *required workflow* has been changed so that it will go to 0 when the project is completed. The new and changed equations are:

```
net hire rate = (required workforce - Workforce)/time to adjust
   workforce
Units: Person/Month

Productivity = 1
Drawing/Person/Month

required work flow =IF THEN ELSE(project is done,0,
   XIDZ(Work Remaining,scheduled time remaining,max work flow))
Units: Drawing/Month

required workforce = required workflow/productivity
Units: Person

time to adjust workforce = 2
Units: Month

Workforce = INTEG(
   net hire rate,
```

```
          0)
Units: Person
```

The project starts off without anyone working on it, and people are brought on relatively quickly to get the work done. This is the most straightforward view of labor requirements and acquisition. It would also be possible to use a planned work intensity profile, involving a gentle ramp up of effort at the beginning and a ramp down at the end.

The model modified in this way produces:

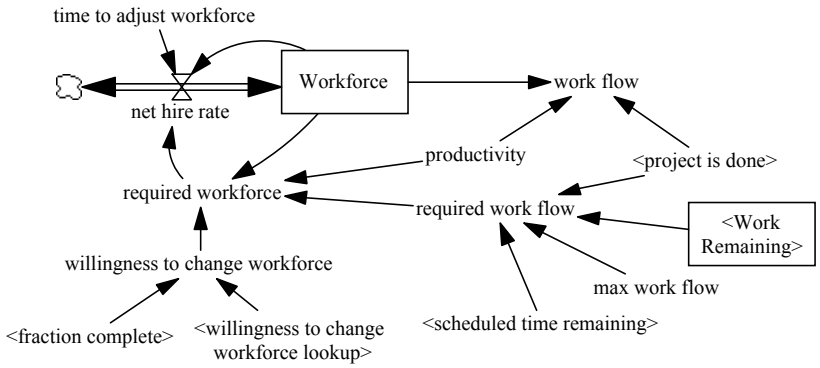


Workforce quickly climbs to 100 people, then climbs slowly until it starts a dramatic increase near the end of the project. There is also a resurgence of project activity after the end of the project, we deal with both of these unrealistic behaviors together.

Willingness to change workforce (project7.mdl)

As the end of the project approaches it is unlikely that more people will be added. As the project progresses the project team tends to stabilize in terms of positions and activities, and, once the project is 80% complete, it is rarely appropriate to make further changes in staffing levels.

We capture this dynamic using *willingness to change workforce*.



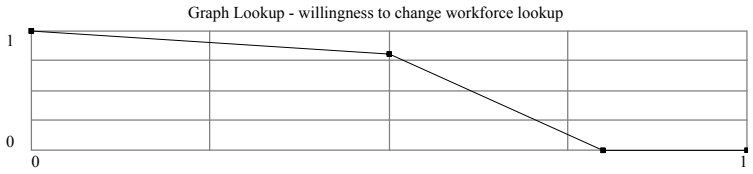
With the new and changed equations:

```
required workforce = IF THEN ELSE(
  Workforce < required work flow/productivity,
  willingness to change workforce*required work flow/productivity +
      (1-willingness to change workforce)*Workforce,
  required work flow/productivity)
Units: Dmnl
```

This equation allows workforce reductions, but does not allow workforce increases late in the project.

```
willingness to change workforce =
         willingness to change workforce lookup(fraction complete)
```

```
Units: Dmnl
```




```
willingness to change workforce lookup
                ((0,1),(0.5,0.8),(0.8,0),(1,0) )
Units: Dmnl
```

Project Restarts

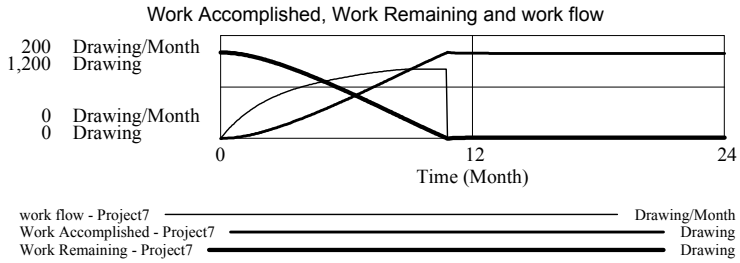
It is possible for a project to start up again after an apparently successful completion because of the discovery of a number of problems. Such a situation is, however, somewhat uncommon and occurs only after significant problem discoveries.

For the model we are working with here this problem is not serious, since it is the dynamics before the project is completed that are of interest. In multistage projects, where more than one activity is taking place and there is resource sharing between activities, restarting a task can be a significant problem.

The way to represent this is almost the same as the formulation of the thermostat problem discussed in Chapter 8. If the project has been declared complete, we do not restart it unless things get very bad.

```
project is done = IF THEN ELSE(project was done :AND:
     fraction complete > restart fraction,1,
     IF THEN ELSE(fraction complete >= 1,1,0))
Units: Dmnl

project was done = DELAY_FIXED(project is done,0,0)
Units: Dmnl

restart fraction = 0.9
Units: Dmnl
```

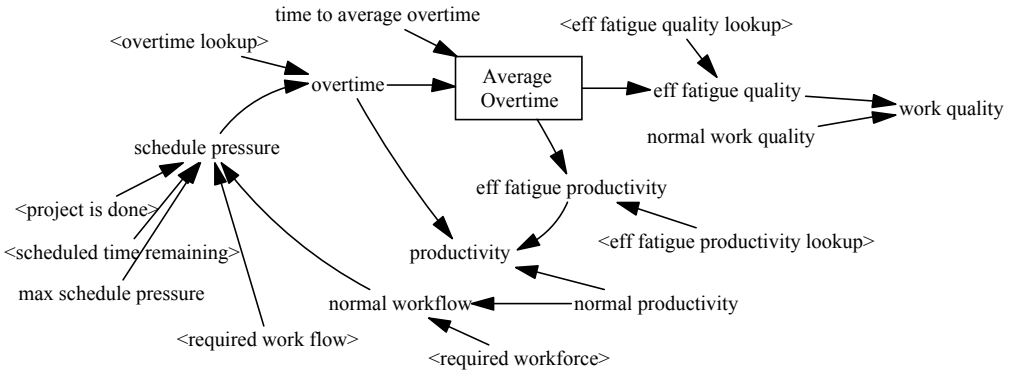
Resulting Behavior



The project is a finishing a little later, with activity flat toward the end. After the project is finished there is still a small amount of rework discovered. This never gets large enough to trigger a project restart and remains unaddressed.

Schedule Pressure (project8.mdl)

By introducing the effective freeze on workforce toward the end of the project we have removed the influence of the schedule on activity. In the final stage of the project people simply continue to work at a normal speedand finish when they finish. While staffing levels may be constant, it is rarely true that the intensity of effort is constant.



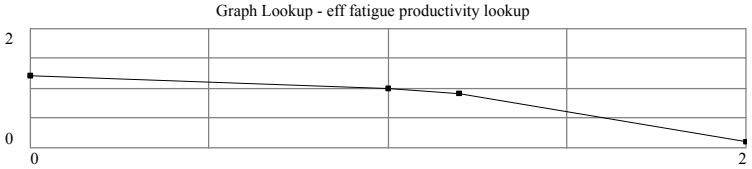
Here we use *schedule pressure* to drive *overtime*, with *overtime* directly affecting *productivity*. *Average Overtime* is used as a measure of fatigue resulting from prolonged overwork. Fatigue lowers productivity and quality.

The new and changed equations are:

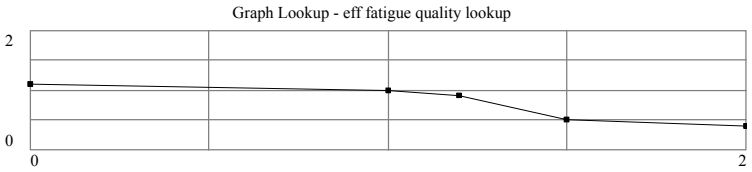
```
Average Overtime = INTEG(
    (overtime - Average Overtime)/time to average overtime,
        overtime)
Units: Dmnl
```

Note that this formulation is the same as *Average overtime = SMOOTH(overtime, time to average overtime)*. As a general practice, the use of dynamic functions such as SMOOTH is avoided because they can introduce behavior that is difficult to determine the causes of. It is good practice to use dynamic function only where they result in substantially less clutter.

```
eff fatigue productivity = eff fatigue productivity lookup(
                Average Overtime)
Units: Dmnl
```




```
eff fatigue productivity lookup((0,1.2),(1,1),(1.2,0.9),(2,0.1))
Units: Dmnl

eff fatigue quality = eff fatigue quality lookup(Average Overtime)
Units: Dmnl
```

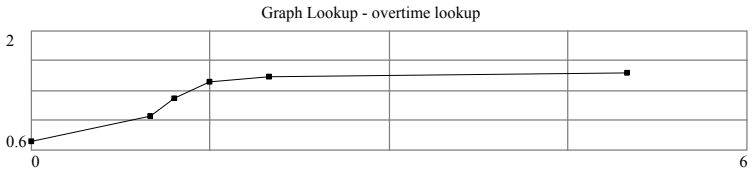



```
eff fatigue quality lookup((0,1.1),(1,1),(1.2,0.9),(1.5,0.5),
          (2,0.4) )
Units: Dmnl

max schedule pressure = 5
Units: Dmnl

normal productivity = 1
```

```
Units: Drawing/Person/Month

normal work quality = 0.9
Units: Dmnl

normal workflow = MIN(max work flow, normal productivity * required
   workforce)
Units: Drawing/Month

required workforce = IF THEN ELSE(
   Workforce < required work flow/normal productivity,
   willingness to change workforce*required work flow/ normal
   productivity + (1 - willingness to change workforce) * Workforce,
   required work flow/normal productivity)
Units: Person
```

Note that this formulation has been changed to use *normal productivity* instead of *productivity*. If this were not the case, you would have a situation in which people were producing above average because of overtime, but this high output was considered the norm for bringing new people on. Depending on the project, this might actually be realistic. In this case, however, it would be necessary to use not productivity, but a perception of average productivity since productivity is not directly observable.

```
overtime = overtime lookup(schedule pressure)
Units: Dmnl
```

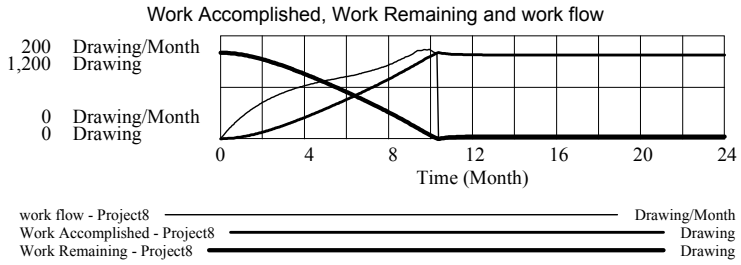



```
overtime lookup((0,0.7),(1,1),(1.2,1.2),(1.5,1.4),(2,1.45),(5,1.5) )
Units: Dmnl

productivity = normal productivity * overtime * eff fatigue
   productivity
Units: Dmnl

schedule pressure = IF THEN ELSE(scheduled time remaining<= 0 :AND:
   :NOT: project is done,max schedule pressure,
   ZIDZ(required work flow,normal workflow))
Units: Dmnl
```

The ZIDZ function is used because at the beginning of the simulation *normal workflow* is 0. Once the schedule is overrun *schedule pressure* goes to its maximum value until the time of completion.

```
time to average overtime = 2
Units: Month

work quality = normal work quality * eff fatigue quality
Units: Dmnl
```

With these changes in place we see the following behavior:

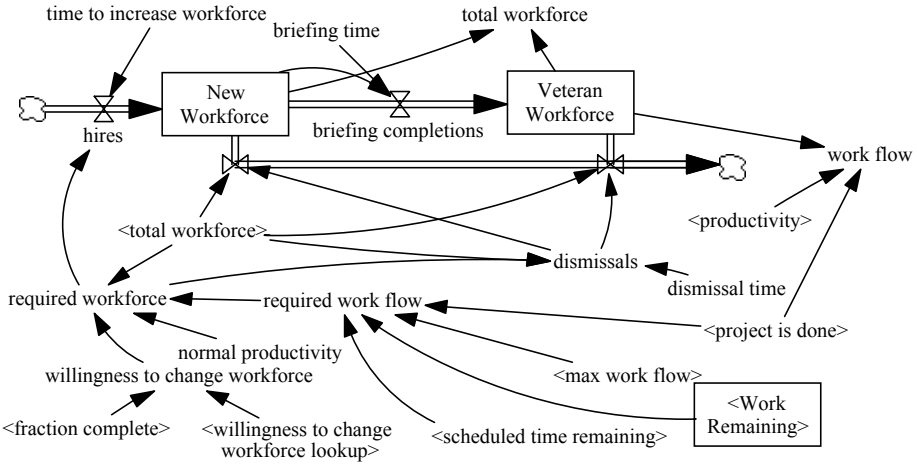


The peak in activity toward the end is accomplished with overtime. The quality toward the end does slip, and there is a small but noticeable amount of rework discovered after the project is completed.

Labor Mix (project9.mdl)

One thing that is immediately noticeable in the above simulations is that even after the project is over there are lots of people still assigned to it. While some amount of wind-down and cleanup makes sense, it is likely to occur quite quickly. This suggests that it is appropriate to have different time constants for the acquisition and dismissal of personnel.

The second thing that warrants consideration is the preparedness of people to do work on the project. We have assumed that every person brought on is immediately productive. While the people being brought on to the project are assumed skilled, they will require some time to be briefed and brought up to speed on their responsibilities.



The workforce is now broken out into two components. People who are hired come into the *New Workforce* pool and after sufficient briefing become veterans. Only veterans are actually contributing to *workflow*, so there is a significant burden to having too many new people. In computing requirement, on the other hand, it is *total workforce* against which a comparison to requirements is being made.

While *hires* flow only into *New Workforce*, dismissals are drawn proportionally from both *New Workforce* and *Veteran Workforce*. *dismissal time* is also set to be significantly shorter than *time to increase workforce*.

The new and changed equations are:

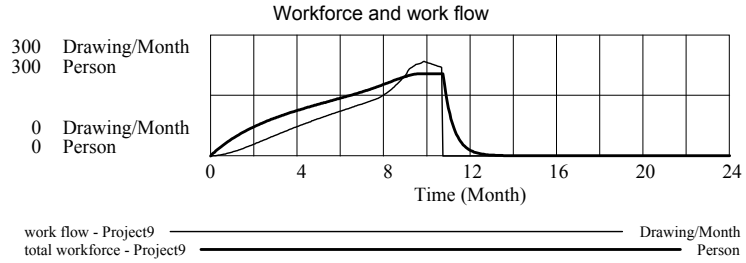
```
briefing completions = New Workforce/briefing time
Units: Person/Month

briefing time = 2
Units: Month
```

```
dismissal time = 0.5
Units: Month

dismissals = IF THEN ELSE(required workforce < total workforce,
   (total workforce - required workforce)/dismissal time,0)
Units: Person/Month

hires = IF THEN ELSE(required workforce > total workforce,
  (required workforce-total workforce)/time to increase workforce,0)
Units: Person/Month

New Workforce = INTEG(
   hires-briefing completions-dismissals * ZIDZ(New Workforce,total
   workforce),
         0)
Units: Person
```

Note the use of ZIDZ to prevent from dividing by zero when *total workforce* is zero.

```
time to increase workforce = 2
Units: Month

total workforce = New Workforce + Veteran Workforce
Units: Person

Veteran Workforce = INTEG(
   briefing completions - dismissals * ZIDZ(Veteran Workforce,total
   workforce),
         0)
Units: Person
```

And this results in the behavior:



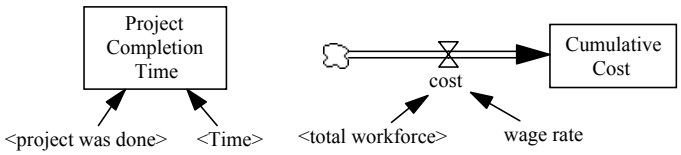
The big things to notice here are the divergence of *work flow* from *total work force*, and the larger number of people eventually needed.

Policy Experiments (project.mdl)

We have, through a series of small steps, developed an interesting project model. We now want to use this model to test some policy ideas. In order to do this we will first add in some accounting equations to measure total cost and final completion date. Then we will formulate an alternative hiring policy to see if performance can be improved.

Accounting Equations

It is often useful to add in summary measurement equations for a model The most interesting measures in this example are total cumulative cost, and project completion time.



With Equations:

```
cost = total workforce * labor cost
Units: $/Month

Cumulative Cost = INTEG(
   cost,
         0)
Units: $

Project Completion Time = SAMPLE IF TRUE(project was done =
   0,Time,0)
Units: Month
```

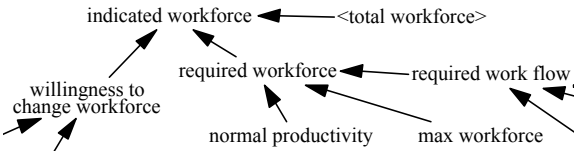
The function SAMPLE IF TRUE is used to return *Time* until *project was done* is true and stay constant thereafter. This formulation will capture restarts that might occur because of low quality.

```
WAGE RATE = 6000
Units: $/Month/Person
```

WAGE RATE includes benefits.

Workforce Cap

As a policy we can experiment with different caps on total staffing. By simple arithmetic with 1,000 drawings to complete and a productivity of 1, 100 people should be able to finish the job in 10 months. There were twice that many people at the end of the last simulation, so maybe we can do better. We want to reformulate *required workforce*, but the current equation for this is already quite involved. To clarify things rename *required workforce* as *indicated workforce*. Now add in a new variable called *required workforce*.



indicated workforce will be used where *required workforce* used to be used and required workforce will be computed differently. Note that when you rename *required workforce* Vensim automatically makes sure it is renamed in the equations it is used in. Thus you only need to write new equations for:

```
indicated workforce = IF THEN ELSE(total workforce <
    required workforce,
    willingness to change workforce*required workforce +
    (1-willingness to change workforce)*total workforce,
    required workforce)
Units: Person

max workforce = 1000
Units: People

required workforce = MIN(max workforce,
     required workflow/normal productivity)
```

`Units: Person`

`max workforce` is initially set very large to allow a non binding policy. If you simulate the model with these changes the same behavior is generated.

Viewing Terminal Values

The two measurements we set up are formulated so that their terminal values are of interest. To review terminal values we will use the Table tool. Set the Time Axis tab of the Control Panel to use only a very small range (see Chapter 12 of the Reference Manual). Clicking on the Table tool generates:

Causes Tab

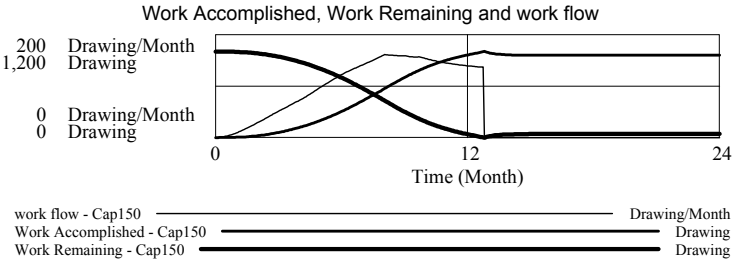
Time (Month)	0	0.0625	24
"Cumulative Cost" Runs: Project			
Cumulative Cost	0	0	8.853 M
"Project Completion Time" Runs: Project			
Project Completion Time	0	.0625	10.75

If you have more than one run loaded you will probably see some blank lines since these variables do not appear in the other models.

The project cost $8.85 million and was completed at time 10.75.

Labor Cap

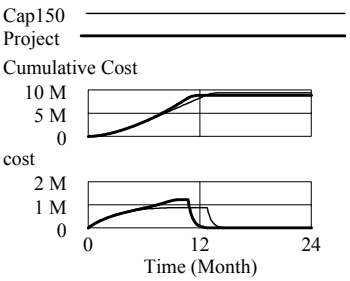
Now do an experimental simulation in which labor is capped at 150. You should get the results:



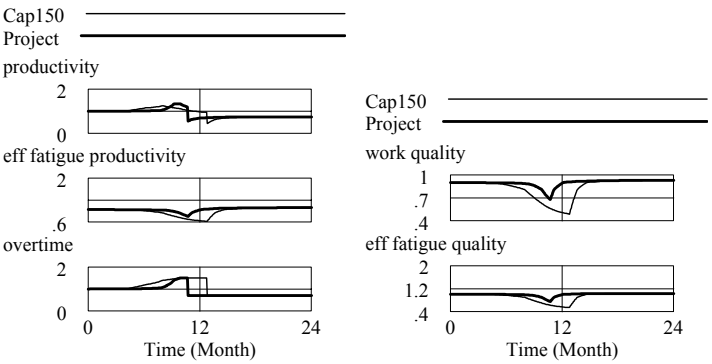
The project stretches out a little bit. If we look at cost, however, the results are a little surprising. *`Cumulative Cost`* is higher with the cap in place. Lets look at why:

Select *`Cumulative Cost`* into the Workbench and click on the Causes Strip tool (be sure to go to the Time Axis Control and click on the Reset button). If you don't have the right runs loaded go to the Datasets Control (Control>Datasets command):

Now clicking on the Causes Strip graph shows the causes of *`Cumulative Cost`*.



Costs do not rise as high, but they continue for longer and add up to more. Now select **`productivity`** and click the Causes Strip graph:



productivity never rises significantly. Schedule pressure comes in early, and leaves a fatigued workforce. Quality is also much lower. As the base model is configured the project finishes in a flurry of activity before anyone has had time to be burned out.

Summary

We have, in this chapter, gone through a step by step process of building a model by continually adding small pieces to an existing model. The advantage of this method is that you always have a working model. The disadvantage is that you can lose sight of the forest for the trees. It is always a difficult problem to determine when to stop adding. If you are not taking a broad view it is possible to add more and more detail without improving understanding. Building in small steps is a powerful approach, but you need review progress from time to time to see if the steps are taking you in the direction you want to go.

4 The Growth of a Field

New products and new technologies, when released into intellectual and commercial marketplaces, can display a wide variety of behavior. There are fads, unmitigated disasters, amazing takeoffs that go bust and, sadly only rarely, real long term roaring successes. Every new product and technology introduced is unique, but some dynamic behaviors are commonplace.

For concreteness we will look at the field of System Dynamics, and attempt to understand the role of a technology, such as Vensim, in the development of the field. We will start with a background description of the problem, put forward a number of hypotheses about what is going on and what policies might be helpful. We will then start building a model to help us test these hypotheses.

The model-building process we will use in this chapter is one of articulating and formalizing theories, and then attempting to incorporate them into a unified model. This process is usually not smooth, since many theories are ill formed and internally inconsistent. Trying to formalize an internally inconsistent theory will illuminate the problem; however, resolving the conflicting ideas so that they make sense can be difficult. Another common difficulty with this model-building process is that two theories can not always fit into a single framework. In some cases, it is easier to develop different models for different theories and then compare the models on the basis of behavior, Reality Checks and data.

Background

In 1956 Jay W. Forrester began applying the principles of feedback and control to the study of economic and management problems. Forrester felt that work in the field was fragmented and focused on problems that would not provide the leverage required to achieve truly superior performance. Thus he pioneered the field of System Dynamics.

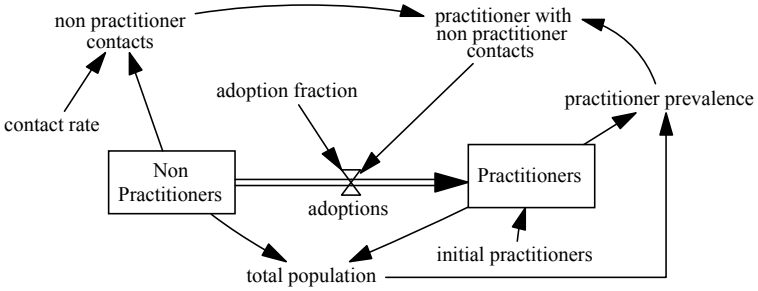
Since 1956 the field has grown, but more slowly than expected by most who have followed the work. In looking at the growth of the field we can generate a number of candidate hypotheses.

Hypotheses

- Growth is exponential but slow (things are good).
- There is a shortage of qualified professors (increase supply).
- There are not enough textbooks (standardize training).
- Only the simplest models can ever be widely disseminated (change goals).
- We need technology to help people understand models (improve accessibility).
- People need to be reeducated to think systemically (increase demand).
- Most people cannot build models (potential market is small).
- We need more skilled practitioners (improve practice quality).
- It is just too hard (make it easier).
- There is a need/skill mismatch (reorient training).
- We need to publicize successful applications (increase marketing).

The Basic Diffusion Process (sdgrow1.mdl)

Regardless of which hypothesis we consider, we need to think about how many people are using the technology. The process starts with the work of several people, then spreads. Holding this very simple thought we can start with:



There are two pools of people: *Practitioners* and *Non Practitioners* (we do our best to keep the language neutral here). These people are all going to conferences, attending meetings and bumping into one another on the street. Since the process of adoption requires that someone instill the idea of the technology into someone else, we are interested in the frequency with which someone who is not a practitioner encounters someone who is a practitioner. Therefore we start with *non practitioner contacts* with anyone, and then multiply by the fraction of people who are practitioners.

practitioner with non practitioner contacts represents contacts between someone who is practicing and someone who is not practicing. There is a chance that, as the result of this meeting, the non practitioner will take up practice. The probability that this happens is called the *adoption fraction*.

The equations for the above model are:

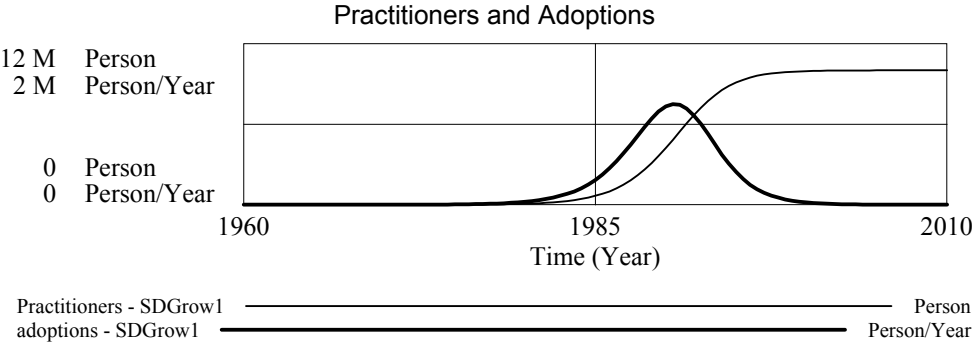
```
adoption fraction = 0.005
Units: Dmnl

adoptions = practitioner with non practitioner contacts *
   adoption fraction
Units: Person/Year

contact rate = 100
Units: 1/Year

initial practitioners = 10
Units: Person

non practitioner contacts = Non Practitioners * contact rate
Units: Person/Year

Non Practitioners = INTEG(
   - adoptions,
        1e+007)
Units: Person
```

Note that we start the model with 10 million people. This is intended to represent the number of academics and skilled professionals for whom this type of work is relevant. This number is an open issue for discussion, though we will not be addressing it in this chapter.

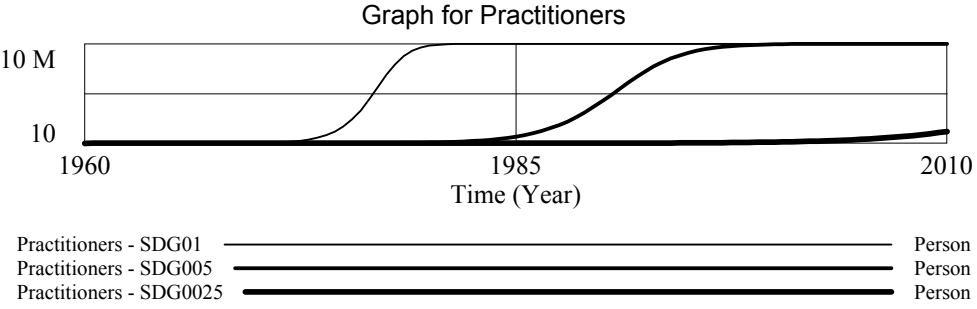
```
practitioner prevalence = Practitioners/total population
Units: Dmnl

practitioner with non practitioner contacts =
```

```
   non practitioner contacts * practitioner prevalence
Units: Person/Year

Practitioners = INTEG(
   adoptions,
         initial practitioners)
Units: Person

total population = Non Practitioners + Practitioners
Units: Person
```

The model is run from the year 1960 to the year 2010 with `TIME STEP` at .125. This model generates the following behavior:



The behavior of the model is quite sensitive to the parameters chosen. For example, if we let *`adoption fraction`* taken on the values .01, .005 and .0025 we get:



For higher values like .01, the number of practitioners grows rapidly and saturates early as the total population adopts the method. On the other hand, for a value of .0025 the number of practitioners barely registers on this scale. One interesting experiment to do is to lower the adoption fraction even further to .001 and see how long it takes before saturation occurs.

A Note on Behavior

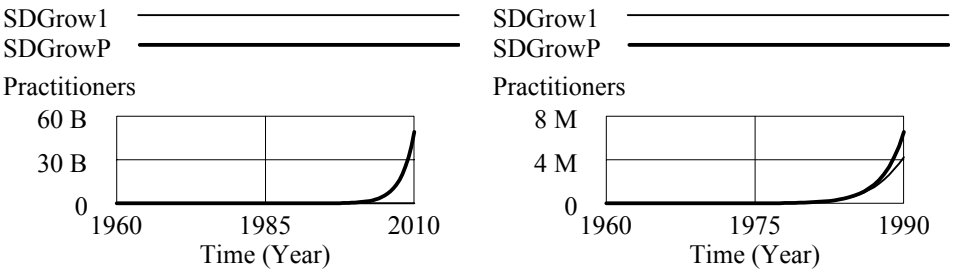
One of the most important features of exponential growth is that there is, seemingly, very little activity for a long period of time, and then an explosion. In this model the explosion is contained because the conserved flow of *`Adoption`* into *`Practitioners`* also depletes the source of growth, the stock of *`Non Practitioners`*. If we break this link in our model, replacing the equation for the level *`Non Practitioners`* with a constant:

```
Non Practitioners = 1E7
```

and, to make it true exponential growth:

`total population = 1E7`

We remove the constraint on growth.



The models diverge after 1989 very dramatically - pure exponential growth makes the diffusion process look like a flat line at 0. Before 1988, however, the two models give very nearly identical results. The graph on the right is a close-up of 1960 to 1990, and there is almost no difference between the two models until near the end of this time.

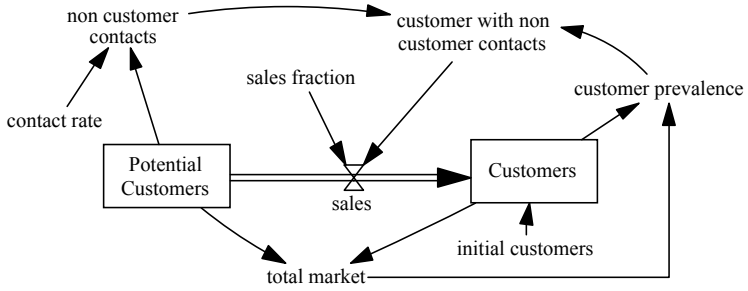
For this model, the implied growth rate is just

`contact rate * adoption fraction = 100 * .005 = .5`

or 50%/year. With *`adoption fraction`* set at .001 there is only a 10%/year potential growth rate, thus the big difference in takeoff.

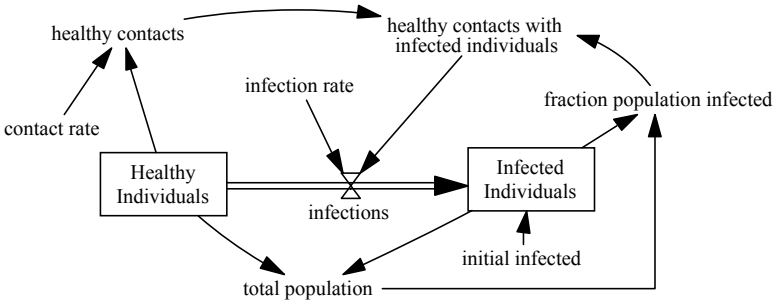
flu.mdl)

The above model is a very generic core for all diffusion processes. Consider new product sales (*sales.mdl*):



`adoptions` has become *`sales`*, and if you recall the behavior of *`adoptions`*, this means that they are small for a long time, then they explode, and immediately collapse. Many companies, such as Atari, have gone through this. Even though the process is generic, they surprised everyone (especially themselves) when their amazing success went bad.

Or consider the spread of a disease (*flu.mdl*)

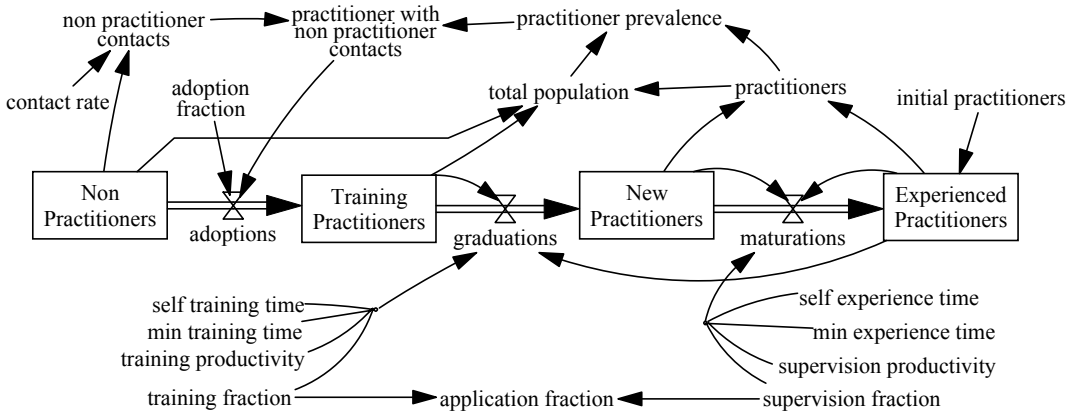


Both of these are exactly the same model, with just the names changed, and both make sense. In order to elaborate either of these you might want to make different changes - adding distinctions between symptomatic, asymptomatic, contagious and non-contagious flu victims, for example. But even as you add structure you will find that strong parallels often remain.

The Adoption Process (sdgrow2.mdl)

We have seen how important the adoption rate is: if it is too low, a technology will take so long to diffuse that it is likely to be lost in the wash of other events and technologies. The way we have modeled it, however, adoption is just a matter of picking up the tool and going to work. This is not, unfortunately, the way life works. After deciding that a technology is good and worth pursuing, it is necessary to spend time and effort to become capable enough to use the technology.

Instead of just looking at *Non Practitioners* and *Practitioners*, we can look at *Non Practitioners*, *Training Practitioners*, *New Practitioners* and *Experienced Practitioners*. *Practitioners* can then be reformulated as the sum of New *Practitioners* and *Experienced Practitioners*. *Experienced Practitioners* can also provide teaching and training to speed the transition from *Training Practitioners* to *New Practitioners* to *Experienced Practitioners.*



This diagram is a little bit busier, but is the same basic structure as the first model. There are six constants that determine the speed with which people can move through training and gaining experience. *self training time* is the time required for a person with no formal training to become sufficiently proficient to be a practitioner. *min training time* is the time required for a person with lots of formal training to become proficient. As *Experienced Practitioners* devote time to training, the average training time moves from *self training time*, to *min training time* according to *training productivity*. The formulation for people becoming experienced is exactly parallel.

The equations for this model are:

```
adoption fraction = 0.005
Units: Dmnl

adoptions = practitioner with non practitioner contacts *
        adoption fraction
Units: Person/Year

application fraction = INITIAL(1 - supervision fraction -
        training fraction)
Units: Dmnl
```

This represents the fraction of time experienced people devote to application (doing research, publishing and helping solve problems) and not training others. This is not used in this model, but will be used in the next refinement.

```
contact rate = 100
Units: 1/Year

Experienced Practitioners = INTEG(maturations,initial practitioners)
Units: Person

graduations =  MIN(Training Practitioners/min training time,
   Training Practitioners/self training time + Experienced
   Practitioners * training fraction * training productivity)
Units: Person/Year
```

Any addition of people devoted to training immediately adds to *graduations* until people are coming out as fast as they can be expected to at which point adding more trainers has no effect.

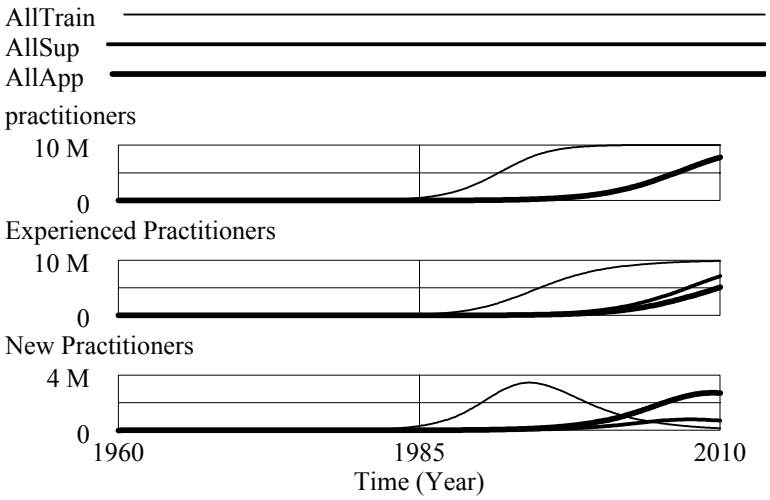
```
initial practitioners = 10
Units: Person

maturations = MIN(New Practitioners/min experience time,
   New Practitioners/self experience time + Experienced
   Practitioners * supervision fraction * supervision productivity)
Units: Person/Year

min experience time = 1
Units: Year

min training time = 0.25
Units: Year

New Practitioners = INTEG(
   graduations - maturations,
         0)
Units: Person

non practitioner contacts = Non Practitioners * contact rate
Units: Person/Year

Non Practitioners = INTEG(
   - adoptions,
         1e+007)
Units: Person

practitioner prevalence = practitioners/total population
Units: Dmnl

practitioner with non practitioner contacts =
   non practitioner contacts * practitioner prevalence
Units: Person/Year

practitioners = New Practitioners + Experienced Practitioners
Units: Person

self experience time = 4
Units: Year

self training time = 2
Units: Year

supervision fraction = 0
Units: Dmnl

supervision productivity = 4
```

```
Units: 1/Year
```

The *supervision productivity* is the number of people per year an experienced practitioner can train. Thus the units are (Person/Year)/Person or 1/Year.

```
total population = Non Practitioners + Training Practitioners +
   practitioners
Units: Person

training fraction = 0
Units: Dmnl

Training Practitioners = INTEG(
   adoptions - graduations,
         0)
Units: Person

training productivity = 20
Units: 1/Year
```

If we simulate this model at the three extremes, with *application fraction = 1* (all effort is devoted to work in the field, and new practitioners must train themselves), *training fraction = 1* (all effort is devoted to training novices) and *supervision fraction = 1* (all effort is devoted to generating experienced practitioners) we get the following behavior:



Devoting all attention to supervision or application both result in a much slower growth and saturation, with the only difference being in the fraction of the people who are experienced. If experienced people spend all their time training new practitioners then a big fraction of practitioners are going to be experienced, but since experienced people do nothing but make more experienced people no useful work comes of it.

If experienced people spend all their time training novices, there is a profound effect on the growth of the field. People who express interest can quickly become proficient and start using the technology. While this is an interesting result, it also suggests a deficiency in the model. If experienced people are only doing training, then all the work being done is being done by *New Practitioners* who are not likely to perform as well as experienced practitioners.

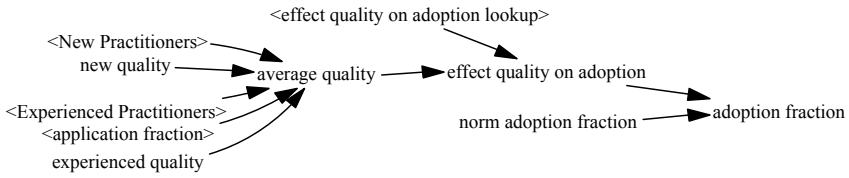
Quality of Work (sdgrow3.mdl)

The willingness of people to adopt a new technology depends on a number of things including the difficulty of learning the technology, the expected benefits and the compatibility of the technology

with existing technologies. While it is important to have lots of people espousing the value of a technology, unless the technology displays significant and valuable results, it will never take off.

We will use *quality of work* as a measure of the success of the technology, and differentiate between new and experienced practitioners in determining the quality of work being done. Quality here represents the fraction of projects that are successfully implemented. Projects that lead to bad decisions, are started but abandoned, never get implemented or otherwise get off track are not successes. We will let the quality of work being done influence *adoption fraction*.

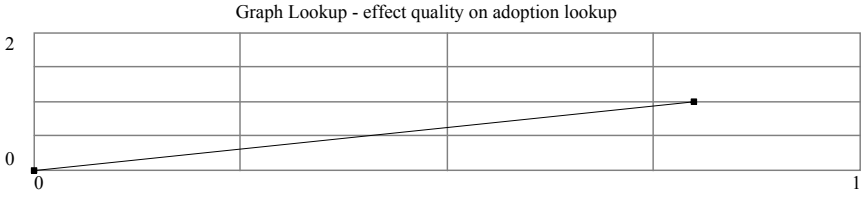
We add new variables to get *average quality* and its effect on adoption:



Now we add the equations:

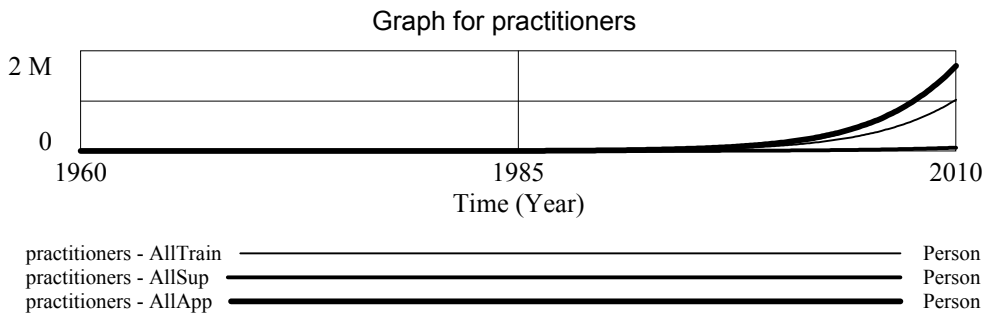
```
adoption fraction = norm adoption fraction *
         effect quality on adoption
Units: Dmnl

average quality =XIDZ( New Practitioners * new quality +
   Experienced Practitioners * application fraction *
   experienced quality, New Practitioners +
   Experienced Practitioners * application fraction,
   experienced quality)
Units: Dmnl
```

The XIDZ function prevents a numerical error (overflow) by setting the **average quality** to **experienced quality** in the special case when the **application fraction** is 0.



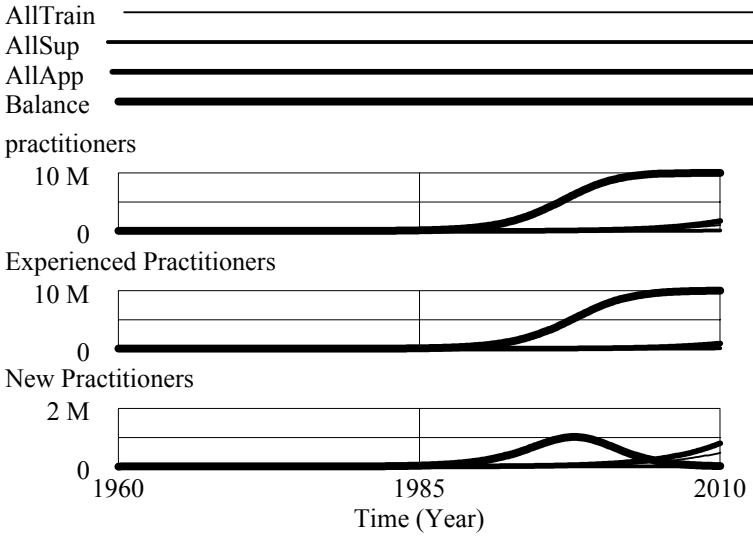

```
effect quality on adoption lookup((0,0),(0.8,1))
effect quality on adoption = effect quality on adoption lookup(
         average quality)
Units: Dmnl

experienced quality = 0.8
Units: Dmnl

new quality = 0.4
Units: Dmnl

norm adoption fraction = 0.005
Units: Dmnl
```

When we simulate this model under the three extremes we get:



Having all practitioners spend all their time on applications (everyone learns by doing) is now the best growth strategy, but all of the growth rates are slow relative to those of the last model. The reason is simple; when only new practitioners are doing applications the quality is low and new interest is lowered. To maximize growth in this model it is necessary to get a balance between applications and teaching.

If we set both *supervision fraction* and *training fraction* to 0.1 we get better results:

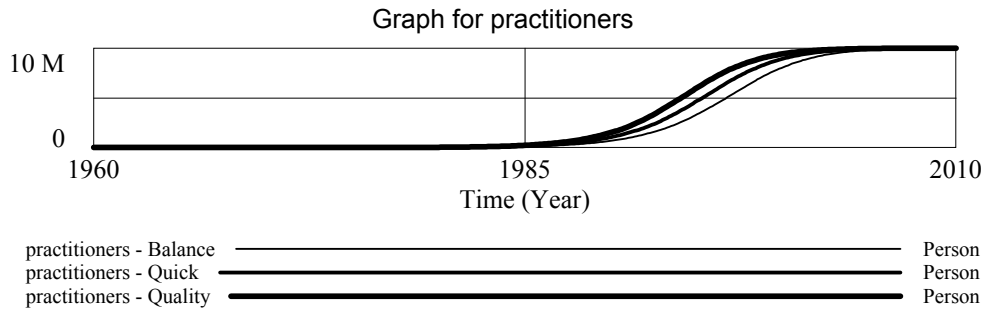


The point here is that as we add additional structure to the model to enhance its realism, the simple-minded strategy of training people like mad falls apart.

Software Tools

The model we have developed does provide a framework for thinking about the role of new software tools. Software provides two things — ease of use and depth of understanding. The first of these decreases the training time required to become proficient. The second increases quality for both new and experienced modelers.

Keeping the 10% of experienced modelers time for both training and supervision try cutting *self training time* and *min training time* in half as an experiment (call this run Quick). Then as an alternative experiment change *new quality* to 0.6 (call this run Quality).



Both ease of use and quality can have a significant impact on the speed of diffusion. Vensim was designed to provide both ease of use and higher quality results and we hope it will speed the growth of System Dynamics.

Conclusions

We have started from a number of written hypotheses and developed a model that has helped us to explore some of these hypotheses in a unified framework. This technique has a major advantage of focusing attention on policy issues from the start and also forces a continuity between the model and the way different people think about the problem. This is very helpful. One of the biggest problems in implementing model results is explaining the results in terms that the people implementing them can relate to. By continually relating a model to hypotheses, this issue is addressed from the beginning and can be less daunting in the end.

At this point a warning is in order. We have built a simple conceptual model to help us think about different issues around growth in the field of System Dynamics. While the model we developed has provided insights and allowed us to explore some of the hypotheses put forward, it is not possible to draw conclusions from it.

When you get an insight from a simple model you need to stop and look around and ask yourself "is this what is happening." In some cases the answer is yes, and the model has given you a new basis for understanding reality and acting on that understanding. In this case the answer is maybe. We have seen some plausible dynamics, but done little to establish confidence that the model represents what is really happening. Unless we go further and make use of data and Reality Checks, we could end up with a model that seems plausible, but is just plain wrong.

5 Capacity and Market Growth

In Chapter 4 we described how, through simply changing variable names, the diffusion model we were working with could be used to describe the introduction of a new product. In this chapter we want to take that idea, and add production dynamics to relate the market to the firm. This will allow us to investigate the use of different strategies to respond to market demand. We will extend this model further to deal with more than one competitor in Chapter 6.

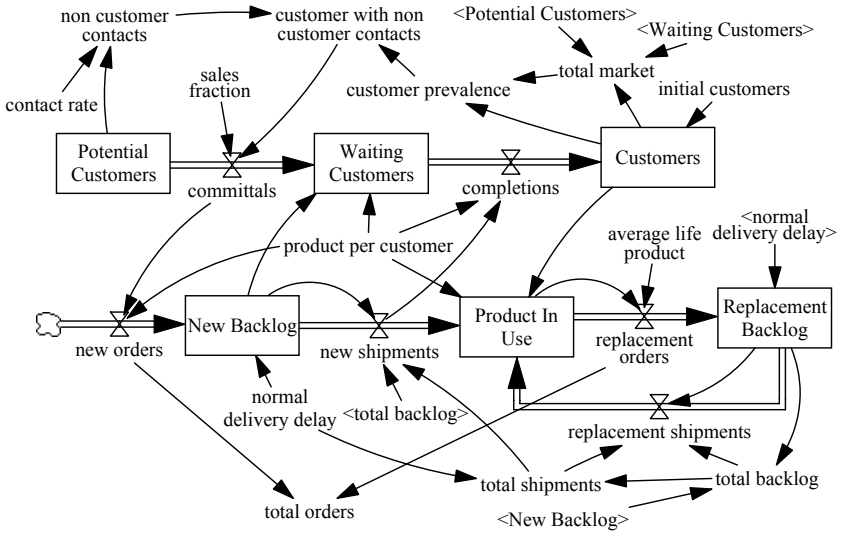
Sales and Replacements (prod1.mdl)

In order to make use of our diffusion model from Chapter 4, we need to refine it to deal with the reality of purchasing a new product. First, in that model, *sales* represents the number of people who are sold on the product. In order to stay in business, however, we need to distinguish between the people who have adopted the product and the number of products actually being bought. Though the two concepts are similar, it is useful to separate them and provide a dimensionally consistent view of the process.

Distinguishing people from products allows us to represent several other important characteristics of the real system, like order backlogs and replacement purchases. Once someone decides to purchase something it cannot always be immediately obtained. Delivery delays are common when new markets are booming. Since people who are waiting do not have the product, they are unlikely to be strong advocates. Finally, and fundamental to survival, people who have purchased a product use it and replace it when it wears out or becomes obsolete. This process allows industries to prosper for extended periods of time, and should not be overlooked.

To address these issues we will set up two parallel stock and flow structures— one for customers and one for products. When a potential customer orders a product for the first time this it called a *New Order*. The order goes into *New Backlog*, and is filled over *normal delivery delay* to become a *Product In Use*. While the customer is waiting for shipment of the product, they are in the *Waiting Customers* pool. It is only after a *new shipment* occurs that customers become an active advocate of the product.

We will run the model for 5 years with *TIME STEP* at .0625.



Though we have added quite a bit of structure to this model, it still has the same dynamic character as the diffusion model. The central feedback loops driving growth and saturation (more customers, more contacts, more sales eventually making everyone a customer) are unchanged. There is a potential buffer (*Waiting Customers*) of people who are not active (neither advocates nor new purchasers) dampening the growth effects. In this first round this will not have a significant impact since we will keep delivery delay short, but when we introduce production it becomes quite important.

The equations for the model are:

```
average life product = 2
Units: Year

committals = customer with non customer contacts * sales fraction
Units: Person/Year

completions = new shipments/product per customer
Units: Person/Year

contact rate = 500
Units: 1/Year
```

The *contact rate* has been increased relative to the original diffusion model in Chapter 4 because it is much easier to show someone a product and say it is wonderful than it is to explain a complex methodology like System Dynamics.

```
customer prevalence = Customers/total market
Units: Dmnl

customer with non customer contacts = non customer contacts *
   customer prevalence
Units: Person/Year

Customers = INTEG(
   completions,
         initial customers)
Units: Person

initial customers = 100000
Units: Person
```

We start with a significant number of customers. If we were starting with a small number of customers it would be necessary to add in advertising or some other activity to get the diffusion process going in a reasonably short period of time (this is a good experiment to try on your own).

```
New Backlog = INTEG(
   new orders - new shipments,
         new orders * normal delivery delay)
Units: Gadget

new orders = committals * product per customer
Units: Gadget/Year

new shipments = total shipments * New Backlog/total backlog
Units: Gadget/Year
```

new orders and *replacement orders* are both competing for the same supply of production (limited by available capacity). This formulation says that those products are rationed proportional to the amount previously demanded. This is a simple, and commonly useful way to represent the distribution of a scarce resource. The formulation *new shipments = New Backlog/normal delivery delay* would give the same results. This formulation was chosen because it will make adding a production sector simpler.

```
non customer contacts = Potential Customers * contact rate
Units: Person/Year
```

```
normal delivery delay = 0.125
Units: Year

Potential Customers = INTEG(
   - committals,
         1e+007)
Units: Person

Product In Use = INTEG(
   new shipments + replacement shipments - replacement orders,
         Customers * product per customer)
Units: Gadget

product per customer = 1
Units: Gadget/Person

Replacement Backlog = INTEG(
   replacement orders - replacement shipments,
         replacement orders * normal delivery delay)
Units: Gadget

replacement orders = Product In Use/average life product
Units: Gadget/Year

replacement shipments = total shipments * Replacement Backlog /
   total backlog
Units: Gadget/Year

sales fraction = 0.005
Units: Dmnl

total backlog = New Backlog + Replacement Backlog
Units: Gadget

total market = ACTIVE INITIAL(
   Potential Customers + Waiting Customers + Customers, Potential
   Customers)
Units: Person
```

The ACTIVE INITIAL function is required here to break a simultaneous initial condition loop involving *Waiting Customers*. What this function does is use the first expression during simulation, but the second expression during initialization to set the values for the different levels in the model. Since both *Waiting Customers* and *Customers* are small at the beginning of the simulation this is a reasonable approximation that breaks the initial value interdependencies.

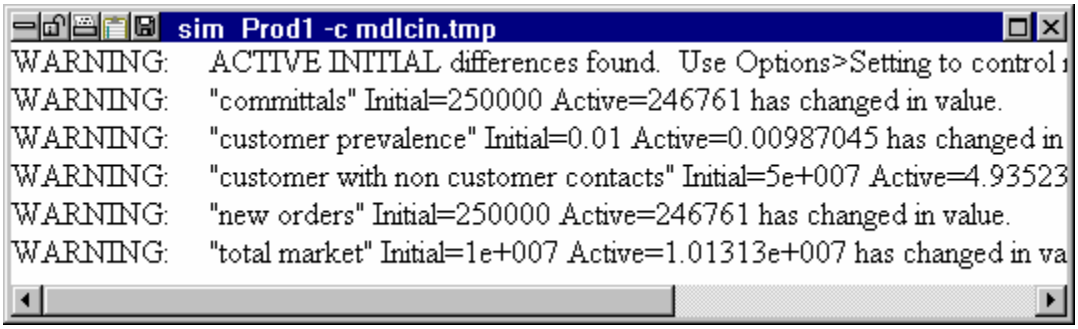
```
total shipments = total backlog/normal delivery delay
Units: Gadget/Year
```

This model was crafted somewhat carefully so that, when we add in the production structure, *total shipments* will be the only equation that requires modification. Though such complete separation of structure is desirable for pedagogical purposes, it is not a goal that should be emphasized. Feedback is pervasive, and excessive isolation of different sectors can lead to unrealistic models.

```
Waiting Customers = INTEG(
   committals - completions,
         New Backlog/product per customer)
```

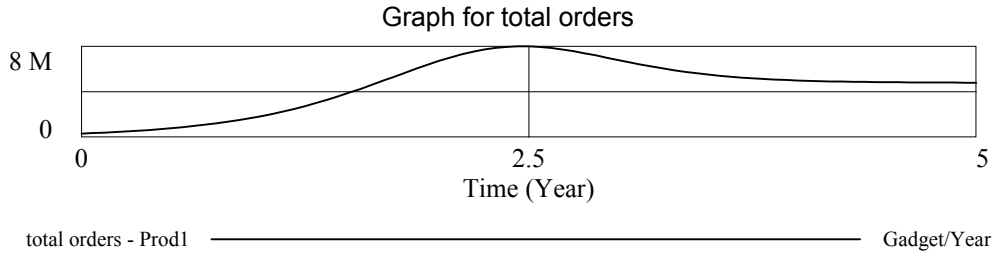
Waiting Customers is initialized to be in balance with the parallel flow of orders and shipments.

When you simulate this model you may, depending on your options settings, receive a series of warnings:



These warnings relate to the use of the ACTIVE INITIAL. The value of *total market* used to initialize the Levels in the model was 1E7. Once all the Levels were initialized, *total market* was computed with the active portion of ACTIVE INITIAL to have a value of 1.013E7. Additional variables also went through these two computations even though the ACTIVE INITIAL function was not explicitly used for them. The different values are reported. All are small changes and no real cause for concern.

This basic model generates the same behavior as the diffusion model discussed in the previous chapter. Because of the addition of *replacement orders*, however, *total orders* has a new profile.



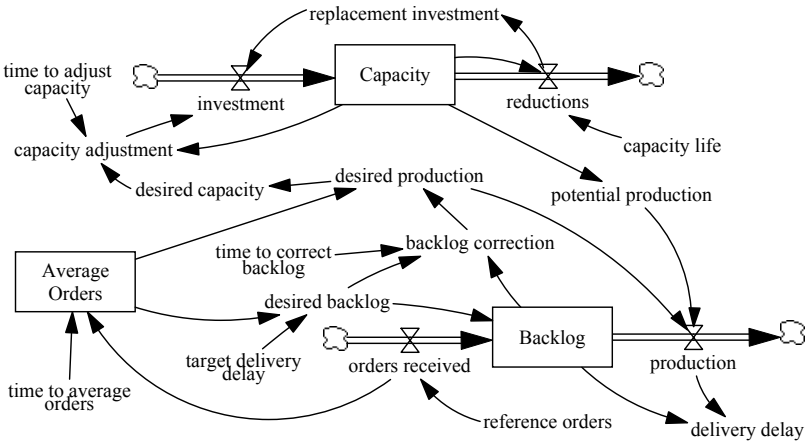
Here orders grow rapidly, but instead of falling back to 0, they go to a sustained value of just under 5 million Gadget/Year. This is the replacement demand of all the customers using the product. Because of the way the model is formulated, total products in use does not reach the value of 5 million that would occur if everyone had the product. This is because once a decision is made to replace a product, it takes time to get the replacement, so some portion of the population is always waiting to receive the product.

Production (prod2.mdl)

In order to model the production side we need to determine capacity (how much can be produced) and track the fulfillment of orders with shipments. The determination of capacity is formulated as a goal adjustment based on a target production. This is very similar to the formulation for hiring used in the Workforce-Inventory model. In this model, however, capacity is thought of as including both capital and labor (a composite ability to produce), so the time constant for adjusting capacity is longer, reflecting the long lead times in acquiring facilities and capital equipment.

The tracking of orders is done through a backlog (orders placed but not yet fulfilled). A backlog is, in many respects, the opposite of an inventory; in this respect the production side does resemble the Workforce-Inventory model. The other difference in this model is that the stream of orders is averaged when calculating target capacity and backlog. In the Workforce-Inventory model we formulated target production directly on the basis of sales. While this was the simplest formulation possible, it is not realistic. People do not, based on a good day or a good month, immediately reformulate all plans for the future. Short term variation in the order stream is smoothed out and therefore has very little impact on future plans.

We create a new view and add in the production sector:



The production sector of this model has been formulated as a standalone view, completely disconnected from the first view. The reason this was done was so that we can test the behavior of this portion of the model in isolation. We will then link up the two parts of the model to see how they operate together. This is a useful method for keeping things under control. In general, the boundaries between subsystems of a model will not be quite as clean as they are in this example, but it is always possible to replace model variables with test inputs to see how fragments of structure behave in isolation.

The equations for this sector, and these are a complete model unto themselves, are:

```
Average Orders = INTEG(
   (orders received - Average Orders)/time to average orders,
         orders received)
Units: Gadget/Year
```

The equation for *Average Orders* could have been written using a SMOOTH function. The explicit integration is used because it emphasizes the nature of the smoothing process.

```
Backlog = INTEG(
   orders received - production,
         desired backlog)
Units: Gadget
```

```
backlog correction = (Backlog - desired backlog)/
         time to correct backlog
Units: Gadget/Year
```

Backlog is initialized at *desired backlog* in order to start this model in an equilibrium.

```
Capacity = INTEG(
   investment - reductions,
         desired capacity)
Units: Gadget/Year
```

Capacity is initialized at *desired capacity* in order to start the model in equilibrium.

```
capacity adjustment = (desired capacity - Capacity)/
         time to adjust capacity
Units: Gadget/Year/Year
```

```
capacity life = 2
Units: Year
```

```
delivery delay = Backlog/production
Units: Year
```

```
desired backlog = Average Orders * target delivery delay
Units: Gadget

desired capacity = desired production
Units: Gadget/Year

desired production = Average Orders + backlog correction
Units: Gadget/Year

investment = capacity adjustment + replacement investment
Units: Gadget/Year/Year

orders received = reference orders * (1 + STEP(1,2))
Units: Gadget/Year
```

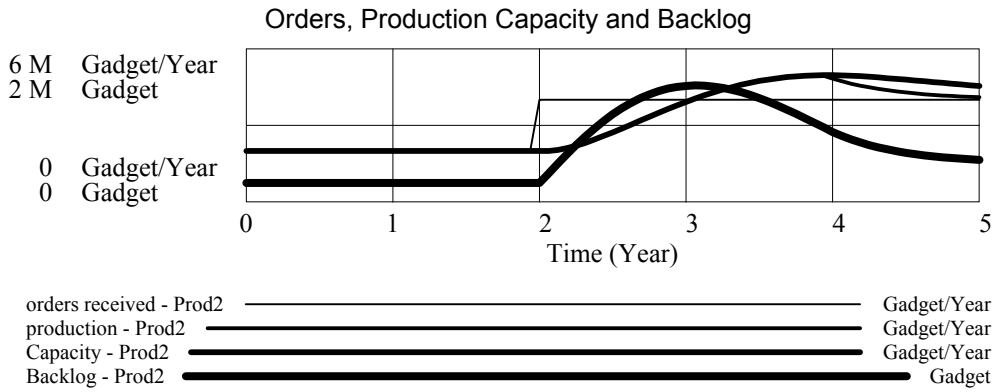
The STEP function in *orders received* causes a doubling of orders at Time 1. Again, a test input using a STEP function is useful because it generates behavior that is easily understood relative to the driving inputs.

```
potential production = Capacity
Units: Gadget/Year

production = MIN(desired production,potential production)
Units: Gadget/Year

reductions = Capacity/capacity life
Units: Gadget/(Year*Year)

reference orders = 2e+006
Units: Gadget/Year

replacement investment = reductions
Units: Gadget/Year/Year

target delivery delay = 0.125
Units: Year
```

The *target delivery delay* is set to the same value as *normal delivery delay* was set in the consumption model. Although it is not necessary, two parallel concepts should generally be close, if not the same in value.

```
time to average orders = 0.25
Units: Year

time to adjust capacity = 1
Units: Year

time to correct backlog = 0.5
Units: Year
```

When we simulate this model the behavior is similar to that of the Workforce-Inventory model in Chapter 1. Following the step increase in orders, the order *backlog* grows for a year. *Production* gradually increases until it exceeds orders, and the *backlog* begins to fall. *Capacity* overshoots, and by year 4 there is excess capacity as the desired production rate falls.



As an experiment you might want to set *target delivery delay* to 1 year and see what happens. This is an unrealistically high value for most products, but does result in interesting dynamics.

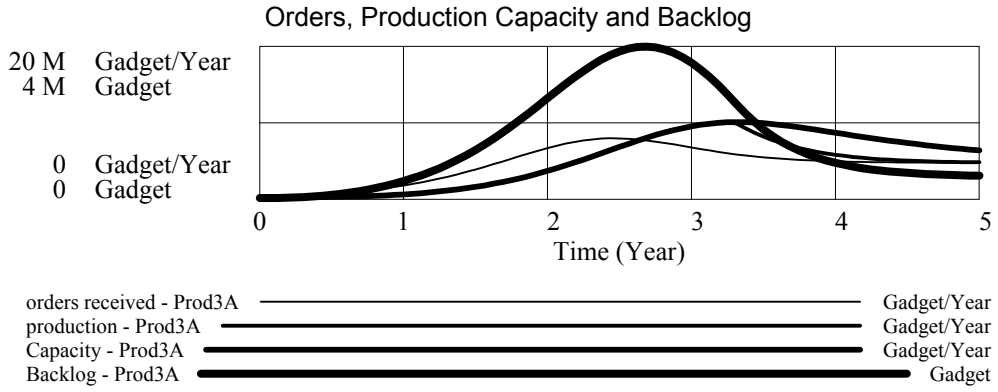
Combining Sectors (prod3.mdl)

What we now want to do is to connect these two models. We will do this one step at a time. First we will use the orders generated by the growth in customers to drive orders. Then we will use the *production* to drive *total shipments*.

We first delete *reference orders*. Next, we add *total orders* and its causes (*new orders* and *replacement orders*) from View 1 using the Model Variable tool. Switching to View 1, we cut *total orders* from the view (highlight with Pointer and Edit>Cut, or Ctrl-X). Back to View 2, we connect *total orders* to *orders received* and change the equation for *orders received* to:

```
orders received = total orders
```

Now simulate the model. Instead of using a test input for orders (*reference orders*), we are now using the output from the first model as the input. (If you are working with the models that come with Vensim please note that the model *prod3.mdl* will not generate the results that follow. To get these results you need to set total shipments = backlog/normal delivery delay in *prod3.mdl* or work from *prod2.mdl* as described above.) We get the results:

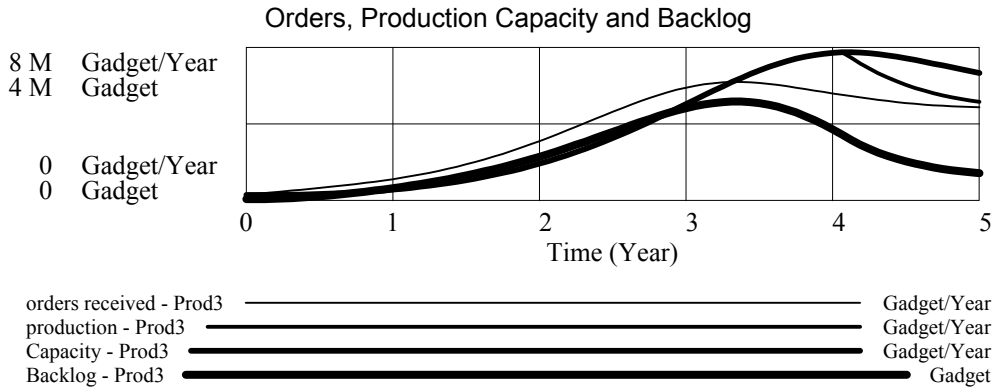


Backlog grows for two and a half years then begins to fall off as new orders slow. Production peaks in the third year (lagging a peak in orders in the second year) and there is excess capacity thereafter.

On the consumption side we still have the assumption that *total shipments* are equal to *total backlog* divided by *normal delivery delay*. This means that the diffusion process is unaffected by production capacity limitations. To finish coupling the two submodels, we change this to:

```
total shipments = production
```

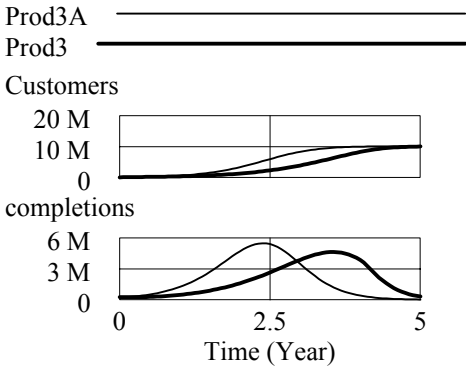
and modify the diagram appropriately. When we simulate this new structure (be sure to use a new name for your run) we get different behavior:



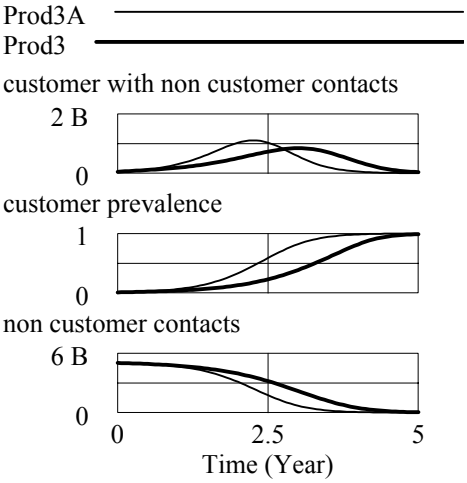
The peaks in production and orders occur much later and are less extreme. The peak backlog is less than 3 million whereas it was almost 5 million in the previous run. This occurs because the diffusion process is now limited by the rate of capacity expansion.

Comparing Runs

At this point it is useful to explore the differences between the last two runs that were made. With the two runs loaded, select *Customers* into the Workbench and click on the Causes Strip graph:



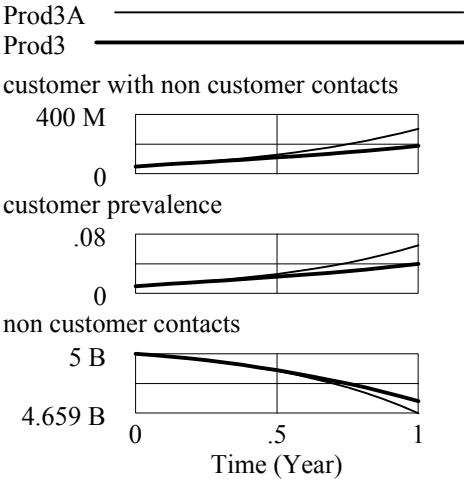
Now we double click on *completions* and again click on the Causes Strip graph. A useful path to follow is: *Customers*, *completions*, *new shipments*, *new orders*, *committals*, *customer with non customer contacts*. This gives you the final graph:



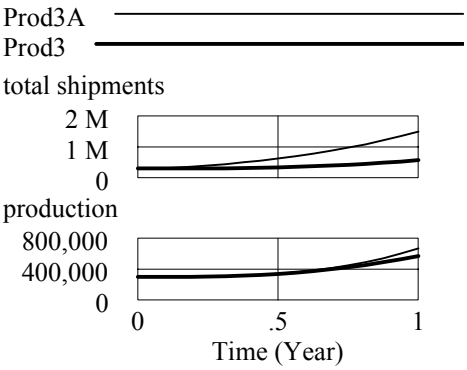
Customer prevalence stays much lower for much longer. Because prevalence is low, there is much more limited contact, and therefore fewer people are hearing about and deciding to purchase the product.

We want to focus in on the first year to understand what triggers the lower customer prevalence. Restrict the time axis to display behavior from about year 0 to year 1 (see Chapter 7 of the Tutorial if you don't know how to do this).

Click on the Causes Strip graph again. Output for the narrower time range is displayed:



With this narrower time range still selected, trace the causes of *customer prevalence*. A useful path is *customer prevalence*, *Customers*, *completions*, *new shipments*, *total shipments*. When we get to the final Causes Strip graph we have:



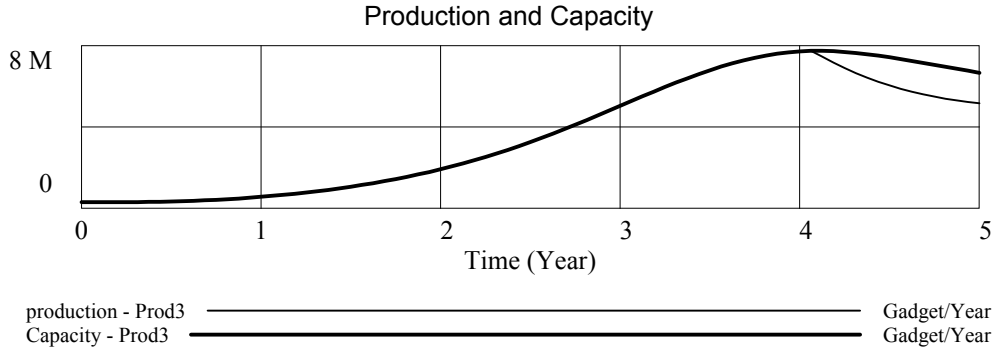
This is the new connection that did not exist in the previous model — total shipments now depends on production, and production takes time to get up to speed due to the delays in perceiving the growth in orders and acquiring capacity.

The decreased production means that fewer people have the product, and this in turn means that demand takes longer to build. In a sense this is good, since it decreases the excess capacity overshoot as the market saturates. From a competitive standpoint it is a problem, and this is explored in Chapter 6.

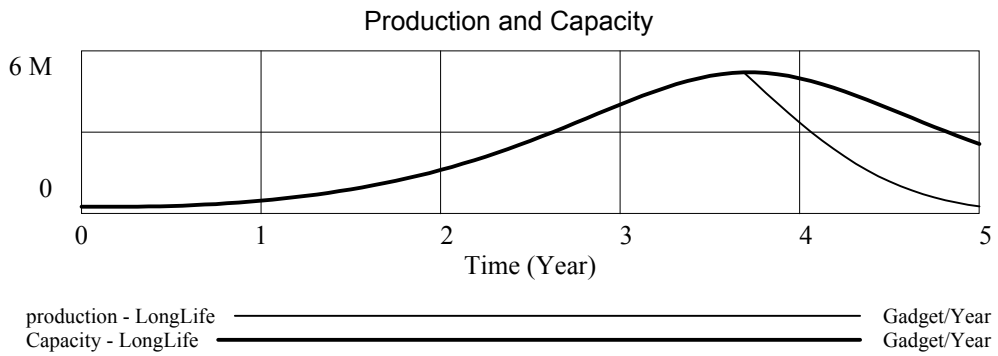
Experimentation

There are a variety of experiments that can be performed with this model. Try changing some of the parameters in the model and look at the resulting changes in behavior. For example, you might decrease *time to correct backlog*, to look at the consequences of more aggressively managing your backlog. Changing *time to adjust capacity* will allow you to look at the consequences of being able to get capacity more quickly or less quickly.

A very interesting experiment is to change *average product life* to a very large number. Consider a plot of *production* versus *capacity* for the base run:



Now look at the same plot with a product that essentially never wears out:



There is a much bigger excess capacity. You will find that as you lengthen the life of the product, the amount of excess capacity increases. Conversely if you shorten the life of the product you can get the model to a point where there is no excess capacity. The difference between hamburgers and chain saws. This is a very simple idea, but one that eludes many people trying to manage a product in an emerging market.

6 Competitive Dynamics†

The models in this chapter use Subscripts and can only be developed in Vensim Professional and DSS.

In Chapter 5 we introduced a small market growth model showing some advantages to responding less quickly to market demand, with a smoother, but more prolonged, transition to sustained replacement. This result quickly loses its meaning when the prospect of competition is introduced. In this case, being slow to respond may mean losing the business to competitors.

This chapter describes how to create equations that allow us to look at the consequences of having more than one producer in the market growth model. To achieve this, we use the Subscript capabilities of Vensim Professional and DSS. While this chapter may prove interesting reading for users of Vensim Standard, you will not be able to construct or simulate the model as described. Constructing a two sector model (for production), if you wish to pursue it, can be done by replicating the production sector and linking the sectors to the customer view by adding in the appropriately modified coordinating equations (described in this chapter for the subscripted model).

Adding Subscripts to the Model (prod4.mdl)

You can either start with the model from Chapter 5 (*prod3.mdl*) and add subscripts as described here, or open the finished model *prod4.mdl* and skip to later in this section. See Chapter 17 of the Vensim User's Guide for instructions on the mechanics of adding subscripts to a model. Only a brief outline of what to do is presented below.

We start by adding a subscript range *producer* to the model. Open the Subscript Control (using the rightmost Icon on the Toolbar) and click on the **New** button. You will be queried for a name. Enter *producer* and click on **OK**. The Equation Editor will open. Type in *US, THEM* and click on **OK.**

```
producer : US, THEM
```

The Equation Editor will close. (You need to close the equation editor after creating the producer subscript range so that it can be reset to handle subscripts the next time it is open. If, instead of closing it, you click on Choose or Next the subscript functionality will not be available.)

Now go to the second View in the model and select everything except *total orders*, *replacement orders* and *new orders*. You can do this by using Edit>All But Shadow followed by a Shift-Click on *total orders*.

Now use the menu command Edit>Set Subscripts. You should get a dialog box labeled **Modify Subscripts for - 20 variables**. Select *producer* in the **Subscript 1** dropdown and Click on OK. The *producer* Subscript is added to the equations for these 20 variables as well as those in which the variables are used. Because of the way this model has been set up, this almost works. There are only two problems. One is *production* and the other is *orders received.*

Because *orders received* is the output of the consumption sector, and because there are no Subscripts for the consumption sector, this variable cannot simply be Subscripted. This equation for it needs to specify how the orders are divided between the two producers. This simplest assumption is that both producers receive half of the orders:

```
orders received[producer] = total orders/2
```

Note that the old equation for *orders received* would not cause any error messages to occur and it would pass units checking. The only problem is that it allocates more sales than there are. You do need to exercise care when adding subscript to a model so that you do not make such an error.

On the other side, *total shipments* used to be equal to *production*. However, the equation

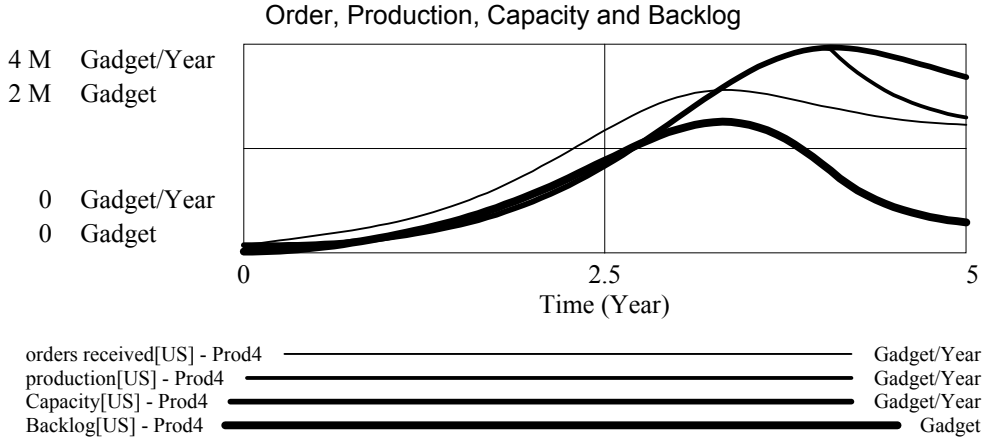
```
total shipments = production[producer]
```

is not valid and Vensim will report an error. Since more than one firm is now producing we need to make

```
total shipments = SUM(production[producer!])
```

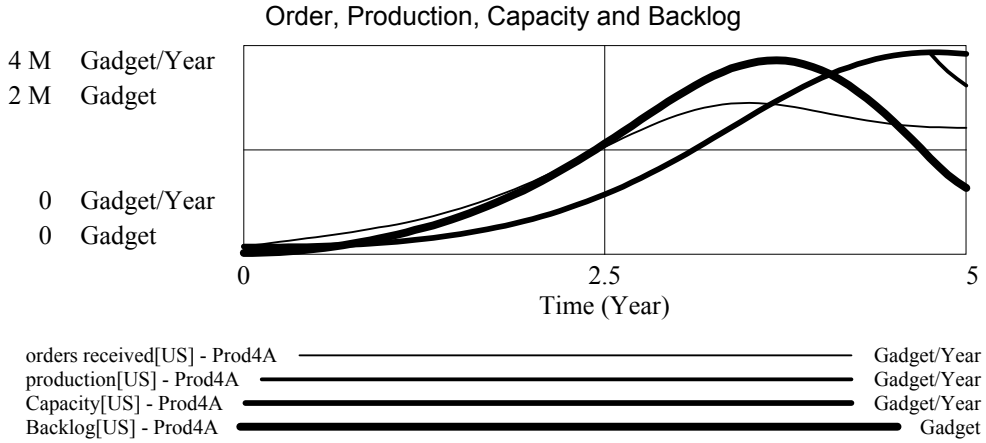
This equation indicates that *total shipments* is the sum of *production* for each *producer*. The exclamation mark ! is used to indicate which Subscript in the equation to sum over.

Simulating this model gives results, on the production side, of:



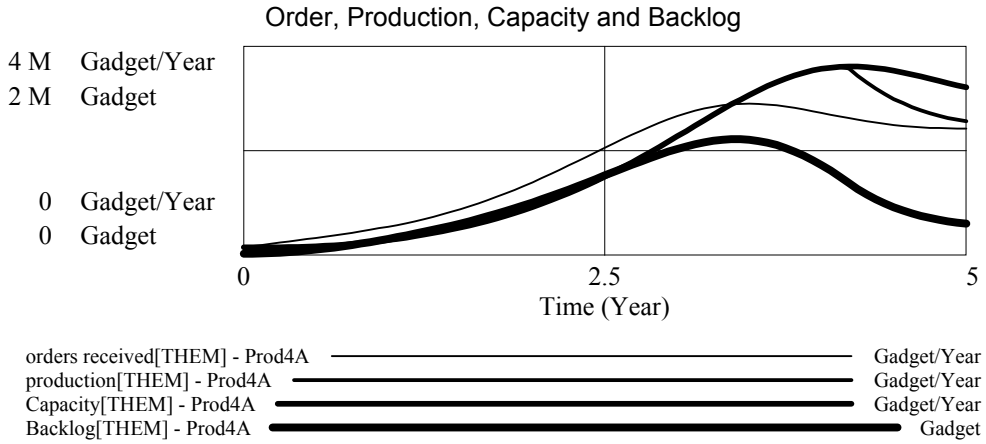
This is the same as the results we previously saw, but on a different scale. What happens, not surprisingly, is that we have two production sectors that are 50% reproductions of the old production sector. If the behavior of your two production sectors is not identical, you should review your model for errors.

Now suppose that we let one of these production sectors (*US*) be more conservative in adjusting capacity. To do this we set *time to adjust capacity[US]* to 2. When we did this with only one production sector, we found a smoother transition with lower capacity overshoot. Now we get the following results:



The market growth process is delayed, but there is almost no difference in the final capacity level reached. You can look at the behavior of *THEM* using this same graph by clicking the **Subscripts**

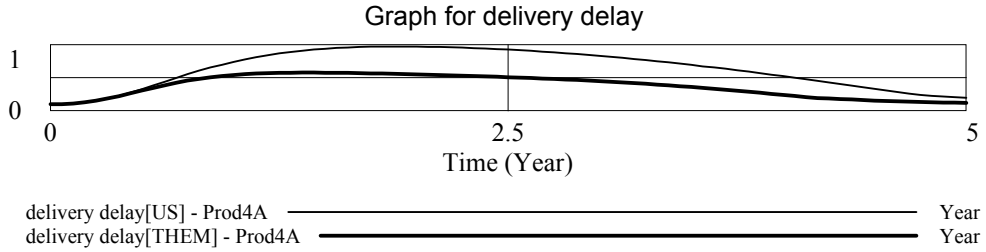
control button, then clicking on the **producer** tab, then clicking on *US* to toggle it off, and finally *THEM* to toggle it on. Now when you activate the custom graph again and you will get:



In this case the timing is about the same as it was in the first run, but there is less excess capacity in the end. This is an interesting example of how a policy effect in an aggregate model can be split among different sectors. This result should not be given to much emphasis, however, because there is something apparently wrong with the behavior of this model.

Demand and Delivery Delay (prod5.mdl)

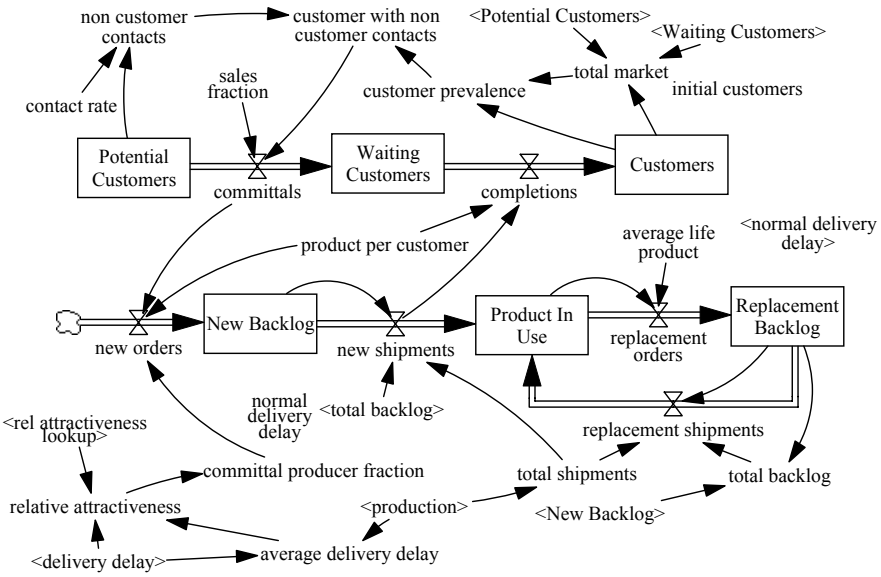
The above results show two producers, one with a huge backlog and one with a small backlog, both receiving the same number of orders. To plot the two delivery delays click on the **producer** tab in the **Subscript Control** and highlight both *US* and *THEM* (or click the **All** button). Now select *delivery delay* into the Workbench, and click on the Graph tool .



Here we have two widely divergent values for *delivery delay*, yet people continue to place the same number of orders with both producers. What we need is a connection from product availability to shipments by the producer to the amount ordered from each producer.

Assuming that the producers are not working together, the only mechanism for shifting demand must come from consumer behavior. Of the producer's activity, the only one directly observable by the customer is *delivery delay*. We can change the model so that customers' new purchases are allocated more favorably to the supplier with the shortest lead time.

In order to do this we need to partially introduce Subscripting into the demand side of the model. We have already created two parallel flows of people and products in this part of the model, and the Subscripting will be applied only to the order and shipment portion of this sector.



Note that initial causes have been hidden to make this diagram more readable. The new structure, shown in the lower left hand corner, determines the *relative attractiveness* of the two companies from *delivery delay*. On the basis of attractiveness, a fraction of total orders, given by the *committal producer fraction*, is allocated to each producer. The remaining structure is unchanged, except that Subscripts have been added to a number of the equations. The affected equations in the consumption sector are:

```
average delivery delay = SUM(delivery delay[producer!] *
   production[producer!]) / SUM(production[producer!])
Units: Year
```

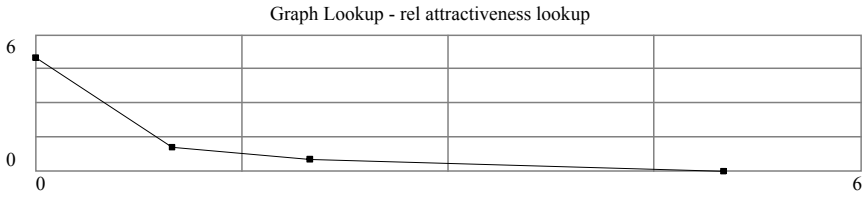
Average delivery delay is computed as the average over the different producers weighted by *production*.

```
committal producer fraction[producer] =
         relative attractiveness[producer]/
               SUM(relative attractiveness[producer!])
Units: Dmnl

completions = SUM(new shipments[producer!])/product per customer
Units: Person/Year

New Backlog[producer] = INTEG(
   new orders[producer] - new shipments[producer],
         new orders[producer] * norm delivery delay)
Units: Gadget

new orders[producer] = committals * product per customer *
         committal producer fraction[producer]
Units: Gadget/Year

new shipments[producer] = total shipments[producer] *
         New Backlog[producer]/total backlog[producer]
Units: Gadget/Year
```

The computation of shipments for both new and replacement product is parallel to the original formulation, but specific to a producer. Once a customer is in with a producer, the customer sticks with that producer.

```
Product In Use[producer] = INTEG(
```

```
   new shipments[producer] + replacement shipments[producer] -
   replacement orders[producer],
         Customers * committal producer fraction[producer] *
         product per customer)
Units: Gadget
```




```
rel attractiveness lookup ((0,5),(1,1),(2,0.5),(5,0) )
Units: Dmnl

relative attractiveness[producer] = ACTIVE INITIAL(
   rel attractiveness lookup(delivery delay[producer]/
         average delivery delay),1)
Units: Dmnl
```

The function ACTIVE INITIAL is used here to prevent a simultaneous initialization problem. The initialization of the production sector requires that orders be known. But without the above ACTIVE INITIAL function the computation of orders requires that *Backlog* be initialized. In order to break an initial value simultaneous equation it is usually most useful to set some variable to its neutral value. In this case the natural variable to set is *relative attractiveness* (since it has a neutral value of 1).

```
Replacement Backlog[producer] = INTEG(
   replacement orders[producer] - replacement shipments[producer],
         replacement orders[producer] * norm delivery delay)
Units: Gadget

replacement orders[producer] = Product In Use[producer]/
         average life product
Units: Gadget/Year

replacement shipments[producer] = total shipments[producer] *
   Replacement Backlog[producer]/total backlog[producer]

Units: Gadget/Year

total backlog[producer] = New Backlog[producer] + Replacement
   Backlog[producer]
Units: Gadget

total orders[producer] = new orders[producer] + replacement
   orders[producer]
Units: Gadget/Year

total shipments[producer] = production[producer]
Units: Gadget/Year

Waiting Customers = INTEG(
   committals - completions,
         SUM(New Backlog[producer!])/product per customer)
```

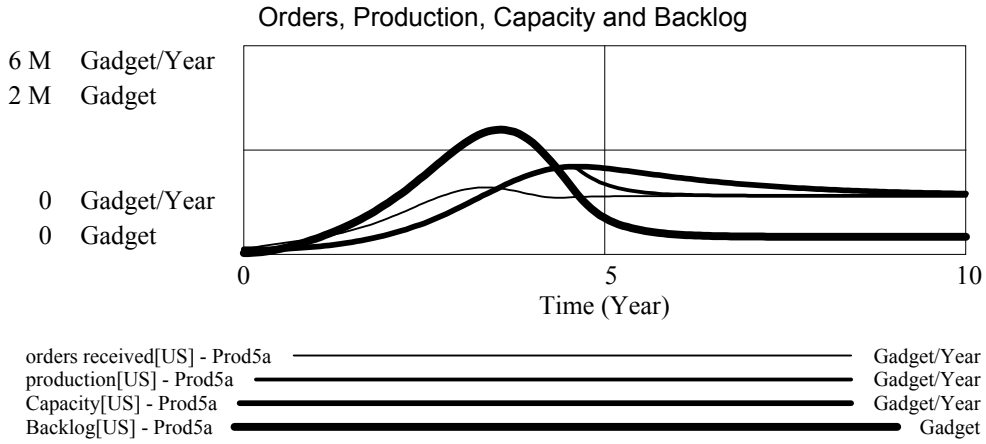
These modified equations include both equations to which Subscripts have been added to all variables, and equations containing some Subscripted variables. There are a number of SUM functions used to go from looking at one production sector to all production sectors.

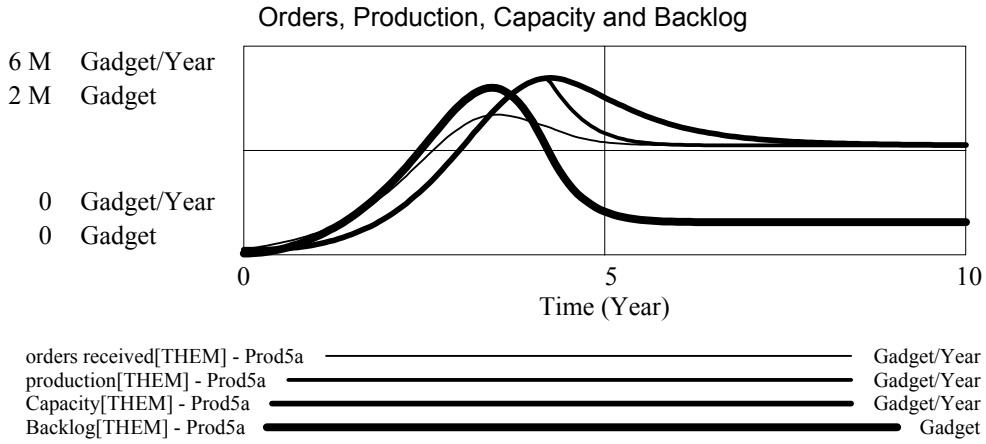
In the production sector we need to redo the equation for *orders received* as

```
orders received[producer] = total orders[producer]
```

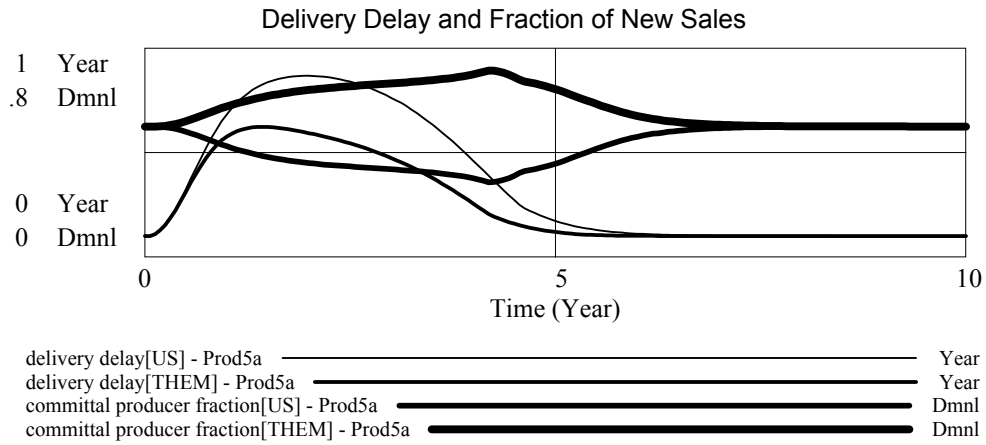
All of the logic for allocating demand is now done in the consumption sector. One important note is that no equation depends on the number of production sectors. This model can be modified to include additional production sectors by simply changing the equation for *producer* subscript range definition.

When this model is simulated using the base parameters the behavior is precisely the same as the previous model, since the relative attractiveness of each firm is identical. Changing *time to adjust capacity[US]* to 2, however, produces dramatically different results.





Here the simulation was run for 10 years to show the long term results. Slowing down the adjustment of capacity does very little to smooth out the adjustment process. Instead, demand shifts to the other supplier, who ends up with nearly twice the final market. In this case if we look at *delivery delay* and *committal producer fraction* for the two suppliers we see:



There is a persistent difference in *delivery delay*, and it is interesting how long we lose market share. Even as our delivery delay falls, so does theirs. Furthermore, because they are growing more quickly than us, *average delivery delay* falls very quickly as well.

Conclusion

We have taken a model showing the interactions between the demand for a product and the ability to produce that product and extended it to deal with more than one producer. In addition to providing good technical background on the use of Subscripts, the model has provided some useful insights into the nature of new product markets. Policies that, in a monopoly situation, smooth out the transition can, in the face of a competitive supplier, simply benefit the competitor. It is true, in general, that as more detail is put in about the parties involved in a process it is possible to identify some as benefiting more strongly than others as new policies are implemented.

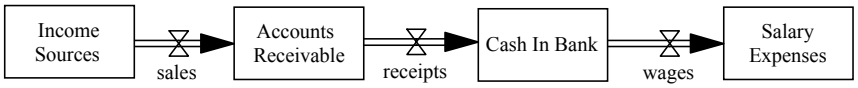
7 Financial Modeling and Risk

In developing models for business, it is often helpful to be able to report standard financial measures. This can help managers understand the business implications of different policies, and also tie the modeling effort to something that is more familiar and better understood. In this chapter we will develop a relatively simple financial model for analyzing a single investment, and then link this model to the one developed in Chapter 5.

Undertaking a new investment is an activity that entails risk. So far, the models we have constructed have not explicitly incorporated any risk assessment but rather focuses on the nature of the internally generated dynamics. In many cases, however, understanding the ranges of behavior that can be generated by a model can be very helpful. In this chapter we will use the multivariate sensitivity analysis tools available in Vensim to assess financial risk.

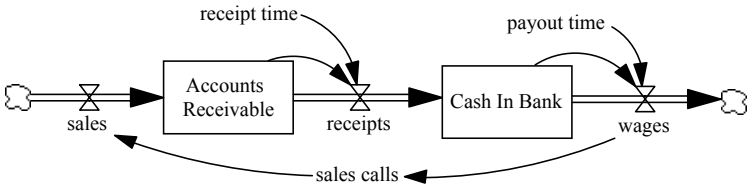
Accounting and Causality

Stock and flow representations and standard accounting practice have some fundamental consistencies. Every transaction is a flow, and all flows change stocks. The focus in accounting is on tracking each and every transaction that occurs. We might represent a simple company's accounting system as:



Note that there are no arrows indicating causality coming into any of the rates. The purpose of accounting systems is not to model behavior, but to manage transactions. The causes for those transactions are outside the realm of the accounting system. Also note the lack of clouds or sources and sinks in the above representation. A basic tenet of double entry bookkeeping is that every Debit has an associated Credit and all transactions are required to be balanced. *Income Sources* and *Sales Expenses* provide a history of what has happened but they are not part of the company.

A more functional model of our simple company might take the form:



There are several things to notice in the diagram above: Clouds have replaced the end levels; policies (albeit incomplete) have been put in place to determine the rates; and new kinds of variables have been added. The policies in place reflect those of the company (*payout time*) , those of its customers (*receipt time*) and those of the sales people in response to receiving good income (though the *sales calls* notion is a little more questionable than the other two).

Levels of Detail

Adding details to accounting systems is generally easy; all that is required is the creation of additional categories. By contrast, adding details to a dynamic model requires more than just another stock or two and the corresponding rates of flow. For every flow in a simulation model it is necessary to formulate a policy that determines that flow. Adding one stock and flow may create many new feedback loops. Therefore, in developing dynamic financial models we do not want to delve into the detail required for a fully functional accounting system. The choice of how much detail to include must depend on model purpose.

An important factor in determining the level of detail to include is the dynamic implications of introducing a finer structure. For example, suppose that when you sell a product you charge both for it and shipping. If you are setting up an accounting system you will want to set up two different accounts for these two revenue sources. For doing financial modeling, however, making the distinction would not be worthwhile. Every time a product is sold it is shipped, every time money is received for a product that has been sold the shipping charges are also received. Unless there are dramatic changes in the shipping mix, shipping charges can netted out or lumped into product sales without changing dynamics.

An Investment Evaluation Model

We are considering investment in a production facility for Thneeds, a made-to-order product. (Thneeds are an imaginary product from the Dr. Suess children's story *The Lorax*.) You know the nominal capacity of the facility, its expected cost, how long it will take to build, the price of Thneeds, variable production costs and the interest rate that will be charged by the bank. The question you need to address is whether or not the investment is a good idea.

At this point you will probably recognize that this is a very different problem than the ones we have been looking at. We are not trying to come up with a hypothesis about where behavior comes from but simply trying to track the consequences of some relatively simple assumptions. Building the financial portions of models often has this straightforward, almost mechanical, flavor. Constructing a model to answer this question does not present any large conceptual impediments. As a bonus, once the financial formulations are constructed we will see how easily they can be integrated into other models.

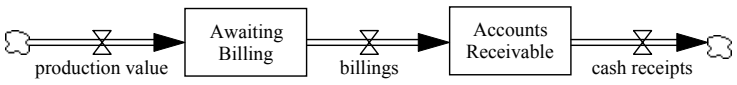
Sales and Receipts

A very simple formulation for revenues and profits would be:

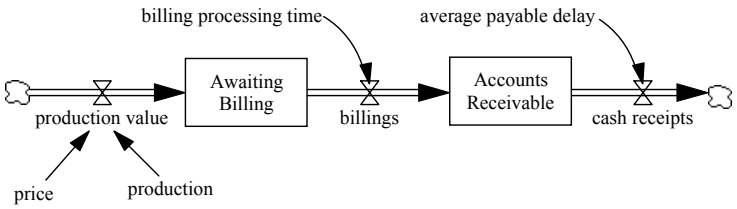
```
profit = revenue - cost
revenue = price * sales
```

While this formulation is perfectly sensible in looking at the big picture, it is not always appropriate for addressing issues in financial performance. This is especially if we are interested in seeing the differences between income and cash flow since this formulation implicitly assumes that the two are the same.

In order to look more carefully at the determinants of cash flow it is necessary to give somewhat more attention to the process of making sales and collecting money. Just the stocks and flows might look something like:



Here *production* value is used to set up a billing process that must first generate a bill and later receive payment on that bill. The policy formulations for *billing* and *cash receipts* are simple. Each is paid out over a period of time proportional to *Awaiting Billing* and *Accounts Receivable*.



The equation *billings* is just:

```
billings = Awaiting Billing/billing processing time
```

and there is a similar equation for `cash receipts`.

Equilibrium Initializations

One of the important formulation techniques that will be used in this model is the initialization of the levels to an equilibrium or steady state value. In equilibrium, the total inflows and outflows of each stock must be equal. To understand how this is accomplished consider what would happen in a structure such as that used above if *price* and *production* were both constant over a long period of time. As long as the product of *production value* is bigger than *billings*, *Awaiting Billing* will rise (or fall if smaller). But as *Awaiting Billing* rises, so will *billings*. Over time *billings* will rise until it is equal to *production value* and then *Awaiting Billing* will stop changing. With constant inputs, the above structure yields constant output for both *billings* and, by the same argument, *cash receipts*.

To work this backward we require that the outflow (*billing*) and the inflow (*production value*) to *Awaiting Billing* be equal:

```
billings = production value
```

substituting this into the equation for *billing* gives:

```
Awaiting Billing/billing processing time = production value
```

or

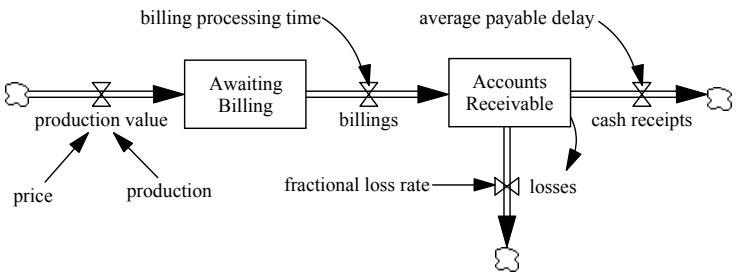
```
Awaiting Billing = production value * billing processing time
```

We can use this expression to initialize *Awaiting Billing*. Using the same logic we can initialize

```
Accounts Receivable = billings * average payable delay
```

When Vensim tries to compute an initial value for `Accounts Receivable` it will need to compute *billings*, and therefore *Awaiting Billing* and therefore *production value* and therefore *price* and *production*. Vensim automatically takes care of this computational sequence problem and will inform you if it discovers any anomalies.

With the above formulation *accounts receivable* only decreases when money is received. It is, unfortunately, also true that some people do not pay their bills and these need to be written off. We can add an outflow of losses to handle this problem:



where

```
losses = Accounts Receivable * fractional loss rate
```

The addition of *losses* complicates the selection of an initial value for *Accounts Receivable* somewhat. Now we need *billings* to be equal to *cash receipts + losses*. Substituting the different formulae we have:

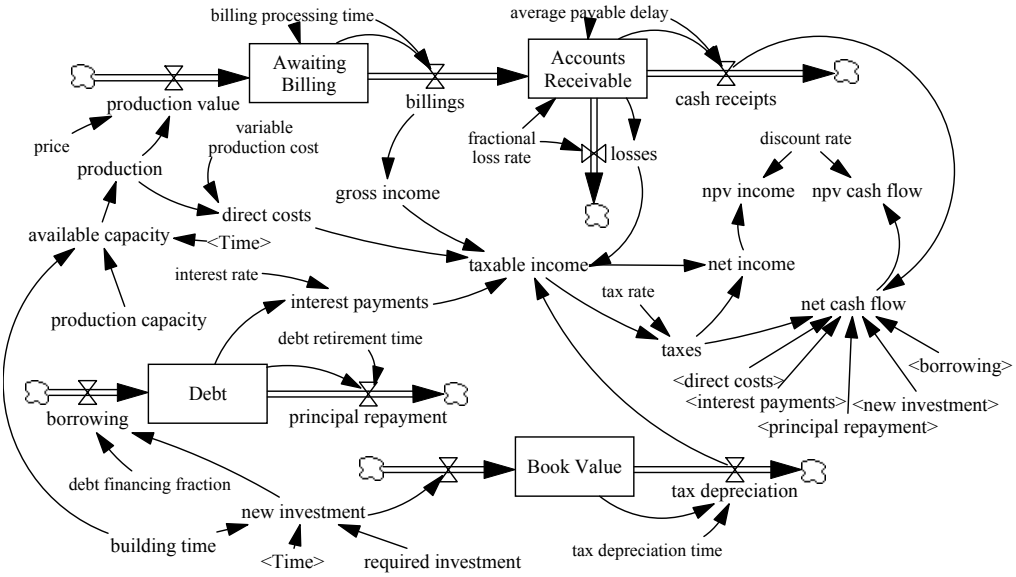
```
Accounts Receivable = billings /
         (1/average payable delay + fractional loss rate)
```

Here *fractional loss rate* is equivalent to the reciprocal of a time constant.

It is not necessary to initialize structure in equilibrium, but it can be very helpful. It is very common for the early part of a simulation run to be dominated by adjustments from imbalances in Levels and Rates. The dynamics generated by this imbalance do not tend to have intrinsic importance and can severely hamper the understanding of model dynamics. For this particular model, we will actually be starting with production at 0, so we could have simply set the two levels to 0 as well. However, the structure we are developing is designed to be easily used in other models and has been made more general with this in mind.

The Complete Model (financ01.mdl)

The remainder of the model is quite straightforward to construct. Usually, the most difficult issues involve representation of the tax system faced by the corporation. We have opted for a simple proportional tax that would be appropriate in many situations. We have also used proportional depreciation of book value for tax purposes and proportional debt retirement. These formulations were chosen because of their simplicity.



We have added in two additional Levels — *Debt* and *Book Value*. If you look for feedback in this model you will find only the short draining loops around each Level. This is a dynamically simple model with a reasonably high amount of detail around the *cash flow* and *taxable income*. The way the model is set up *required investment* occurs over *building time* and then production comes on line at capacity. The model, as constructed, is for a fixed up front investment followed by a realized cash flow from production.

The model uses the NPV function to compute a net present value for both *income* and *cash flow*. The NPV function is a dynamic function that takes as arguments a stream of payments, a discount rate, an initial value and an adjustment factor. We are using 0 as an initial value and 1 as the adjustment

factor. With these argument the NPV will, at the end of the simulation, report the present value of the stream as of the beginning of the simulation.

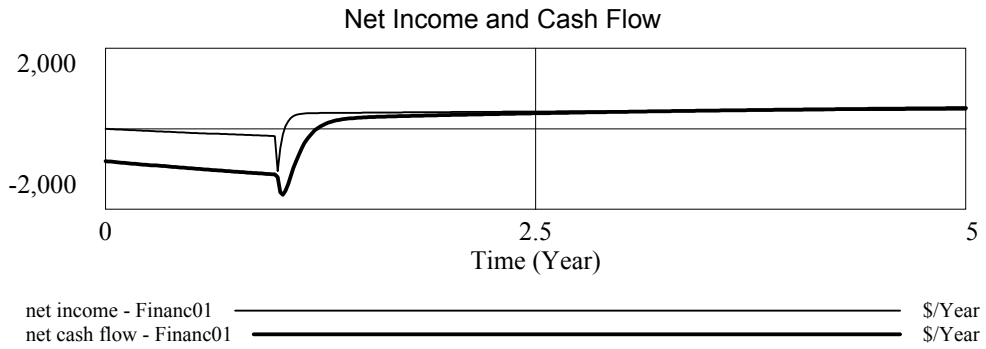
Model Equations

```
Book Value = INTEG(
   new investment - tax depreciation,
         0)
Units: $

tax depreciation = Book Value / tax depreciation time
Units: $/Year

taxable income = gross income - direct costs - losses -
   interest payments - tax depreciation
Units: $/Year

production = available capacity
Units: Widget/Year

available capacity = IF THEN ELSE ( Time >= building time,
    production capacity, 0)
Units: Widget/Year

tax depreciation time = 10
Units: Year

tax rate = 0.4
Units: Dmnl

Accounts Receivable = INTEG(
   billings - cash receipts - losses,
         billings/(1/average payable delay + fractional loss rate))
Units: $

average payable delay = 0.09
Units: Year

Awaiting Billing = INTEG(
   price * production - billings,
         price * production * billing processing time)
Units: $

billing processing time = 0.04
Units: Year

billings = Awaiting Billing / billing processing time
Units: $/Year

borrowing = new investment * debt financing fraction
Units: $/Year

building time = 1
Units: Year

cash receipts = Accounts Receivable / average payable delay
Units: $/Year

Debt = INTEG(
   borrowing - principal repayment,
         0)
Units: $

debt financing fraction = 0.6
Units: Dmnl
```

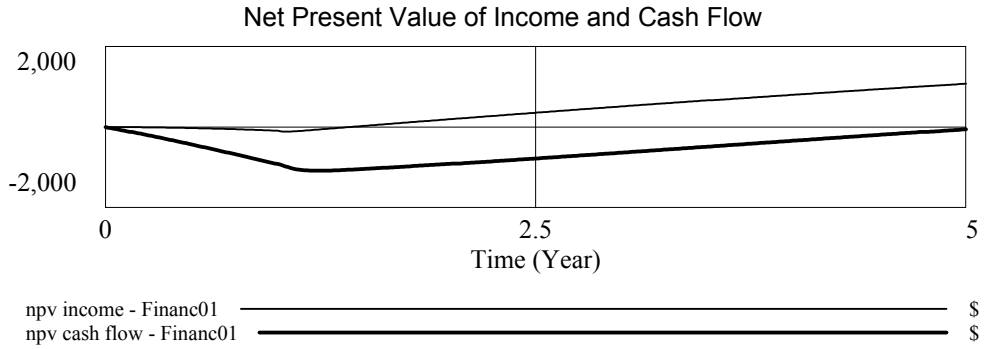
```
debt retirement time = 3
Units: Year

direct costs = production * variable production cost
Units: $/Year

discount rate = 0.12
Units: 1/Year

fractional loss rate = 0.06
Units: 1/Year

gross income = billings
Units: $/Year

interest payments = Debt * interest rate
Units: $/Year

interest rate = 0.12
Units: 1/Year

losses = Accounts Receivable * fractional loss rate
Units: $/Year

net cash flow = cash receipts + borrowing - new investment -
   direct costs - interest payments - principal repayment - taxes
Units: $/Year

net income = taxable income - taxes
Units: $/Year

new investment = IF THEN ELSE (Time >= building time, 0,
   required investment / building time)
Units: $/Year

npv cash flow = NPV (net cash flow, discount rate, 0, 1)
Units: $

npv income = NPV (net income, discount rate, 0, 1)
Units: $

PRICE = 1
Units: $/Widget

principal repayment = Debt / debt retirement time
Units: $/Year

production capacity = 2400
Units: Widget/Year

required investment = 2000
Units: $

taxes = taxable income * tax rate
Units: $/Year

variable production cost = 0.6
Units: $/Widget

TIME STEP = 0.015625
```

Simulation Results

We run the model for 5 years with TIME STEP = 0.015625.



At the end of the simulation *npv income* is $1,073 while *npv cash flow* is -$54.70 (this is on a total investment of $2,000. If we plot the present values over the course of the simulation we get:



There is a significant difference between the income and cash flow profiles — and this is a result of the manner in which income is computed.

Sensitivity Testing

From the above simulations, you could conclude that on an income basis the investment looks attractive, while on a cash flow basis it does not. We have, however, built a model that contains quite a few assumptions, and these assumptions are known to be uncertain. We could go in and change the assumptions one at a time and simulate the model to understand the implications. There is, however, an alternative to this, known as Monte-Carlo Simulation or Multivariate Sensitivity Simulation (MVSS). To use this, we will set ranges on the uncertain assumptions, then Vensim will simulate the model multiple times randomly selecting values for the uncertain assumptions.

Open the **Simulation Control** and create a sensitivity control file (e.g., *financ01.vsc*). Select the following parameters, distributions, and ranges (Chapter 11 of the *Tutoria*l has details on setting up and running Sensitivity Simulations):

```
average payable delay = RANDOM_UNIFORM(.07,.11)
billing processing time = RANDOM_UNIFORM(.03,.05)
building time = RANDOM_UNIFORM(.8,1.2)
fractional loss rate = RANDOM_UNIFORM(.05,.08)
interest rate = RANDOM_UNIFORM(.09,.15)
price = RANDOM_UNIFORM(.9,1.2)
production capacity = RANDOM_UNIFORM(2200,2600)
required investment = RANDOM_UNIFORM(1800,2200)
variable production cost = RANDOM_UNIFORM(.5,.7)
```

Be sure to choose the **Multivariate** option and set the number of simulations to 200.

In order to do a sensitivity analysis you will need to select a set of variables for which to view the results. Though Vensim normally stores all the results, this is not practical in the case of sensitivity analysis, and so we need to specify a shorter list. Chapter 11 of the Tutorial has information on setting up save lists. For this model it is useful to save (*financ01.lst*) :

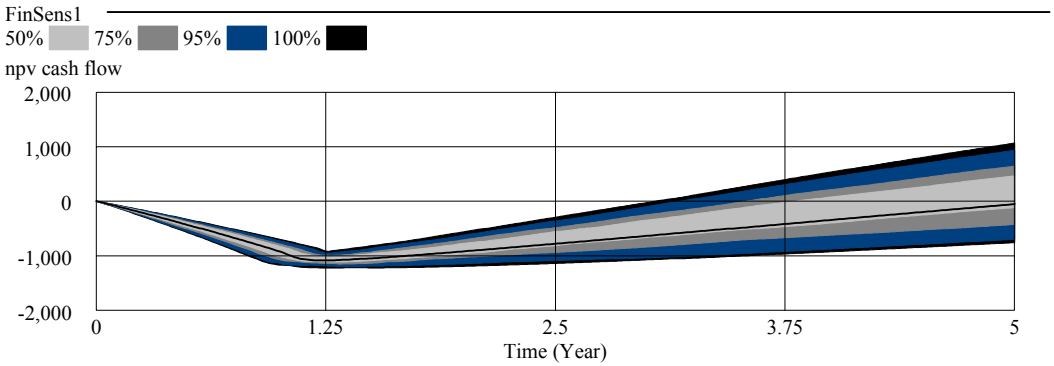
```
npv cash flow
npv income
net cash flow
net income
```

Now run the simulation. Vensim first does a normal simulation, and then the multiple sensitivity simulations. On a 133 Megahertz Pentium based computer this model takes a few seconds to run 200 simulations. Depending which Vensim configuration you are using and what computer you have this process can take significantly longer. Once the sensitivity simulations start you will see a window showing each simulation as it is finished. If this is going very slowly, you can stop before Vensim completes all 200 simulations by clicking on the **Cancel** button. The results, if you do stop early, are likely to be a little different from those reported here.

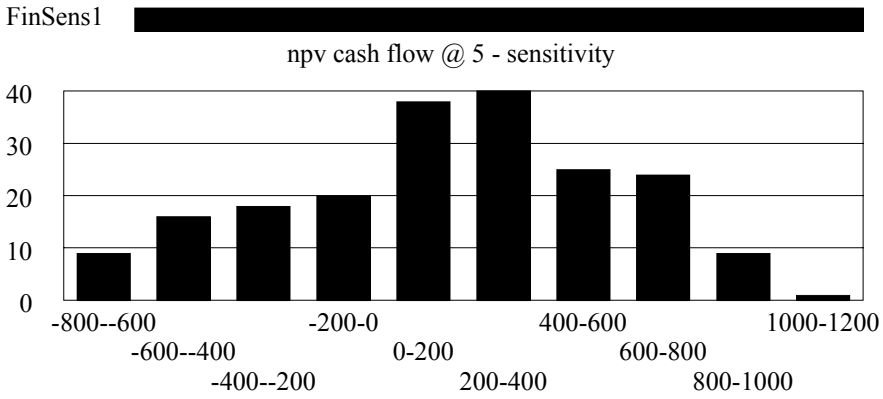
Displaying Sensitivity Results

There are three tools available for the display of sensitivity results: the Sensitivity Graph tool, the Bar Graph tool and, if you are working with Vensim Professional or DSS, the Stats tool. Chapter 11 of the Tutorial describes how to bring in a toolset with the Sensitivity tool available and Chapters 12 and 13 of the Reference Manual describe setting up and configuring the Analysis tools.

Select *npv cash flow* as the workbench variable and click on the **Sensitivity Graph** tool. You should see something like this:



This graph shows the uncertainty in the net present value of cash flow as it changes over time. At any time, half of the simulations have generated a value within the 50% region, three quarters within the 75% region and so on. Two things are worth noting about this graph. First the uncertainty grows over time - a natural result of the cumulative nature of a present value calculation. Secondly, the percentiles are not evenly spaced. We can see this more clearly with the Bar Graph tool . We need to check on the **Histogram** and **Sensitivity** checkboxes in the Bar Graph options. Now click on the Bar Graph tool and you will get the output:



Even though we have used a uniform (flat) distribution of uncertainty inputs, this graph indicates that the *npv cash flow* is distributed with something of a bell shaped curve. This is because we are combining together in this model a number of different independent sources of error. A well known result from statistics is that doing this tends to lead to variables that are normally distributed.

These results show how you can use sensitivity analysis to understand the implications of uncertainty in input on the likely results. The results for the model here show that modest uncertainty of the assumptions results in modest uncertainty around the value of the investment. An investment needs, of course, to be judged against alternative investments. The histogram shown above gives substantially more information than an expected present value in making that judgment.

Financial Modeling and Market Growth (financ02.mdl)

The financial model we developed was designed to help evaluate a single investment. The structure itself, however, is quite general. We can easily make use of it with the market growth model developed in Chapter 4 (*prod3.mdl*). That model already had the concepts of investment and production, and we will need to integrate these two into the financial model.

In order to join the two models, begin with the financial model. Select the entire model structure and copy it. Now open the market model (*prod3.mdl*) Create a new view. In the blank screen, use Paste Structure to insert the financial structure into the market model. Save your new model as *financ02.mdl* or another name you prefer. Chapter 8 of the *Vensim Tutorial* has instructions for pasting structure from one model into another.

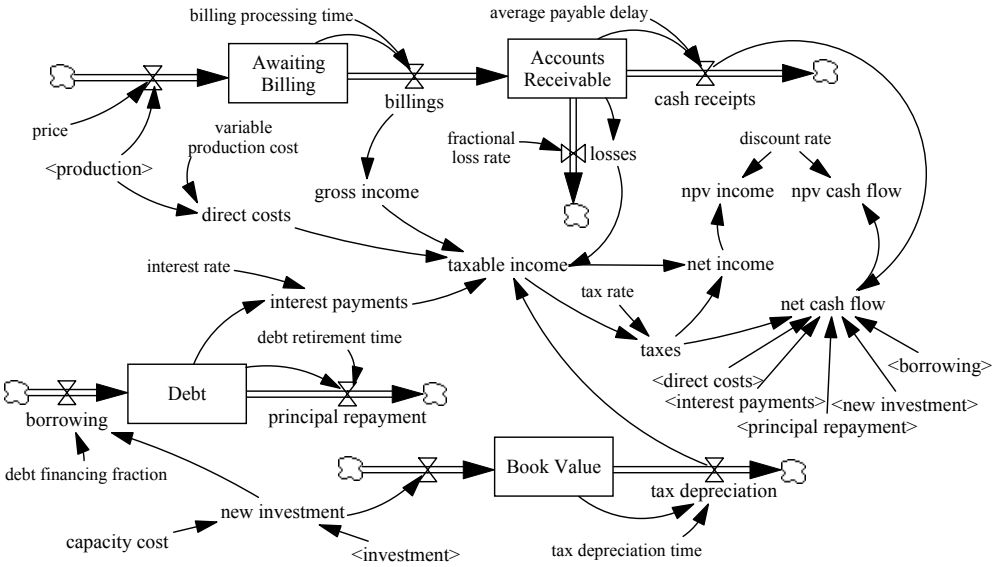
Looking at the view in the new model you will see that the financial structure looks the same except that *production* has been renamed *production 0*. This is because the variable *production* already existed in the market growth model. The renaming prevents a conflict from arising when the new structure with a different definition for *production* is pasted in. In this case, however, we want to use the old variable *production*.

Use the Shadow Variable tool to insert a copy of the market model variable *production* just below *production 0*. The variable *production* should appear in angle brackets <>. Now use the Variable Merge tool and drag *production* on top of *production 0*. You will be asked if you want to delete variable "*production 0*" and replace it with "*production*."

Make sure that you dragged *production* onto *production 0* and not the other way around. Answer No if you made a mistake. Otherwise click on Yes and the variable *production 0* will be replaced by <*production*>.

Now use the Delete tool and, in the lower right hand corner of the sketch, remove *required investment*, *building time*, *production capacity* and *available capacity*

from the model. These variables will no longer be used. Also remove the arrow from **<*Time*>** to *new investment*. You can then either hide **<*Time*>** or cut it from the view.



Click on the Model Shadow variable tool and add *investment* to the sketch near *new investment* and draw an arrow from *investment* to *new investment*. You will also need the add a new variable *capacity cost* as a cause of *new investment*. Your diagram should now look like the one above.

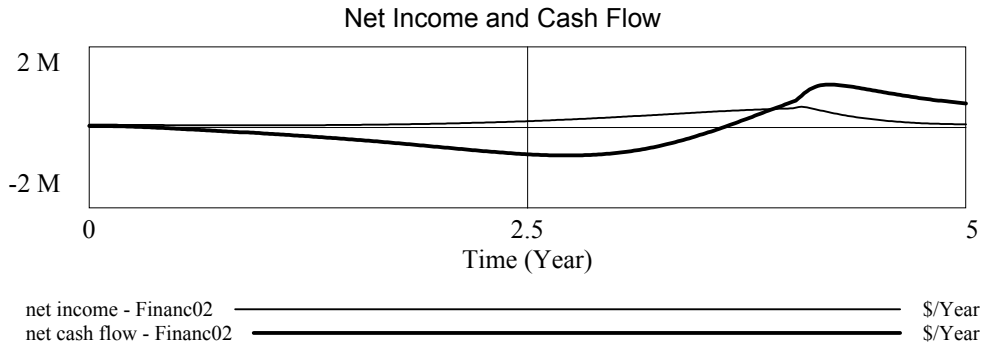
Click on the Equation Edit subtool and you will see *new investment* and *capacity cost* highlighted. The equations for these are:

```
new investment = investment * capacity cost
Units: $/Year

capacity cost = 0.5
Units: $/(Gadget/Year)
```

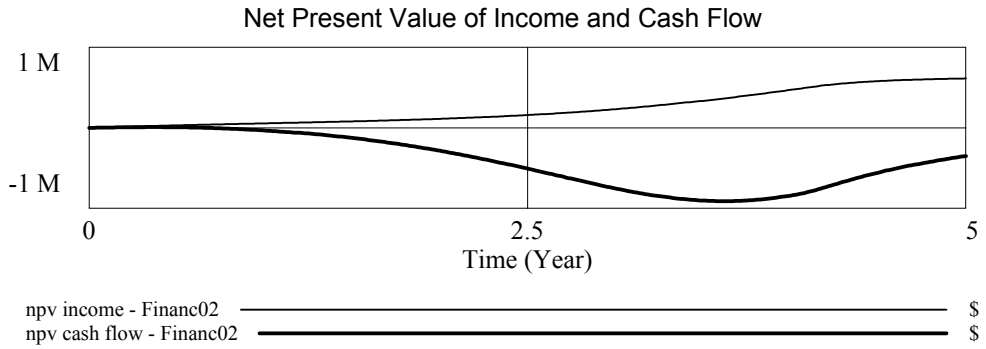
Change the value for of *tax depreciation time* to 2 years to reflect the shorter lived capacity assumptions in the production model. You will also need to change TIME STEP in the model to 0.015625 because of the short term dynamics around the billing process.

Simulation Results

Simulating this model gives the following:



There is a very important divergence between income and cash flow. Cash flow is negative for several years during the start-up phase of the industry. Income is falling when cash flow reaches its best levels. There is also a divergence of the net present values:

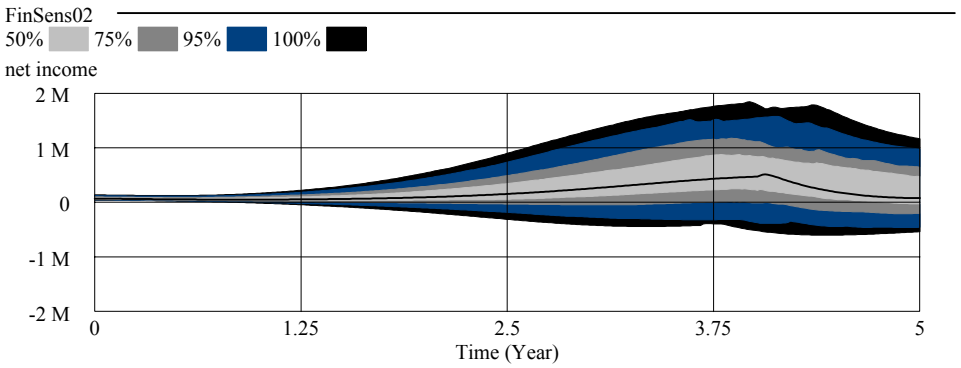


Sensitivity Tests

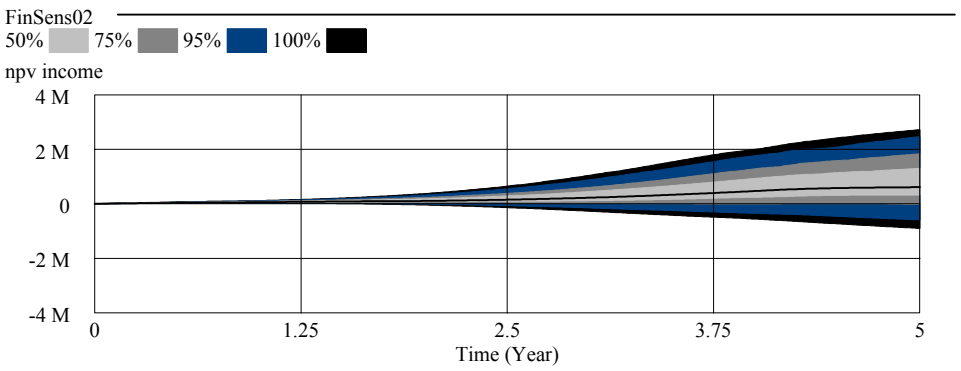
It is also possible to run sensitivity tests on this model. In this case we can consider uncertainties now only in the basic cost assumptions, but also in the basic market parameters. Using the sensitivity control file (*financ02.vsc*):

```
average payable delay = RANDOM_UNIFORM(.07,.11)
billing processing time = RANDOM_UNIFORM(.03,.05)
fractional loss rate = RANDOM_UNIFORM(.05,.08)
interest rate = RANDOM_UNIFORM(.09,.15)
price = RANDOM_UNIFORM(.9,1.2)
variable production cost = RANDOM_UNIFORM(.5,.7)
capacity cost = RANDOM_UNIFORM(.4,.6)
initial customers = RANDOM_UNIFORM(90000,110000)
sales fraction = RANDOM_UNIFORM(.004,.006)
```

We can simulate this and get the results on net income:



Here the range of possible outcomes is enormous. Similarly if we look at the net present value of income we get:



Conclusions

We have developed in this chapter a model that helps evaluate the financial consequences of investment plans, and also allows us to investigate the uncertainty surrounding these results. The model developed has not introduced any significant dynamics, but acted more as a measuring device. For a variety of business models including a financial sector is helpful as a means of conveying model results and understanding the implications of alternative policies.

In many cases, the financial sector in a model will also be an important part of the overall model dynamics. Feedback from cash flow and income changes peoples behavior. In some cases it may be necessary to curtail new investment, or defer maintenance spending in response to low profits or deficient cash balances. The model developed in this chapter provides some of the material to begin to incorporate these effects in your models.

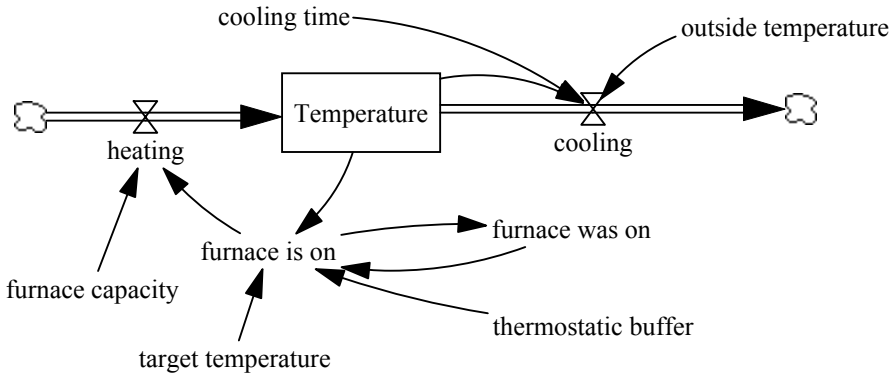
8 Furnaces, Pendulums and Oscillation

In Chapter 2 we went through a detailed development of the Workforce-Inventory oscillator. Although useful in its own right, that model is most important for the insights it provides into the process of oscillation. In this chapter we review two additional models for different problems that share the same basic structure as the Workforce-Inventory model. Each produces oscillations, and each has a special characteristic that requires special attention.

Thermostatic Control (thrmstat.mdl)

A simple mechanical system that is often encountered is a thermostatic control attached to a furnace. This system is interesting because it is designed to oscillate. The purpose of a thermostat is to keep a room in a comfortable temperature range, without constantly switching the furnace on and off. To do this, the control must establish a buffer range. When the temperatures goes below the bottom of the range the furnace comes on, when it goes above the top of the range the furnace turns off. For all temperatures within the range the furnace stays on if is it on, and stays off if it is off.

The reason that this is different from the other models we have considered so far is that it involves a discrete event - the furnace turning on or off. To make matters worse, the way the problem has been described, the switching of the furnace is dependent on whether the furnace is already on. This seems to be a paradox. The solution to the paradox, in this case, has to do with the granularity of time. The switching on and off of a furnace is not really a discrete event, it is just relative to the processes of heating and cooling that it seems instantaneous. For modeling purposes, we resolve this paradox by tracking the last known state of the furnace:



The equations for the model are:

```
Temperature = INTEG(heating-cooling,70)
Units: Degrees Fahrenheit

cooling = (Temperature - outside temperature)/cooling time
Units: Degrees Fahrenheit / Hour

cooling time = 8
Units: Hour

outside temperature = 35
Units: Degrees Fahrenheit
```

The formulation for cooling is the same as the stock adjustment formulation for *net hire rate* used in the Workforce-Inventory model. If *heating* is zero the cooling effect will take

Temperature from its current value toward *outside temperature* over *cooling time*. The interpretation of *cooling time* as the average time for an individual particle of the room to cool is not useful here since individuals are molecules and the concept of temperature requires averaging over molecules. *cooling time*, instead, should be thought of as the time over which most (63%) of the gap between the current room temperature and the target or equilibrium temperature is closed. For example, if the outside temperature was 0, and the inside temperature was 100, then after 8 hours (with the furnace off) the inside temperature would be 37. The temperatures would never become exactly equal to 0, but after 24 hours it would be very close (5 degrees).

```
heating = furnace is on * furnace capacity
Units: Degrees Fahrenheit /Hour
```

```
furnace capacity = 10
Units: Degrees Farenheit/Hour
```

heating represents the addition of energy to the house. It is very much like turning on a faucet and letting water run into a sink. The logic for turning the furnace on is what keeps the temperature within the target range.

```
furnace is on = IF THEN ELSE(furnace was on :AND:
  Temperature < target temperature + thermostatic buffer,1,
  IF THEN ELSE(Temperature < target temperature -
    thermostatic buffer,1,0))
Units: Dmnl
```

This formulation indicates that if the furnace is already on, and *Temperature* is below the target plus the buffer, the furnace should be left on. Otherwise, the furnace is turned on only if *Temperature* is below the target minus the buffer.

```
target temperature = 70
Units: Degrees Fahrenheit
```

```
thermostatic buffer = 2
Units: Degrees Fahrenheit
```

```
furnace was on = DELAY FIXED(furnace is on,0,0)
Units: Dmnl
```

furnace was on needs to have a memory so it must be a dynamic variable. The *DELAY FIXED* function provides this memory. The minimum delay for the *DELAY FIXED* function is *TIME STEP*, and the delay time of 0 is used to emphasize that we are concentrating on the most recent state of the furnace. It would be possible to formulate *furnace was on* as

```
furnace was on = INTEG((furnace is on - furnace was on) /
         TIME STEP,0)
```

This would give the same results, but might have numerical problems leaving *furnace was on* with a non-zero value different from one. This is especially likely to be a problem if an integration technique other than Euler integration is used.

You will notice that the formulation of *furnace was on* always starts this variable at 0. If you were to attempt to start this variable at the value of *furnace is on*, as in:

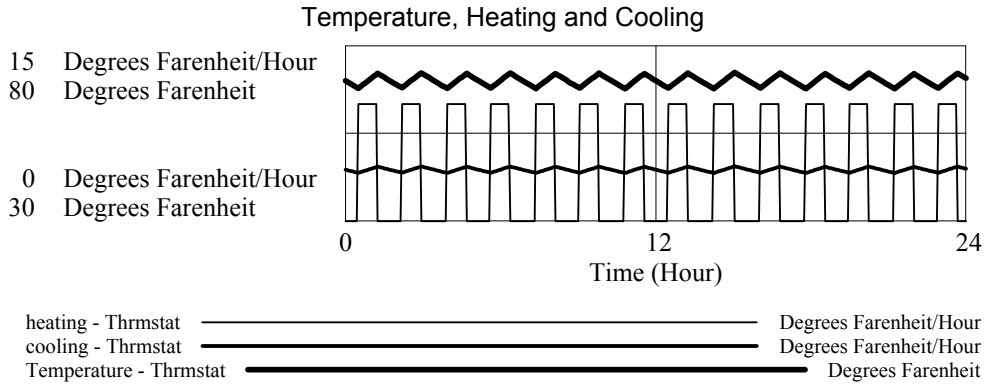
```
furnace was on = DELAY FIXED(furnace is on,0,furnace is on)
```

a simultaneous initial value condition would result. You need to give *furnace is on* an initial numerical value to get simulation underway.

The control constants have been set to run this model for 24 hours in intervals of 1/16 (.0625) of an hour.

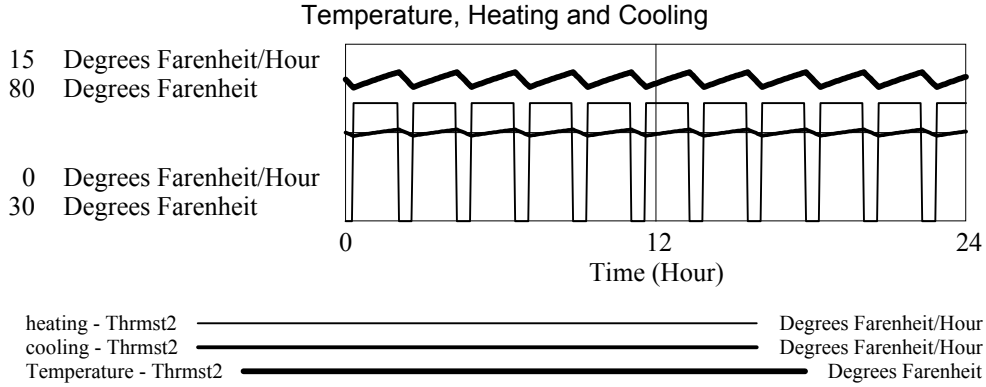
Simulation

When you simulate this model you get the following behavior:



Temperature follows a sawtooth pattern over the course of the day. The rate of cooling varies modestly around a value of 4 Degrees Farenheit/Hour. The furnace switches from on for a little less than an hour and off for a somewhat longer period of time.

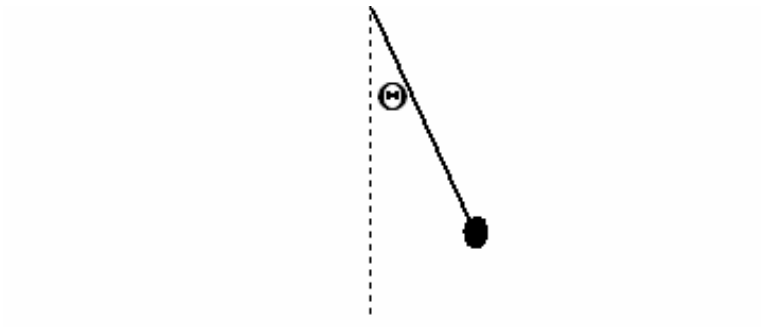
If we do an experiment in which *outside temperature* is lowered to 10 degrees (from 35) we get the behavior:



Here the furnace is on much more, heating takes longer and cooling is more rapid.

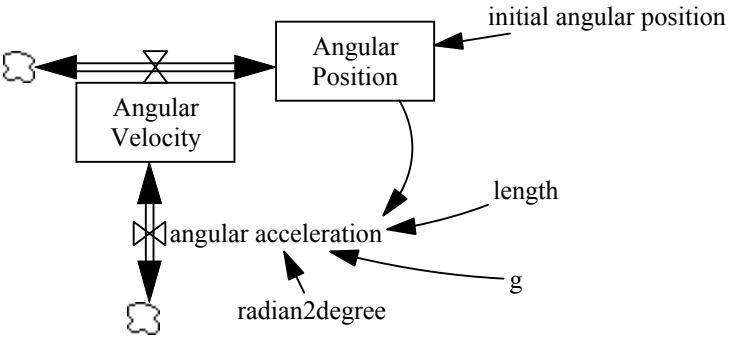
The Pendulum (pendulum.mdl)

A pendulum consists of a weight attached to a string suspended from a hook or other solid attachment. It is a common and well understood device that has been in used in clocks for several centuries. The reason that pendulums are valuable for time keeping is that they maintain a relatively constant period so long as their range of motion is small.



If we let Θ represent the deviation of the pendulum from straight up and down, then the force on the pendulum can be decomposed into a component parallel to the string (*mg* cos(Θ), where m is the pendulum mass and g is the acceleration of gravity) and a component perpendicular to the string (*mg* sin(Θ)). Since the weight can only move perpendicular to the string, the accelerating force is perpendicular to the string, so that the linear acceleration is just *g* sin(Θ). We can convert this linear acceleration to a radial acceleration by dividing by the length of the string.

We represent this physical problem with the diagram:



Notice the cascaded levels. The acceleration of an object changes its velocity, which in turn changes its position. This means that there is a rate which is also a level. Though this makes good sense for this example, such a situation is actually quite rare for nonphysical models. In the Workforce-Inventory example *Inventory* is analogous to *Angular Position*, and *Workforce* is analogous to *Angular Acceleration*. In that case, however, *Workforce* does not directly cumulate into *Inventory*, but is instead responsible for a process (*production*) that cumulates into *Inventory*, all of which introduces more loops as discussed in Chapter 1. This model has only a single feedback loop.

For simplicity, this model uses angles computed in degrees. *radian2degree* is a constant (=360/2π) that converts between radians and degrees, and is used in the computation of sin(Θ). To maintain dimensional consistency within the equations the unit of measure Radian is considered to be equivalent to dimensionless.

The equations for this model are:

```
Angular Position = INTEG(Angular Velocity,initial angular position)
Units: Degree

Angular Velocity = INTEG(angular acceleration,0)
Units: Degree/Second

angular acceleration = -radian2degree*
      SIN(Angular   Position/radian2degree) * g / length
Units: Degree/Second/Second
```

In this equation you will notice that *Angular Position* is converted from degrees to radians by dividing by *radian2degree*. The result (-sin(Θ) *g*/*length*) is then in radians and is converted back to degrees by multiplying by *radian2degree*.

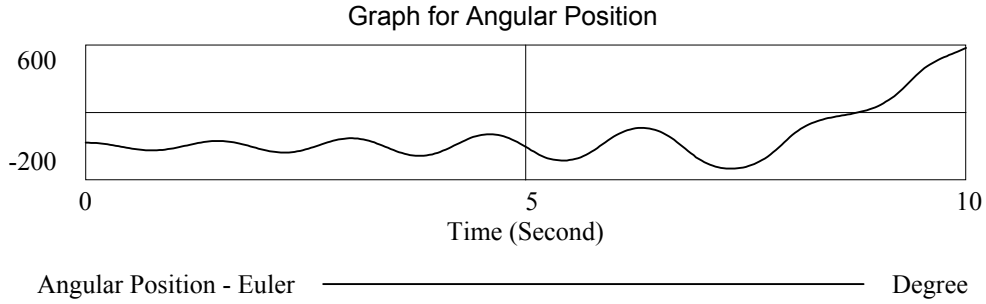
```
g = 9.2
Units: Meter/Second/Second

initial angular position = 20
Units: Degree

length = 0.5
Units: Meter

radian2degree=57.296
Units: Degree
```

Simulating the Pendulum

If you have ever solved the equations of motion for a pendulum all of the above should seem very familiar with one exception. Normally the simplifying assumption sin(Θ)=Θ is made. Because we are simulating these equations, rather than solving them explicitly, the simplification is not necessary. In fact, we can use this model to understand the implications of larger swings on the period of the pendulum, something that is not possible if the equations are simplified.

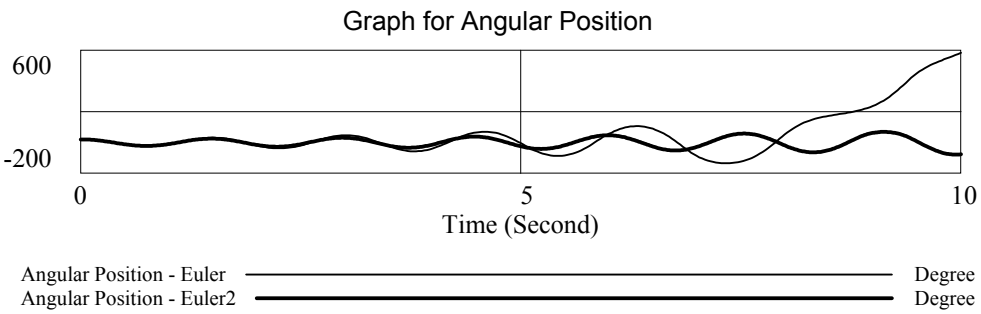
Euler Integration

We want to simulate this model for 10 seconds at intervals of 1/32nd of a second (.03125 seconds). If you attempt this using Euler integration you will get the surprising result:



If you are puzzled, this graph says that if you raise the weight to 20 degrees and let go, the pendulum goes back and forth a few times, and then begins to spin around its axis. While this may be consistent with a child's notion of how a swing ought to work, you would be hard pressed to design a pendulum to do this. Something is wrong, and it has to do with integration.

When you are using Euler integration and suspect that it is giving bad results, a good test is to cut the integration period in half, and see if the results change. We can do this by making the simulation run Euler and changing *TIME STEP* to .015625.:



The results change a great deal, but still do not make sense. The oscillations are still growing, and rotation would likely result in a few more seconds.

In order to get good results from this model using Euler integration you need to make TIME STEP very small indeed (< .001). If you do want to try this be sure to change *SAVEPER* to be a number (.0625 or .03125) or Vensim will try to store 10,000 values for each variable and, most likely, fail to do so. There is, however, a better solution.

The reason that Euler integration fails for this model is that this model represents an undamped oscillator on the edge between stability and instability. Euler integration is a simple linear extrapolation method, and when you try to extrapolate for something on a curve, you always overshoot the turning point. Normally a small overshoot is not much cause for concern, but in this case it provides a little bit extra displacement on each cycle, and that adds up to give complete divergence.

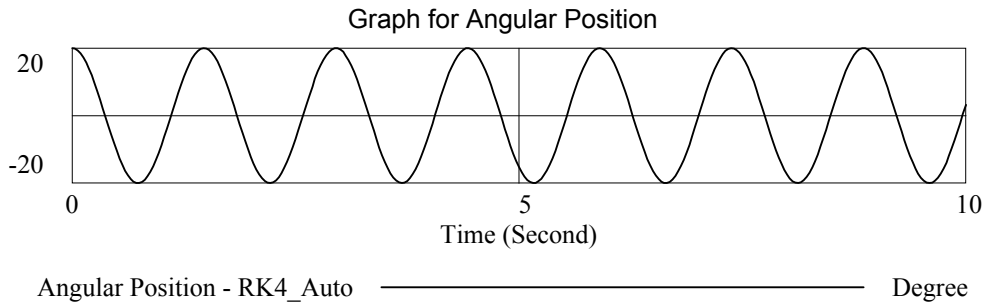
Runge-Kutta Integration

Vensim provides alternative integration techniques for dealing with models such as this. These methods, referred to as Runge-Kutta Integration, provide higher order extrapolation, looking at both the trajectory and how the trajectory is changing to give a better solution.

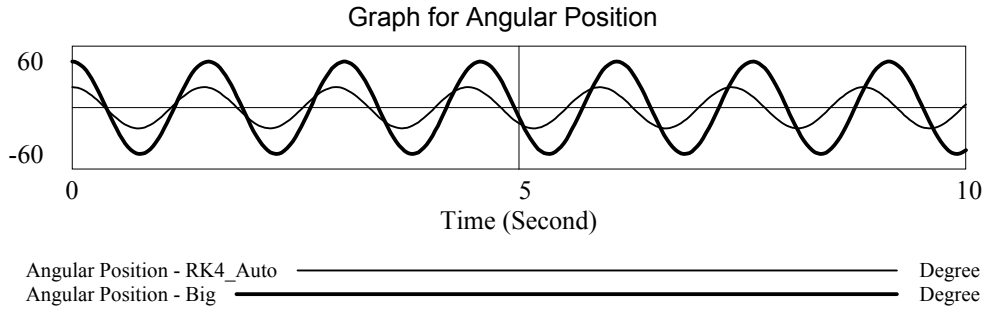
The different choices under Runge-Kutta are RK4 Auto, RK4 Fixed, RK2 Auto and RK2 fixed. These use higher order approximation (2nd and 4th) to the underlying continuous system. In general, RK4 Auto is the most accurate technique to use. It performs, automatically, the experiment of decreasing *TIME STEP* as done above to check accuracy. If accuracy is below an acceptable tolerance, the integration interval is decreased further until the desired accuracy is obtained. This is not always possible, and you will sometimes receive warning messages about being unable to achieve the desired accuracy. RK4 Fixed is the same as RK4 Auto except that it does not do this accuracy checking. RK4 Fixed is faster, but you need to test, just as you do for Euler, to see if accuracy is a problem.

It is important to note that when you are using the Runge-Kutta integration techniques you will not see all intermediate computations. *TIME STEP*, and therefore normally *SAVEPER* are not affected by the integration technique. With any of the Runge-Kutta integration techniques there will always be computations made between *TIME STEPS*. You can, consequently, see a level that does not seem to be the accumulation of its inflows and outflows. For truly continuous systems this will not matter, but if there is any switching going on things can get very confusing.

For small models it is best to use RK4 Auto. The other integration techniques are intended to allow you to trade off accuracy and simulation time to fit your needs. When you simulate this model using RK4 Auto you get the results:



This is more like what you would expect - continued oscillation to plus and minus 20 degrees. Changing the initial position to 45 degrees (no longer a small deviation) gives:



You get the same qualitative behavior, but the period is longer. This is different from what is typically learned in introductory physics, in which the period is constant due to the approximation of sin(Θ)=Θ. You might change the model so that:

```
angular acceleration = -radian2degree *
          (Angular   Position/radian2degree) * g / length
```

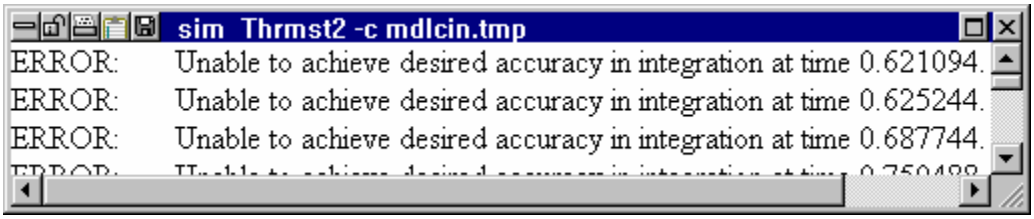
and see what behavior results.

More on Runge-Kutta Integration

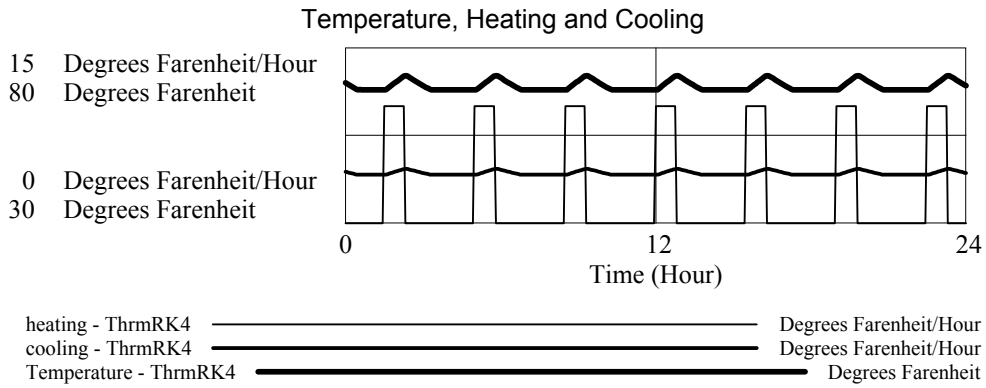
We have looked at two models in this chapter and one of them required the use of Runge-Kutta integration in order to get good results. We did not use Runge-Kutta on the Thermostat model and there was a reason for that.

Runge-Kutta Simulation for the Thermostat Model (thrmstt2.mdl)

The results for the Thermostat model were obtained using Euler integration. If you try to simulate this model using RK4 Auto you will get the messages:



. . . . . and the results:



This is different from our previous results, and clearly wrong. Temperature is flat for long periods of time during which the furnace does not appear to be on. As we have formulated this model it is inherently discontinuous, and Runge-Kutta integration is not going to help. The flat spots in *Temperature* are the result of computations within *TIME STEP*. What is happening is that during computation the trigger point to turn the furnace on is reached. However, *furnace was on* remains at 0 during this time because it is a discrete delay and discrete delays change only at *TIME STEP*.

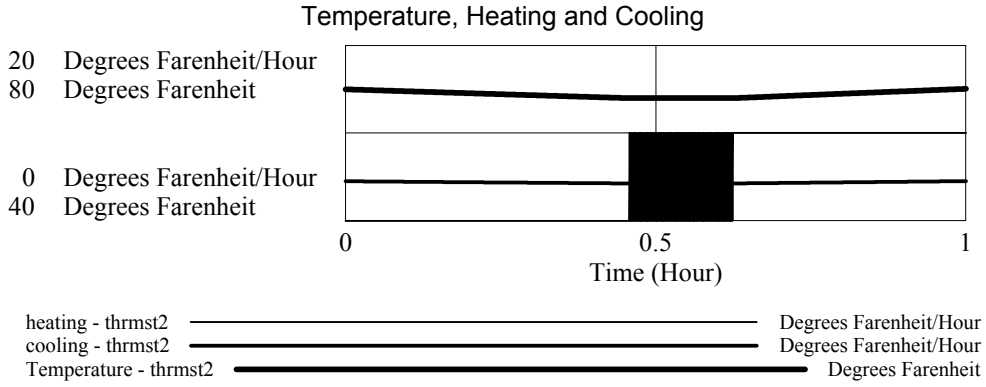
It is not possible to look inside of the integration technique, but we can reformulate the model slightly to get a picture of what is happening (*thrmst2.vmf*). We introduce a new variable *sampled furnace is on* that is the same as *furnace is on* at the old *TIME STEP*, but retains that value when *TIME STEP* is made smaller. This is done with the equation:

```
sampled furnace is on = SAMPLE IF TRUE(MODULO(Time,0.0625) = 0,
        furnace is on,furnace is on)
```

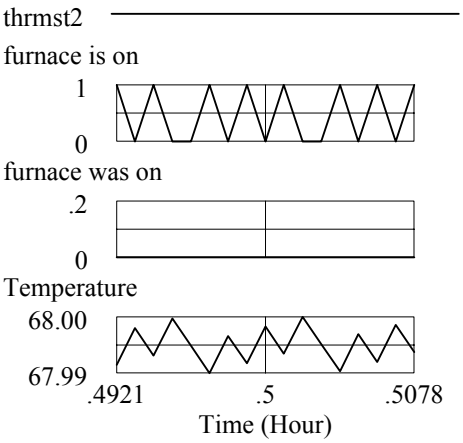
This equation forces *sampled furnace is on* to hold its value until *Time* is a multiple of 0.0625, then switch to the new value. *furnace was on* is rewritten to depend on *sampled furnace is on*

```
furnace was on = DELAY FIXED(sampled furnace is on,0,0)
```

This model has exactly the same behavior as the first model when *TIME STEP* is .0625. However, if we set *TIME STEP* to 1/1024 (0.0009765625) and reduce the length to 1 (we have also set *OUTSIDE TEMPERATURE* to 34) we get:



There is a flat spot in temperature during which the furnace cycles from on to off at every time step (appearing as a dark rectangle). If we zoom in on this time period (by holding down the shift key and dragging the mouse) we can look at the causes of *furnace is on*.



If you look at the time scale you will see that we are focused in on a very short period of time. *Temperature* is very nearly constant. The furnace is switching on and off to hold it constant. *Furnace was on* remains 0. The reason that *furnace was on* remains 0 as long as it does is because it is sampled only at multiples of .0625. Since the furnace is flickering on and off, it can take a while to catch the instant in which the furnace is on and switch to searching for the shutoff condition.

It is important to emphasize that what we are investigating here is the behavior of simulation techniques for this model, and not the behavior of the physical system. Were we to attempt the construction of a thermostat without a buffer zone these issues would apply directly to the physical system.

Conclusions on Integration Techniques

The dramatic differences between the results of Euler and Runge-Kutta integration are rarely as clear as they are in this chapter. For most models of social systems the different techniques do not lead to dramatically different results. For physical systems, in which the interrelationships are exact and based on physical laws Runge-Kutta integration is almost always preferable. The one exception to this rule is for models such as the Thermostat model in which we have explicitly introduced discrete time concepts. If you use any discrete time concepts, especially any of the DELAY functions, you should probably stay with Euler integration.

9 Discrete Functions

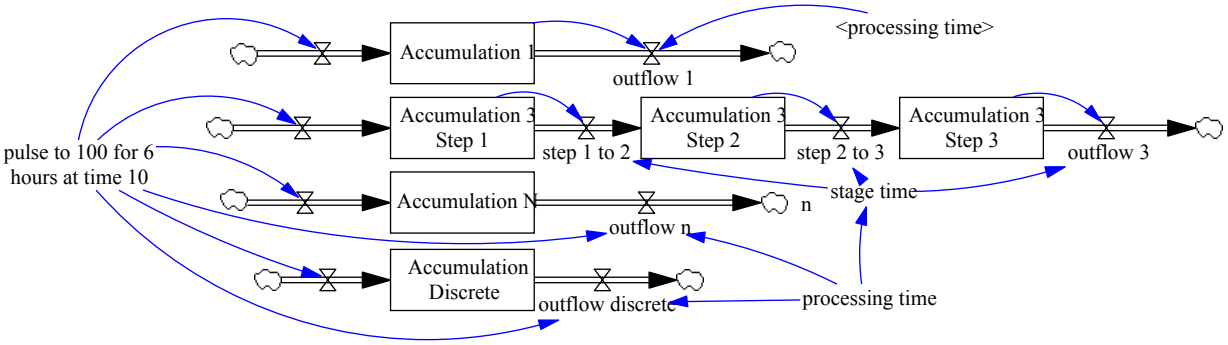
Most system dynamics models are made up entirely of continuously formulated stocks and flows. For most problems that deal with aggregate structures this is perfectly appropriate. Adding up a large number of discrete activities typically gives you something that is as easily approximated by a continuous formulation as it could be by a discrete formulation. Still, there are situations where it is simply easier to make use of discrete formulation and Vensim has a number of functions that can help do this.

This Chapter is really just a series of examples on how Vensim's discrete functions can be used to represent different processes. The purpose is basically to provide some useful how to material for creating your own models. This is also the purpose of the Molecules, as discussed in Chapter 10.

The only discrete function supported in Vensim PLE and PLE Plus is DELAY FIXED.

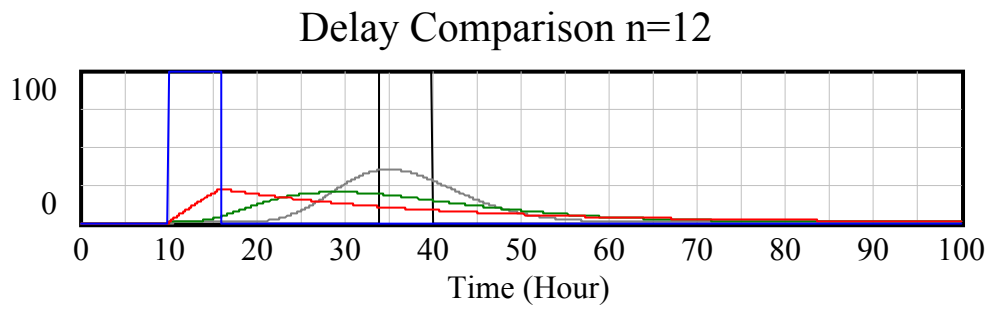
Continuous Versus Discrete Delays

Before we get started on the discrete examples it is useful to see the continuum that exists between discrete and continuous models. This is most vividly displayed by looking at a model that has several types of delays *mguide\9discret\discon.mdl*..

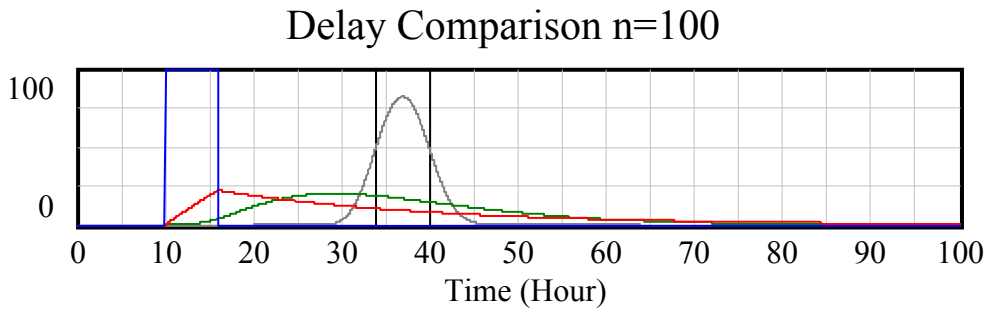


In this model *outflow 1* is the outflow of a first order delay, *outflow 3* a third order delay, *outflow* n and nth order delay (*n* is a model constant) and *outflow discrete* a discrete or infinite order delay. The order of a delay is determined by the number of levels involved in the delay. For the first and third order delays this number is obvious. For the nth order and discrete delays the number of levels is hidden in the internal structures used to compute the equations.

Now look at the behavior of this model when n is 12:



and when n is 100



It is useful to experiment with this model, though it does become slow to simulate with large values for n. The key thing to observe is that as n increases we get closer and closer to a discrete delay, but never actually get there. Even with a 100th order delay there is a noticeable difference, though if you compare *Accumulation N* and *Accumulation Discrete* they are not much different.

As a note, for the discrete delay the number of levels is not really infinite, it is actually equal to *processing time/TIME STEP*. The output of the high order delay functions, such as *DELAY FIXED* and *DELAY N*, are held constant across each *TIME STEP*. This allows alternative integration techniques to be mixed with discrete delays in a sensible manner.

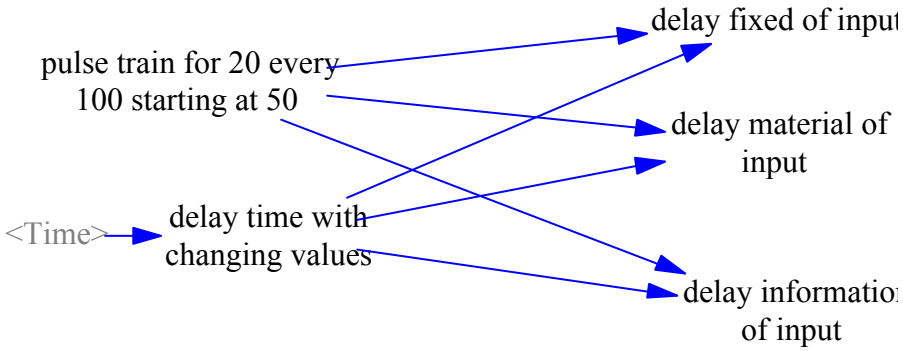
Material and Information Delays

There are three basic discrete delay functions: DELAY FIXED, DELAY MATERIAL and DELAY INFORMATION. The first requires a fixed delay time and, with a fixed delay time all the functions behave the same. It is only when the delay time changes over time that differences emerge.

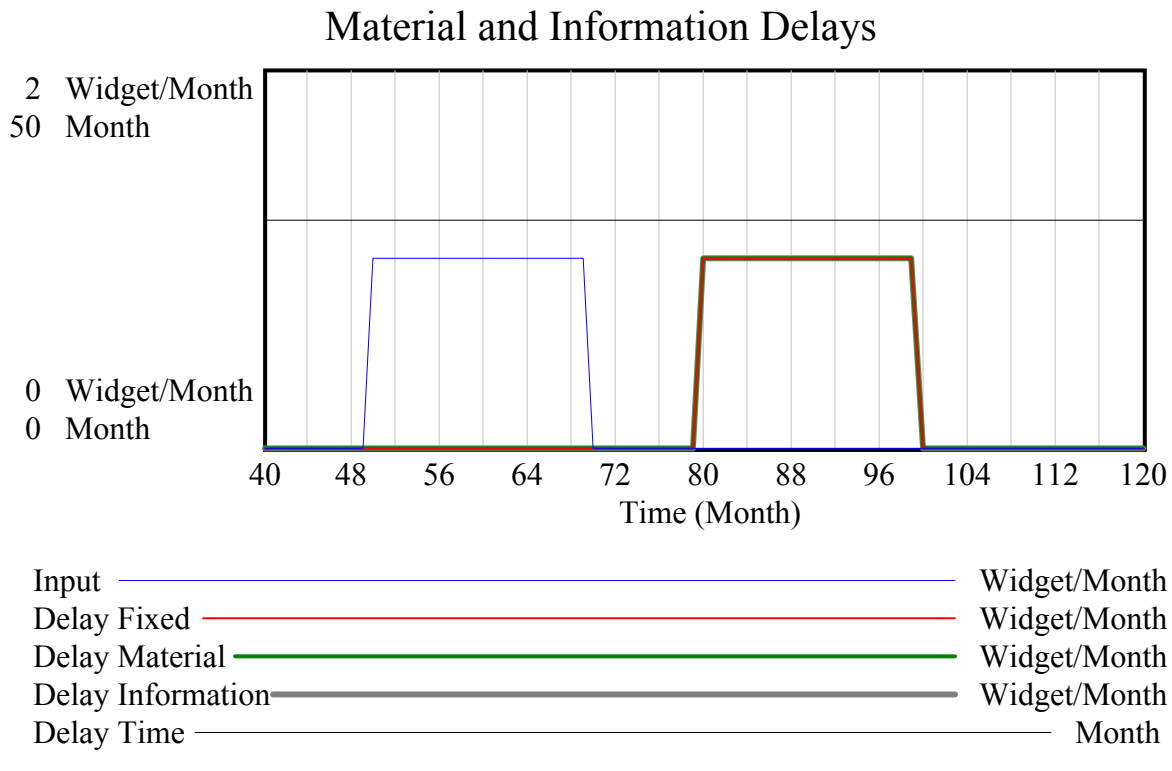
Material Delays including, DELAY MATERIAL, DELAY1, DELAY3 and DELAY N all have the property that they preserve quantities. That is, the accumulation of all the input is matched to the accumulation of all the output. This means when the delay time decreases, the amount of output will increase and vice versa.

Information Delays including DELAY INFORMATION, SMOOTH, SMOOTH3 and SMOOTH N all preserve the range of the input. That is, these functions will never return a value bigger than the biggest input value or smaller than the smallest input value no matter what happens to the delay time. This is completely appropriate for situations where the delay is being used to slow a signal (for example from *price* to *target production*) to account for the delays in processing information. If you were to formulate *expected price* using a material delay and the *perception time* decreased it would cause an increase in *expected price* even if *price* were constant.

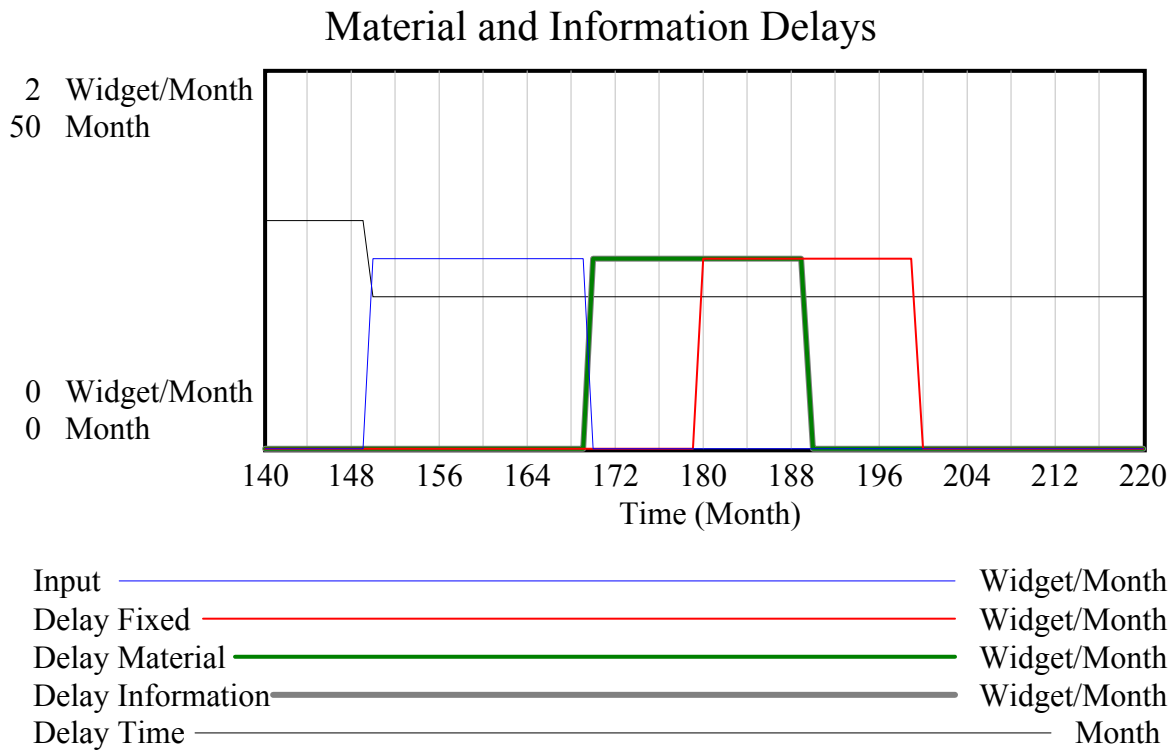
The model *delay1* demonstrates these three functions and how they change in response to changing delay time.



In this model the delay time is set up to highlight these difference. From time 50 to 100 it is constant and all the functions do the same thing:

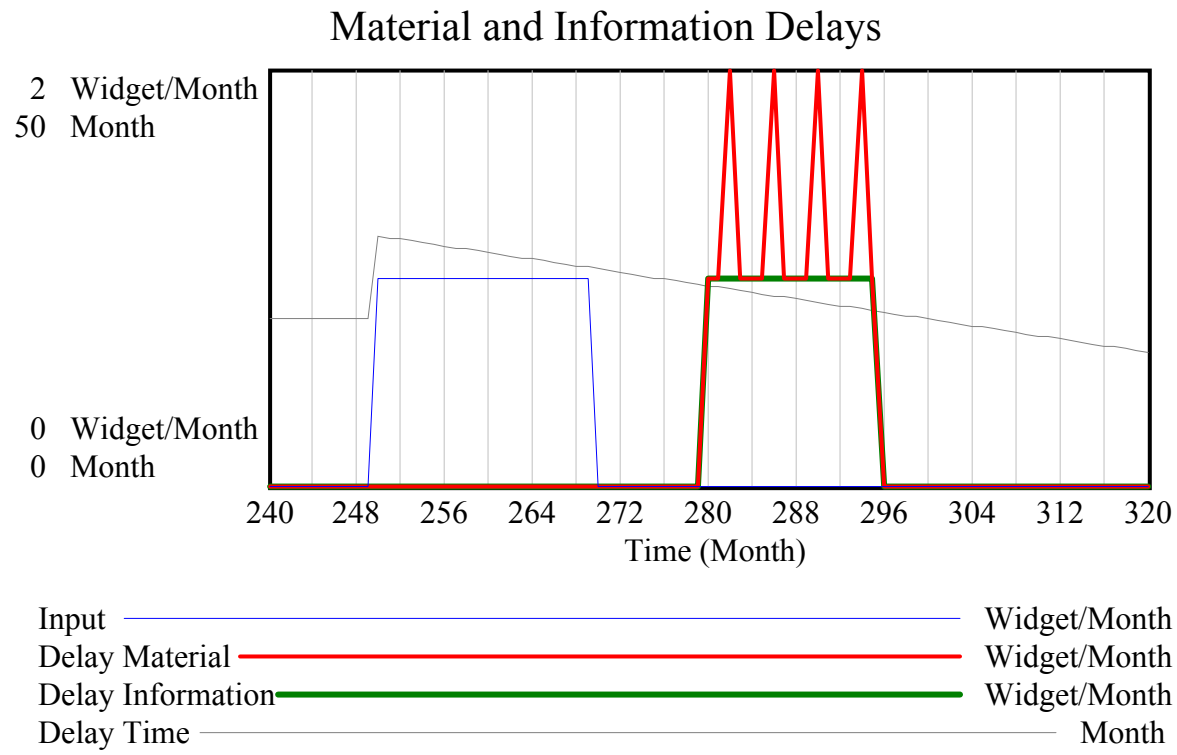


Then from Time 150 to 200 it is constant at a different value:



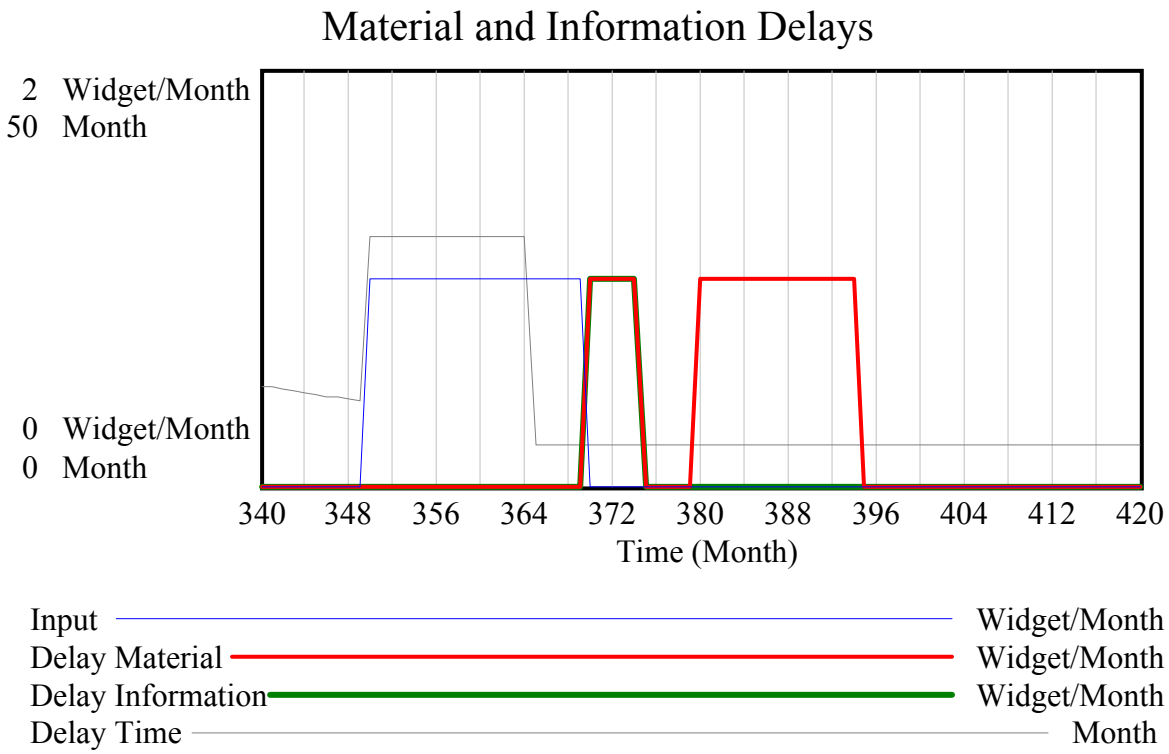
The DELAY FIXED function uses the delay time that it initially had (30) while the other two use the changed delay time of 20. No matter what happens to the delay time the DELAY FIXED function will always use its initial value. The fixed delay is dropped in the following examples.

If the delay time decreases over time we get



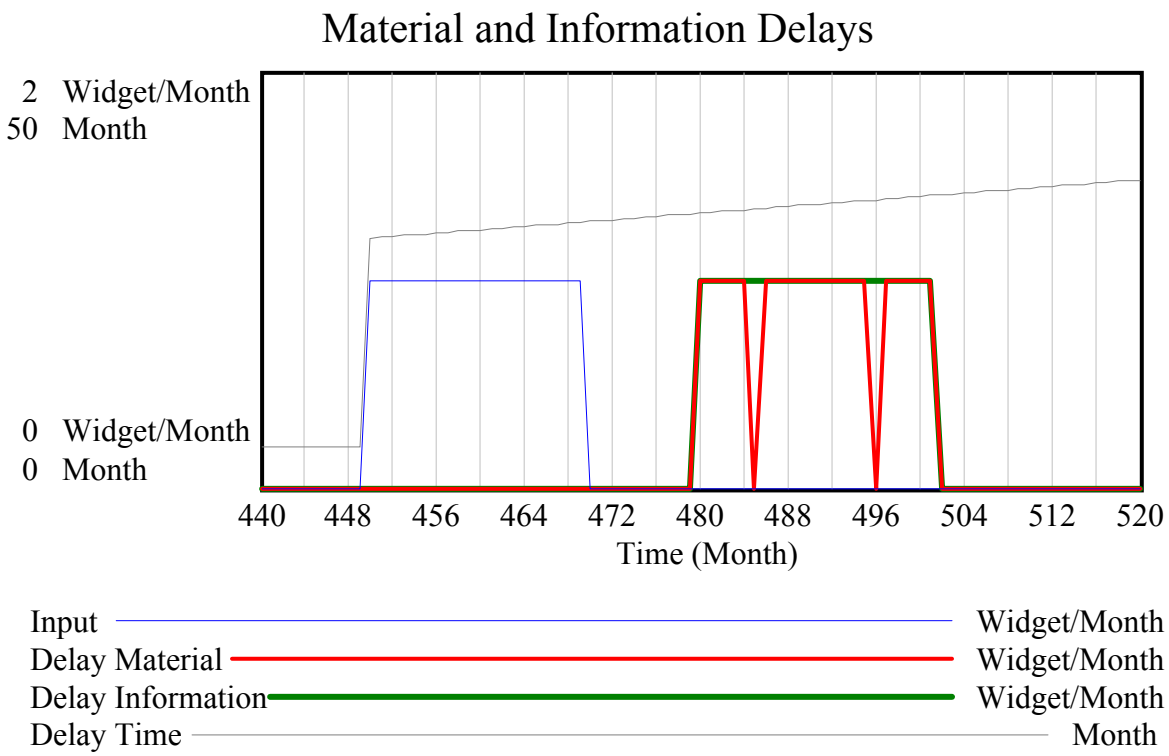
The information delay has a constant output, but for a shorter duration. The material delay has the same duration, but throws out some additional material on several occasions so that the total output is the same as the total input (20 Widgets).

If the decrease in delay time is more sudden we get:



Here the widgets in the material delay actually pass those put into the delay earlier and come out first. With the information delay, on the other hand, that information is treated as obsolete and discarded.

Finally when the delay time is increasing:



The duration of output is increased. With the material delay this means that at two times nothing comes out while the information delay simply holds constant at 1 widget/month.

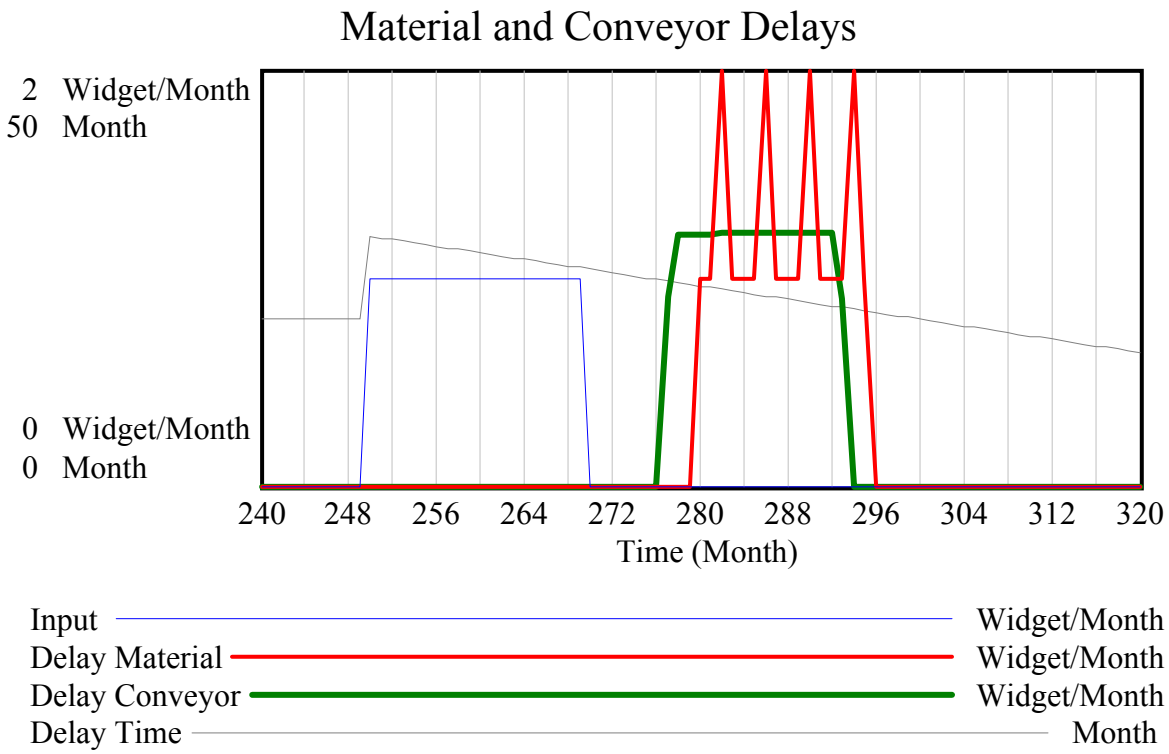
A note about material passing other material. If you do not want this to happen you can use the DELAY CONVEYOR function. See the discussion of this below.

Conveyors

The DELAY CONVEYOR function allows you to introduce a delay process that is analogous to putting material on a conveyor and then possibly speeding up and slowing down that conveyor. The DELAY CONVEYOR function has the additional capability of draining the elements on the conveyor which breaks the analogy somewhat but can be quite useful.

If the DELAY CONVEYOR function is used with a fixed delay time and no leakage than it will behave just as the DELAY MATERIAL (or DELAY FIXED) function does. It can be useful to do this if you want to specify the exact distribution of material within the delay process when it is starting.

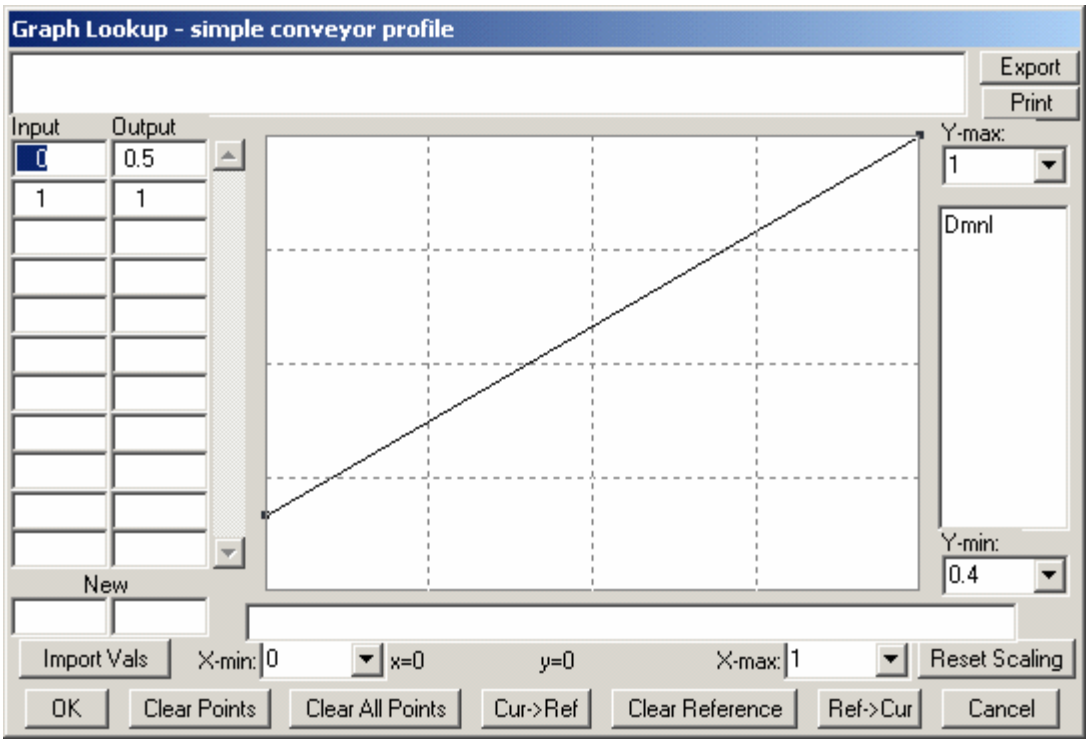
When the delay time changes the behavior of DELAY CONVEYOR is much more continuous then DELAY MATERIAL. The model *delay2.mdl* provides a comparison of these two functions for different profiles of delay time change. For example if the delay time is decreasing you will see the following comparison:



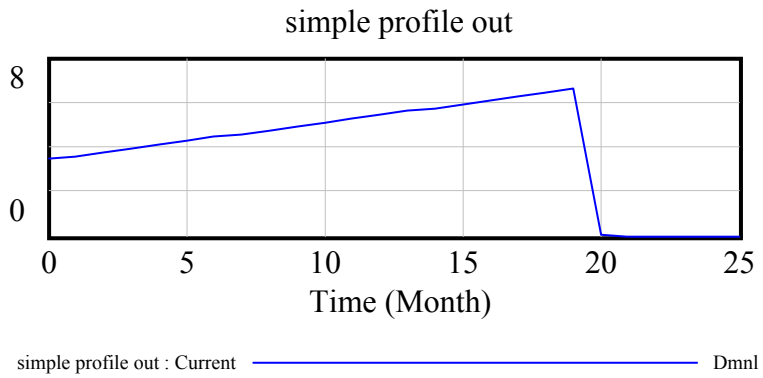
First, we don't get any of the doubling up, but instead a higher average value. Second, the conveyor finishes earlier because the decreasing conveyance time influences material that is already in the conveyor but not influence the DELAY MATERIAL function.

Initialization of Conveyors

In order to initialize a conveyor you provide a profile of the material in the conveyor to Vensim. This profile shows how material will come out of the conveyor. Vensim automatically scales both the domain and range of the profile so that the total amount of material and the total initial conveyance time matches what is specified. The model *convey1.mdl* demonstrates this. For example using the Lookup:

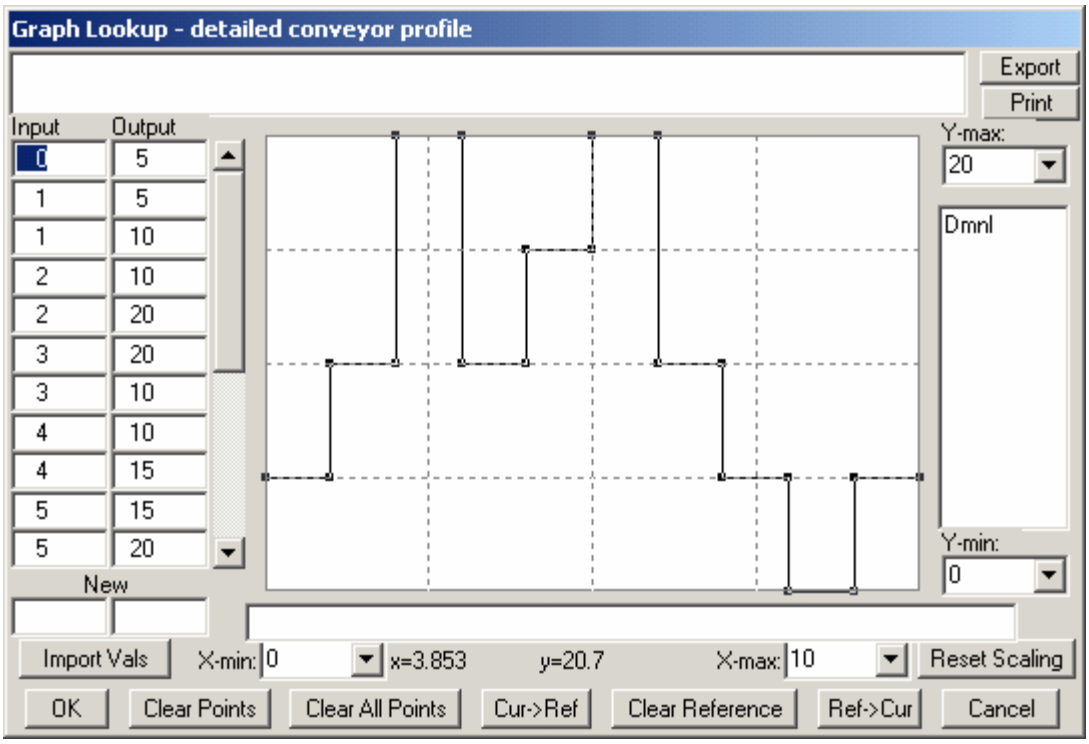


To initialize the DELAY CONVEYOR function would give the output (the input is zero so only initial material is ever output)



The total amount of stuff and the time over which the output occurs are determined by other arguments, but the shape is given by the Lookup. This makes it convenient to setup an approximate profiles, and this is usually all that is needed.

If you want to specify exactly what will come out at each time period you will need to use a more elaborate Lookup. For example using the lookup:



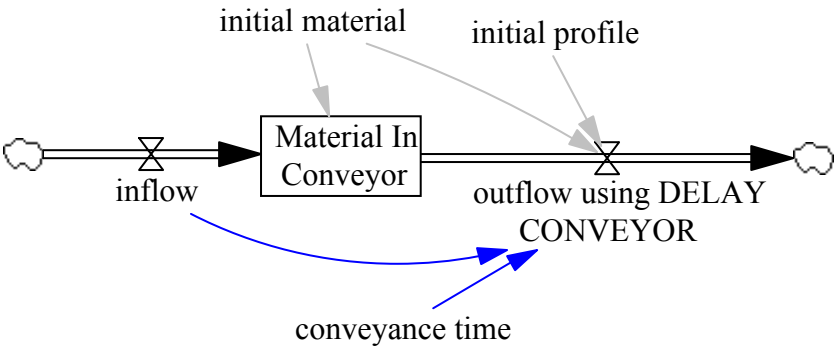
With an initial stock of 100 and delay time of 10 will give the output for *detailed profile out* of .

```
Time      0     1     2     3     4     5     6     7     8     9
          10
5  10     20    10    15    20    10    5     5e-6  5     0
```

In this case is you were to change either the total stock or, more importantly, the delay time these exact numbers would not longer hold. Notice also the small roundoff error that occurs at time 8. Such noninteger results are to be expected when using DELAY_CONVEYOR.

Material in Conveyors

If you are using conveyors without drainage then the amount of material in the conveyor is simply determined by the structure (*convey2.mdl*) :

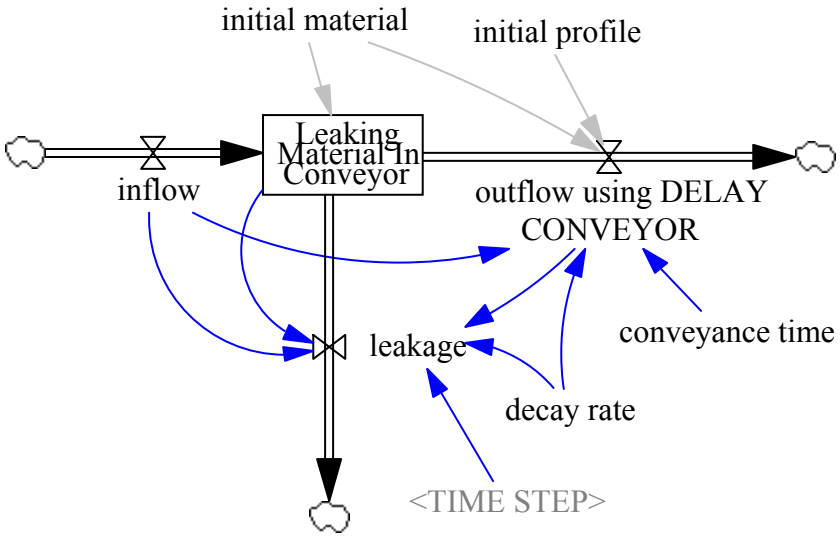


Here initial causes are displayed for clarity.

If there is leakage, however, a somewhat different structure is necessary. The leakage that occurs depends not only on the material currently in the conveyor but also on material that will be coming into

the conveyor. Stated differently, if you put something on the conveyor at time 0, then by time 1 it will already have done some leaking.

This more complete structure is in *convey3.mdl* and looks like:



And the equation for leakage is:

```
leakage= (Leaking Material In Conveyor +
(inflow-outflow using DELAY CONVEYOR)*TIME STEP)*
decay rate
```

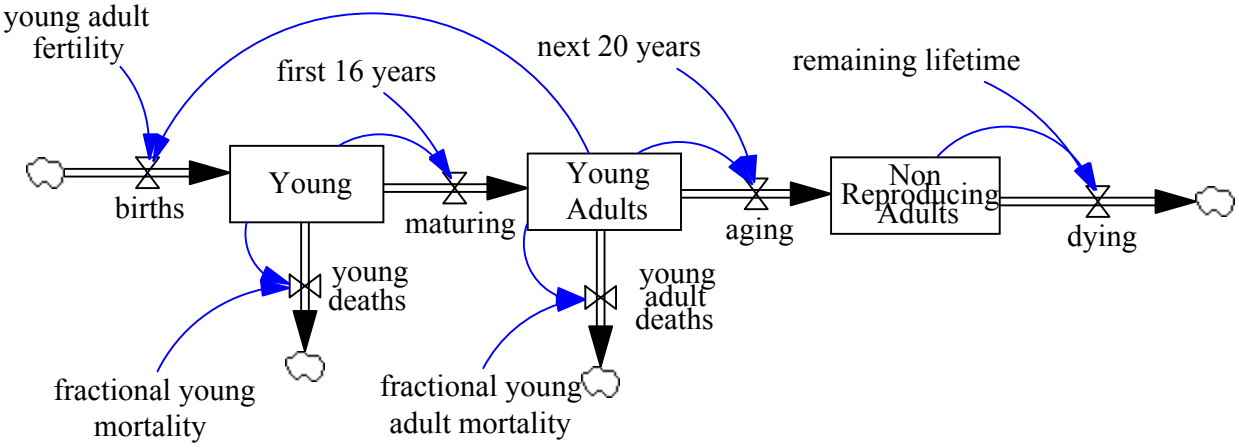
The middle term in this equation is the one that is used to correct for the looking ahead issue. It is a relatively small error, and one that is made smaller still then TIME STEP is small. It is often most sensible to simply ignore this and use the simpler formulation:

```
leakage= Leaking Material In Conveyor* decay rate
```

This formulation will cause a small error that is not important in most settings. The error does tend to accumulate at a positive rate (because the cumulative is less than cumulative inflow when there is leakage).

Population Example with Conveyors

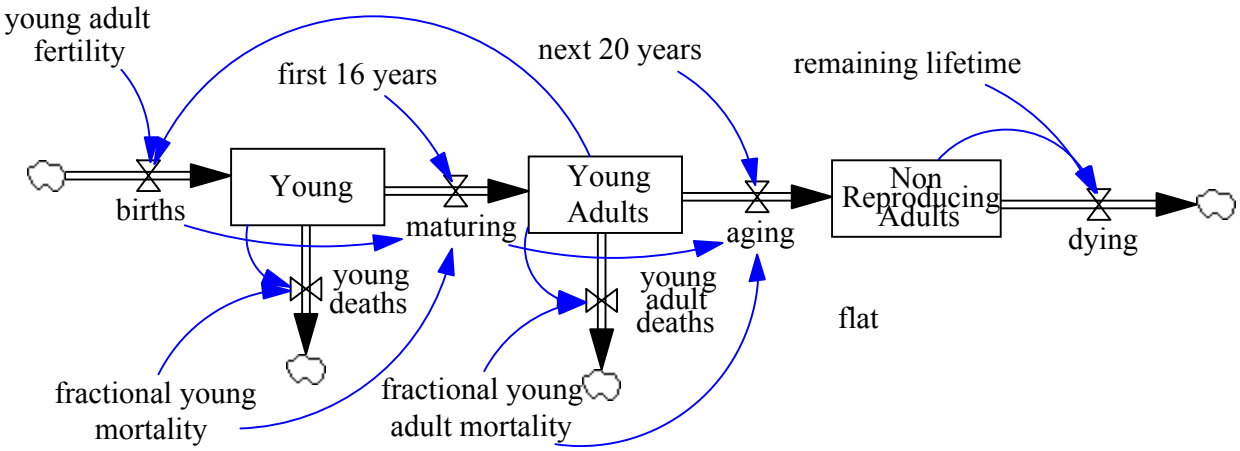
One common situation where you might want to include some discrete logic in an otherwise continuous model would be population dynamics. Using cohorts is a standard approach to doing this but something like (*convey4a.mdl)*



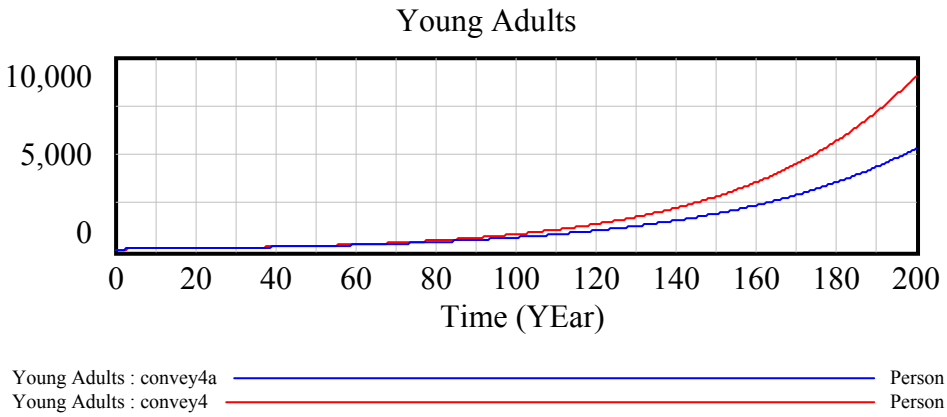
If the above model is formulated with the obvious equations such as

```
Maturing = young/first 16 years
```

Then, because of the nature of exponential decay, an increase in births at time 0 will lead to more young adults a year later. To overcome this we can reformulate with conveyor delays as (*convey4.mdl)*.



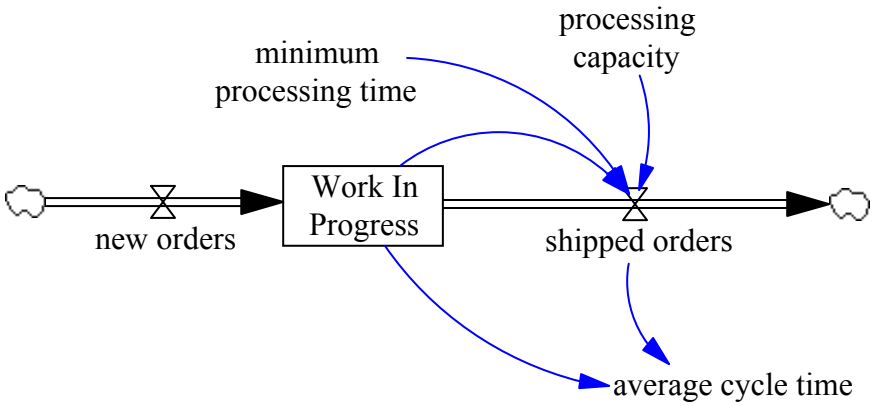
This is almost the same except that *maturing* and *aging* are being computed using DELAY CONVEYOR. The behavior, with the same set of parameters, is significantly different:



Queues

A Queue is simply a place where you put things and then take them away. The most common example of a queue is a line that you have to wait in, and such a queue is called a FIFO queue which means first in first out (the first person to arrive is the first person served). Vensim has some functions that make it easier to deal with keeping track of things in FIFO queues.

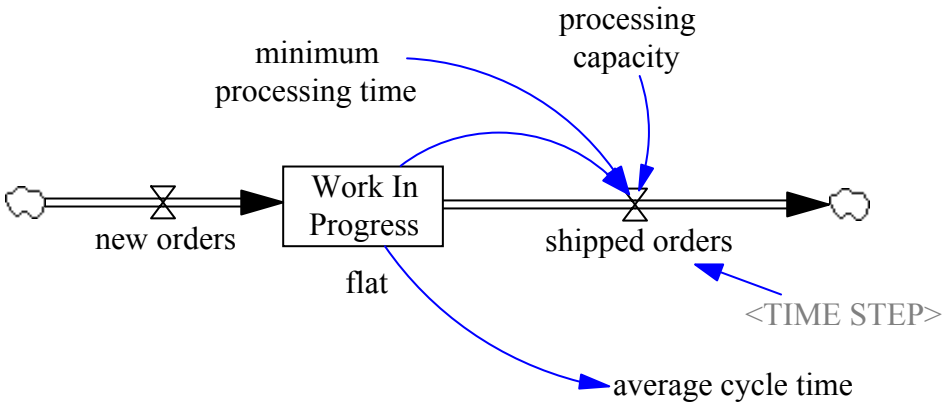
The QUEUE functions in Vensim all have two parts. First these is a special level that, like a regular level, allows you to accumulate things, but keeping track of things by when they arrive and exit rather than lumping them together into one number. For example suppose you are looking at order processing (*queue1.mdl)*:



This is a traditional continuous approach to representing this and does not make use of any queues. The equation for average cycle time is

```
average cycle time=ZIDZ(Work In Progress,shipped orders)
```

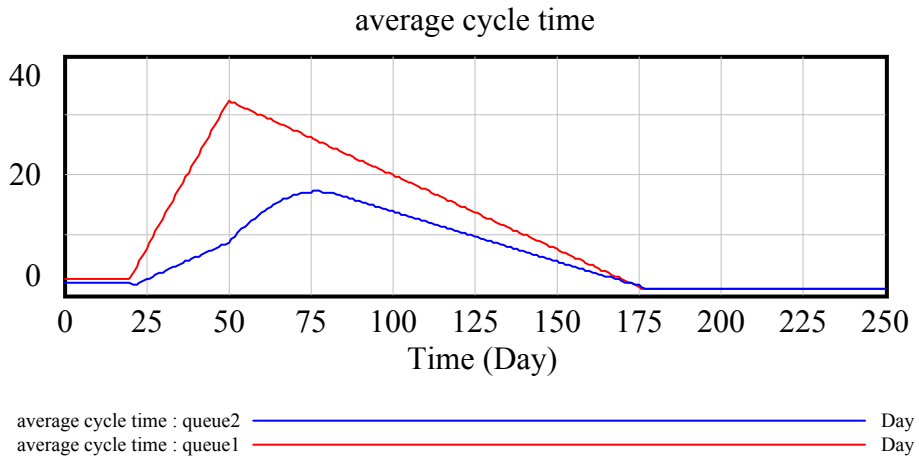
Compare this to a formulation using queues:



The diagram is actually nearly the same, except for the Lookup called *flat* and the dropping of the arrow from *shipped orders* to *average cycle time*. The changed equations are:

```
Work In Progress= QUEUE FIFO( new orders, shipped orders,
   flat, 100, 3 )
Units: Widget

shipped orders= MIN(processing capacity,
   QUEUE AGE IN RANGE(Work In Progress,
         minimum processing time, 1e+009)/TIME STEP)
Units: Widget/Day

average cycle time=QUEUE AGE AVERAGE(Work In Progress, 0)
Units: Day
```

If you compare simulations of these two models you will see that most variables behave the same with the exception of *average cycle time*.



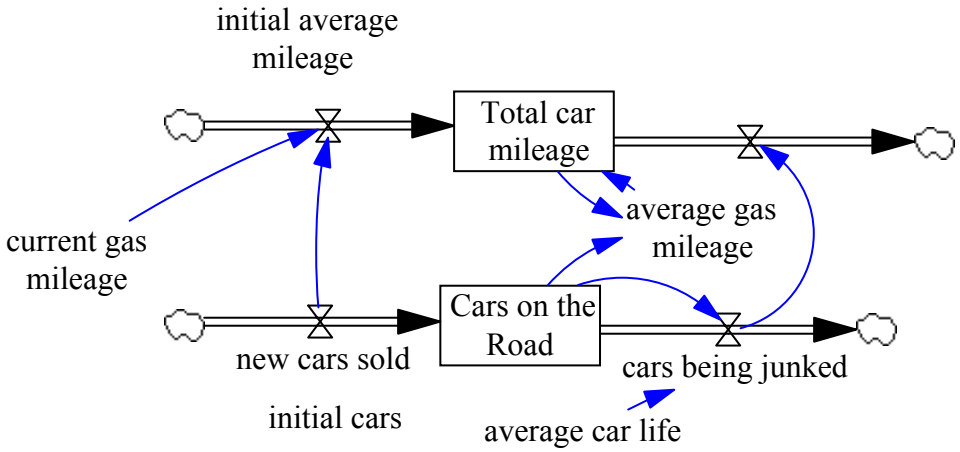
In the continuous formulation average cycle time increases from 2 to 32 in 30 days. That could only be true if nothing were being produced and all the orders were place at time 20. The formulation using QUEUE provides a more accurate answer.

In general the most value of the QUEUE functions comes because they give you the ability to get accurate cycle time and performance measurements.

QUEUES with Attributes

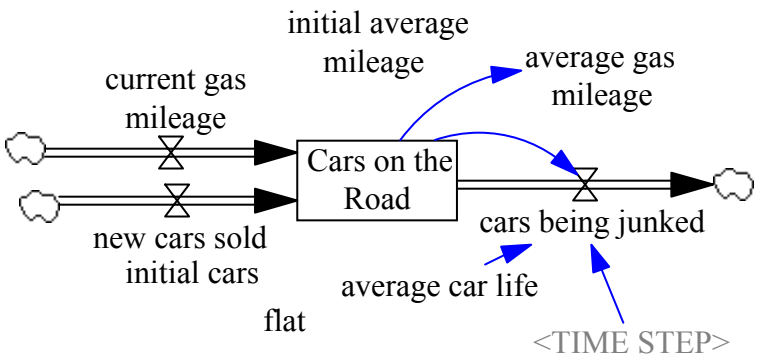
In addition to keeping track of how long something has waited in a queue, it is often useful to understand the distribution of attributes within the queue. In discrete event simulation the approach is to take individual entities and track them as they function within a system. Sometimes it makes sense to go to that level of detail using subscripts, but sometimes something a little bit less detailed is called for. Using queues with attributes allows you to get to somewhat more detail without the full machinery of subscripts.

As an example consider keeping track of the gas mileage of different vintages of cars. The typical continuous approach to this problem would be to use what is called a coflow as is shown here (*queue3.mdl*).



In a coflow the attribute is tracked in a separate level with units of measure `Car*Mile/Gallon`.

The coflow operates on a negative exponential residence distribution rather than a FIFO approach. Formulated using FIFO queues the model would become (*queue4.mdl*):



We have removed the abstract level total car mileage, and now show two flows into *Cars on the Road*. One of these flows is really the flow of an attribute, but this seems the clearest representation. The equation for *Cars on the Road* becomes:

```
Cars on the Road = QUEUE FIFO ATTRIB( new cars sold,
   cars being junked, current gas mileage, 0, flat, flat,
   initial cars, initial average mileage, average car life)
```

This equation is really taking two level equations and computing them at the same time. The *average gas mileage* is now computed using the formula

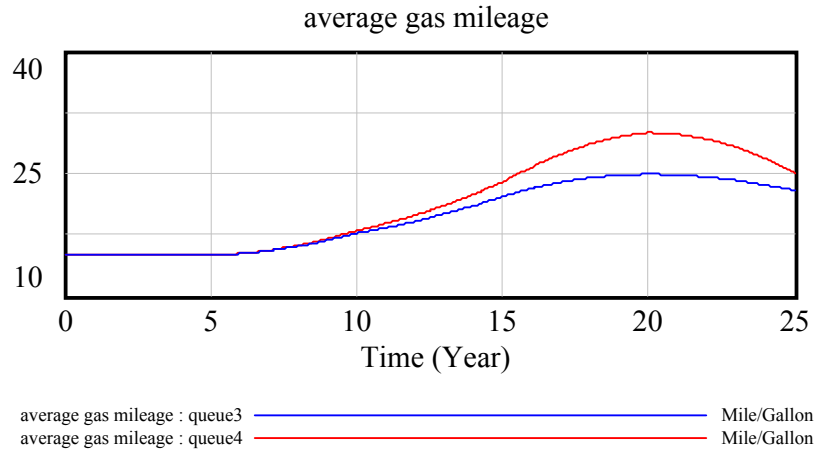
average gas mileage=

QUEUE ATTRIB AVERAGE(Cars on the Road, -1)

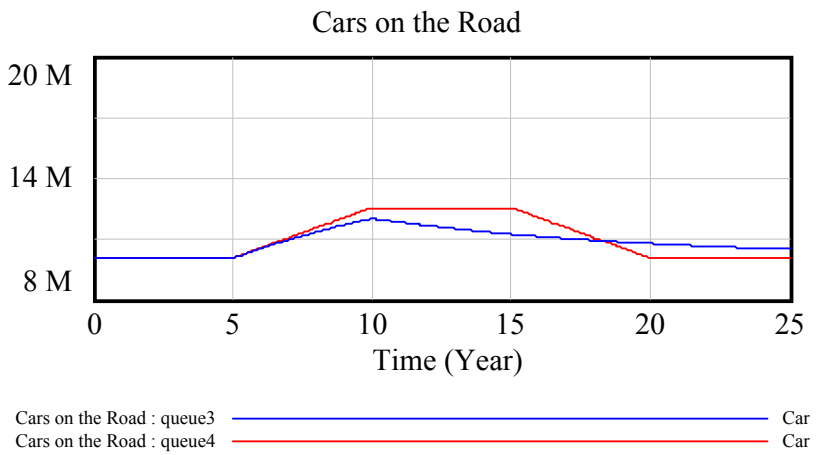
Finally the equation for *cars being junked* is replaced by:

```
cars being junked=QUEUE AGE IN RANGE(Cars on the Road,
   average car life, 1e+009)/TIME STEP
```

Looking at average gas mileage we see that there is significant difference between the two models:



An, of course there is a difference in *Cars on the Road*:

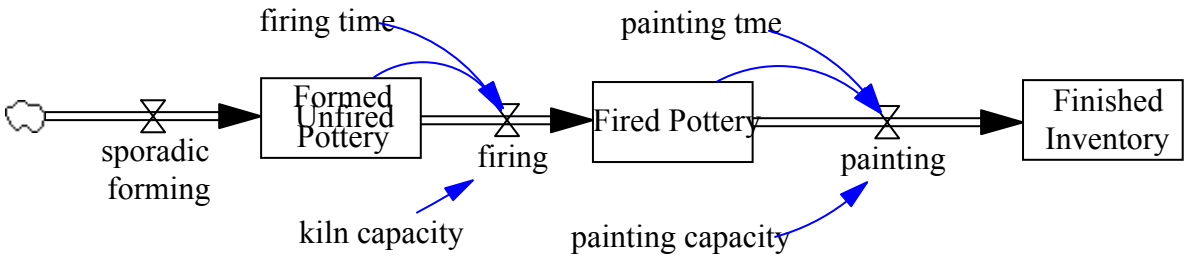


Unlike the cycle time computation, however, one of these is not clearly more right than the other. Both represent approximations. Cars are retired with neither a negative exponential distribution nor as a FIFO queue process.

Batch Delays

Most delays take some inflow, shift it in time, and smooth it out. As we saw in the first section the higher order the delay the less smoothing occurs. In some cases, however, you might want a smooth inflow to be broken up into batches, and this is what DELAY BATCH does.

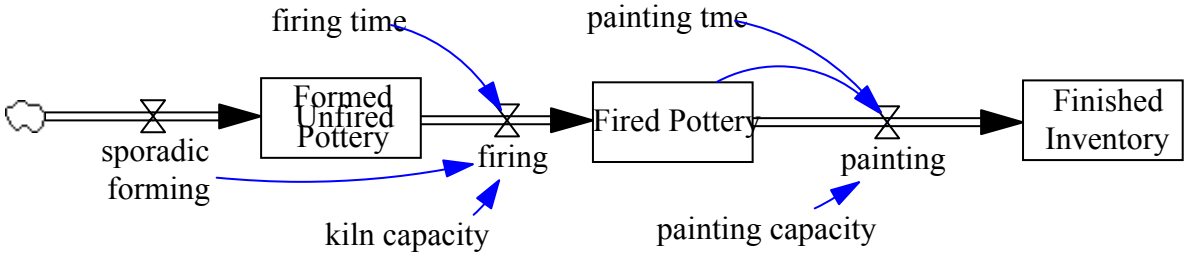
Consider the production of pottery in a facility with a single large kiln. A typical continuous representation of this would be a structure such as (*batch1.mdl*):



Here firing is given by the equation:

```
firing=  MIN(kiln capacity/firing time,
   Formed Unfired Pottery/firing time)
```

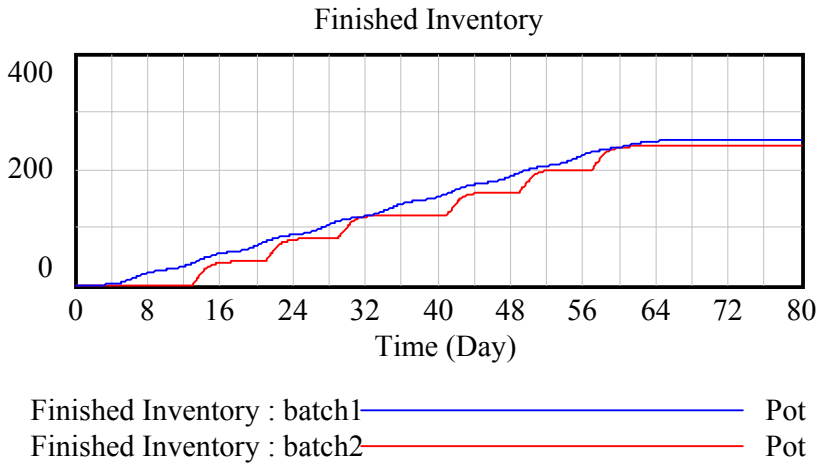
However, if the kiln loaded, then started it is more accurate to represent the model with a batch process (batch2.mdl):



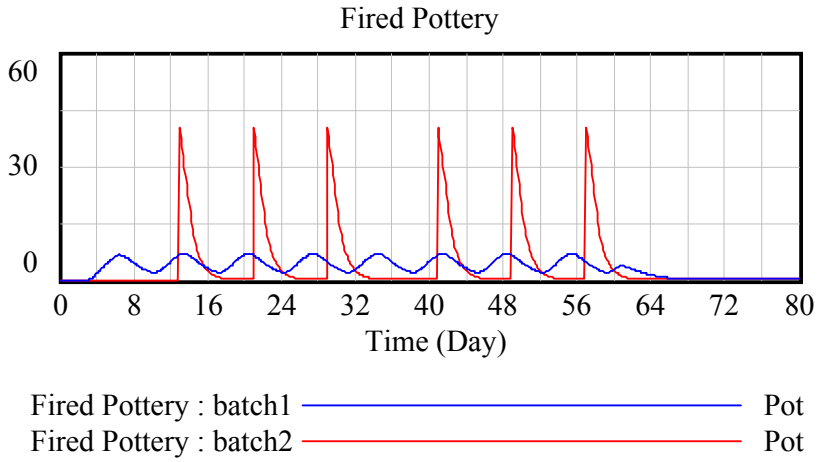
Now the equation for firing is:

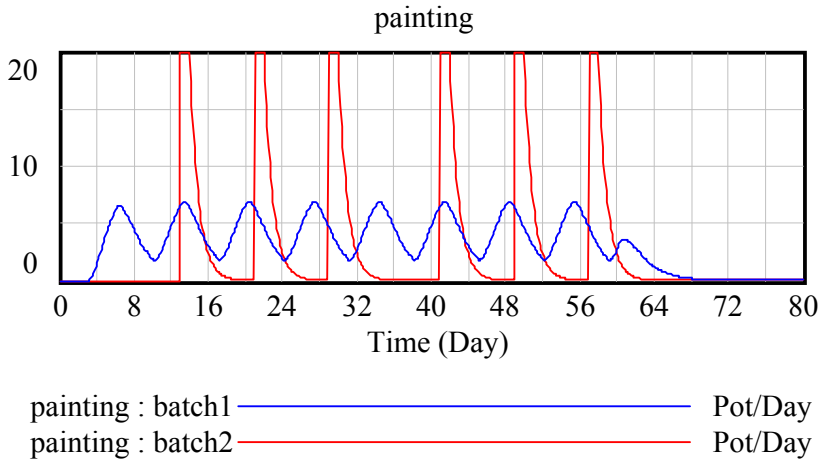
```
firing=  DELAY BATCH(sporadic forming, kiln capacity,
firing time, 0, 0, 0)
```

The final output is much more episodic, though not too much different:



but there is a much bigger variation in the amount of fired pottery and consequently in the level of painting activity.





In situations where real activities are episodic and the capacity constraints tend to be limiting during those episodes using the DELAY BATCH function can give more accurate results.

10 Molecules and Other Resources

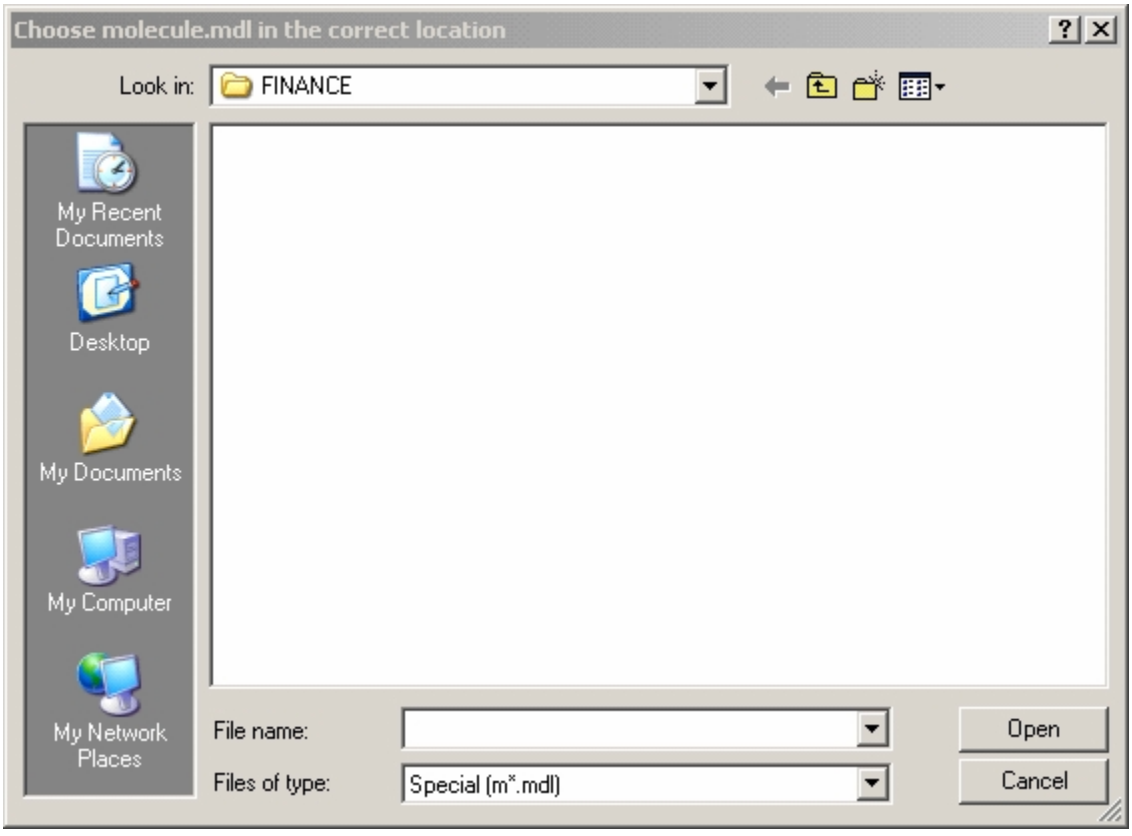
Molecules are small pieces of structure that repeat themselves again and again. The have been around a long time, but were formalized by Jim Hines in the 1990s. Basically Jim posed the question "why are experienced modelers so much faster at developing models than beginners?" The answer he came up with was that experienced modelers have building block structures stored in their heads that make it much easier to formulate most equations. He called these elements of structure Molecules.

The molecules themselves are not part of Vensim. What Vensim provides is a mechanism to open and review the molecules. You can also, if you wish, extend or even replace the molecules with your own.

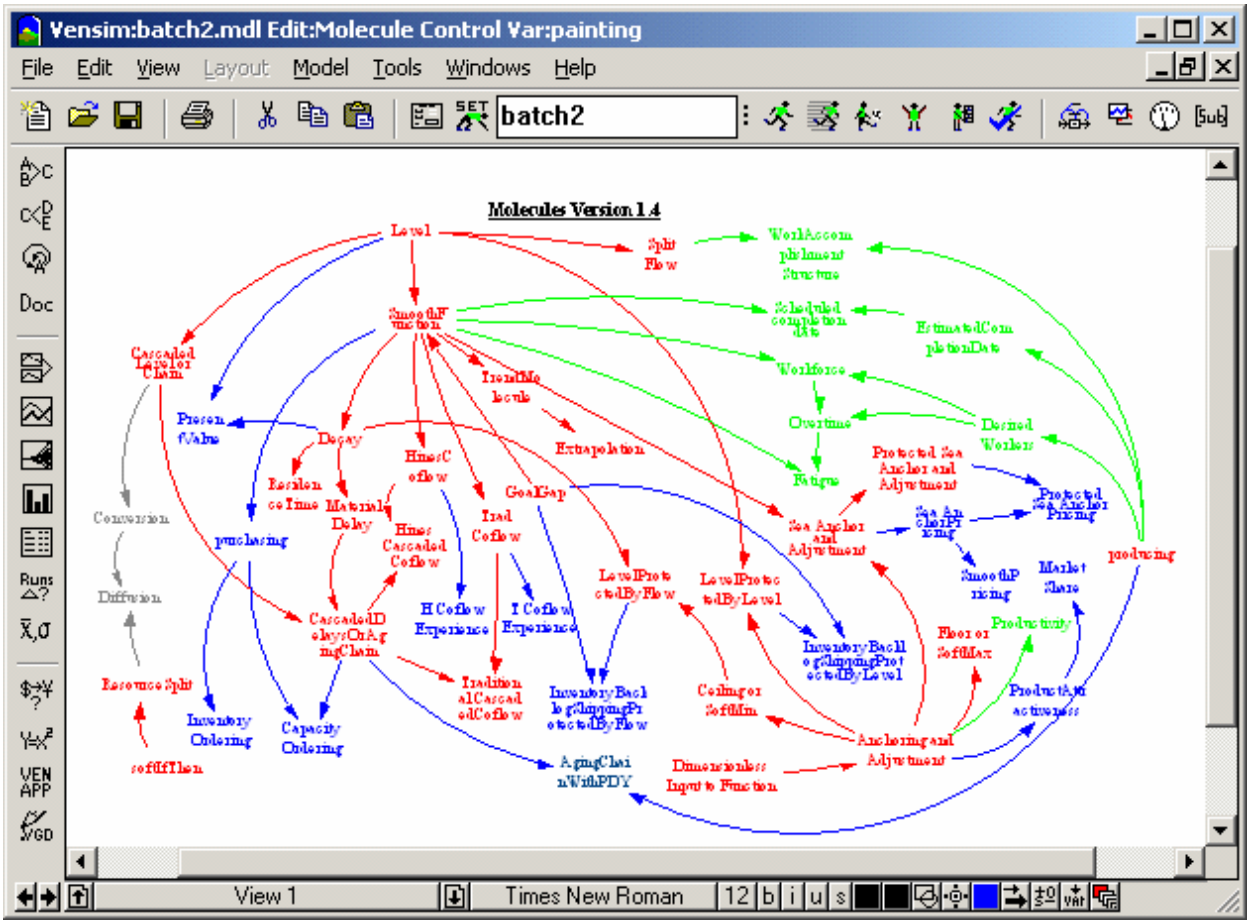
Getting and using the Molecules

The molecules are available from our website http://www.vensim.com/molecule.html. They are supplied free of charge and you are welcome to make use of them in your models with or without attribution. There is, of course, a prohibition on using them to create a conceptually similar resource for commercial purposes.

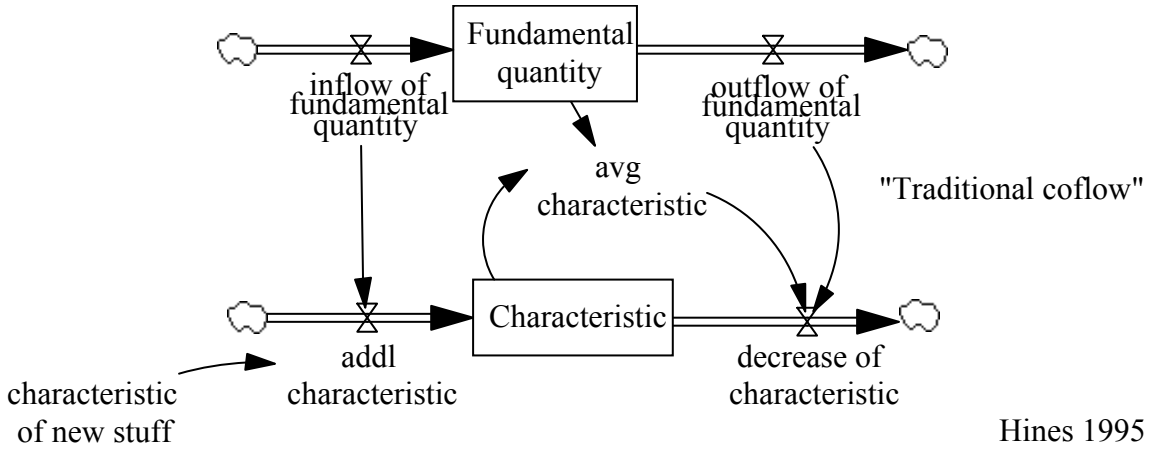
To install the molecules just download and run the installer. After you have installed them you can access them using the **Windows>Molecules** menu item. The first time you do this you may be asked for to location.



Just navigate to the place you installed them (normally *c:\program files\vensim\molecule*)and select *molecule.mdl*. After that when you ask for the molecules you should see the molecule window:



Click on one of the molecule names to open it. For example the Traditional Coflow



The molecules are complete little models, but you can't simulate them from the molecule window. You can, however, peek at the equations by holding down the Ctrl+Shift keys and clicking on the variables. You can copy and paste them too other places.

From our experience with molecules there are most useful for study. Why you can copy and paste them the terminology and units all need to be changed anyway and this doesn't really save much time. In addition, it is often appropriate to make small changes to the molecules to adapt them to new situations.

Extending the Molecules

If you open the model *molecule.mdl* in as a regular model you will be able to edit the different equations. If you look at the equation for *Trad Coflow* you will see that is has the comment:

#Tcoflow.mdl#

If you want to add molecules to the list just put them in the model *molecule.mdl* and then reference their names as shown. It is also permissible to nest definitions of molecules in this manner.

Appendix A — Models that Come with Vensim

Vensim ships with a number of different models. Many of the models are described in this modeling guide, and many in the User's Guide. There are also a number of models that are not described in the documentation. Here we outline the location and naming conventions for the documented models and describe a number of other models that come with Vensim.

Modeling Guide Models

The Modeling Guide models will normally install into the *mguide* subdirectory of the *models* subdirectory of Vensim. Normally the full path for this directory if *c:\Program Files\Vensim\models\mguide*. When you install Vensim you have the option to install these models into a different drive or directory. If you have done this they will be in a different location.

Under the *mguide* directory there are subdirectories for each chapter in this guide. The directories are numbered with the Chapter number, followed by a brief shorthand for what the model or models are about. The directories are:

- ***1Fundam*** - the fundamental structure examples from Chapter 1.
- ***2WFINV*** - the workforce-inventory example from Chapter 2.
- ***3PROJ*** - the project models developed in Chapter 3.
- ***4GROW*** - models dealing with growth and diffusion discussed in Chapter 4.
- ***5cap*** - models dealing with capacity adjustment in a growing market discussed in Chapter 5.
- ***6comp*** - models dealing with competitive dynamics discussed in Chapter 6. These models make use of subscripts.
- ***7fin*** - models of financial variables linked to the emerging market model discussed in Chapter 7.
- ***8osc*** - other examples of oscillating structures. A pendulum model and a model of a furnace discussed in Chapter 8.
- ***9discret*** models incorporating discrete functions detailed in Chapter 9.

The names of the individual models are mentioned in their respective chapters.

User's Guide Models

The User's Guide models will normally install into the *guide* subdirectory of the *models* subdirectory of Vensim. Normally the full path for this directory if *c:\Program Files\Vensim\models\guide*. When you install Vensim you have the option to install these models into a different drive or directory. If you have done this they will be in a different location.

Under the *guide* directory there are subdirectories for each chapter in the *User's Guide*. The directories are named *Chap01*, *Chap02* and so on. The User's Guide gives details on the model names. Note that Chap18 has two subdirectories — one for calibration and one for policy optimization.

Many chapters provide starter models for you to work from, and these are located in each Chapter directory (*Chap01*, *Chap02* etc). User's Guide models that are in complete form are stored in a *\complete* subdirectory of each Chapter.

Sample Models

The Sample models will normally install into the *sample* subdirectory of the *models* subdirectory of Vensim. Normally the full path for this directory if *c:\Program Files\Vensim\models\sample*. When you install Vensim you have the option to install these models into a different drive or directory. If you have done this they will be in a different location.

We give a very brief description of what the models are and when appropriate an indication of how they relate to the Vensim documentation. The list is arranged alphabetically by directory.

intro.mdl

This is the only model in the *sample* directory itself. It is closely related to the workforce inventory model described in Chapter 2. In this model there is a fixed workforce and a loop through overtime, morale and productivity that causes the model to oscillate.

bpr

The *bpr* subdirectories contains several models related to business process engineering.

BPR0.vmf

This is a simple word an arrow description of a business process. This model cannot be simulated. There is no feedback in this model.

BPR1.vmf

This model presents programming style flow diagram that outlines a business process. The model cannot be simulated but demonstrates the flexibility of sketches built in Vensim.

BPR2.vmf

This is a mechanical simulation model of a business process. This model has only limited feedback in the form of stopping processing when there is no work to do. It is very similar to the first project model developed in Chapter 3.

BPR3.vmf

This model has the same basic physics as bpr2.vmf but uses feedback policies to manage the capacity allocations throughout. It can generate oscillatory behavior just like the workforce inventory model of Chapter 2.

Extra

age.mdl

A model about populations aging using aging cohorts.

agechain.mdl

A similar model to *age.mdl* but using chained stocks to achieve the aging.

ball.mdl

This model simulates a bouncing ball as it impacts the ground and uses two different time steps depending on its position.

BURNOUT.mdl

This is a worker burnout model written by Jack Homer and published in *The System Dynamics Review* Volume 1, Number 1, 1985, Pages 42-62.

caffeine.mdl

This model shows the effect of caffeine on the central nervous system through effects on drowsiness.

commod.mdl

A model of the Hog market commodities trading, based on the real market originally created by Denis Meadows.

compete1.mdl

A simple price/inventory stock model.

cool.mdl

Model of a system cooling (cup of coffee).

corpgrth.mdl

Jay Forrester's simple implementation of a corporate growth model. As featured in *Principles of Systems*, by Jay Forrester (Pegasus).

eit.mdl

Easter Island tree decimation by the inhabitants of Easter Island hundreds of years ago. Extremely simple model.

epidemic.mdl

The classic Bass diffusion model of the spread of an epidemic.

gravity.mdl

Gravitational attraction of two massive bodies.

nephron5.mdl

This is a model of a rats kidney. It demonstrates some fast/slow dynamics and uses FIND ZERO function to solve them. See Chapter 8 of the *Reference Manual*. It is based on the work of Erik Mosekilde.

Nep4simul.mdl

The same model using the SIMULTANEOUS function as an alternative.

POPGAME.mdl

This is a simple population model configured to be used in gaming. It is essentially the same as the model developed in Chapter 6 of the *Tutorial*.

poppyr.mdl

This model features a large subscripted base of ages in a population.

PROCRAST.mdl

This is a simple model of the procrastination process. It is closely related to the project models discussed in Chapter 3.

PROJ.mdl

This is a simple project model and is very nearly the same as that developed in *Introduction to System Dynamics Modeling with DYNAMO,* by G.P. Richardson and A.L. Pugh (The MIT Press, Cambridge, MA, 1981) available from Pegasus Communications.

rabfox.mdl

The rabbit model extended to a predator/prey system with foxes.

snowball.mdl

Simple exponential growth in a snowball.

speed.mdl

Simple goal seeking with negative feedback to attain a particular speed.

TUBS.mdl†

This is a simple model of water flow that demonstrates a number of useful subscripting tricks.

world.mdl

This is the model presented in *World Dynamics* by Jay W. Forrester (The MIT Press, Cambridge, MA, 1971; second edition, 1973).

finance (finance.vmf)

This is a financial model that is attached to a emerging market model. It is closely related to the financial modeling discussed in Chapter 7.

kalman (wfkal.mdl)

This is a version of the workforce inventory model developed in Chapter 2 that has been modified to demonstrate the use of Kaman Filtering. The use of this model is discussed in Chapter 10 of the *Reference Manual*.

maint1 (maint1.vmf)

This model is a simplified look at the proactive maintenance problem. It focuses on the transition from a reactive maintenance strategy to a proactive strategy looking at material consumption and labor requirements.

market (market.vmf)

This is a model of market growth in a market for which sales representatives are fundamental to achieving this growth.

mproject (mproj3.vmf†)

This is a multistage model of projects. It is related to the project models in Chapter 3 and also the multistage project models discussed in Chapter 13 of the *Tutorial*.

urban (urban.vmf)

This is the model of urban development and decay presented in *Urban Dynamics* by J.W. Forrester (The MIT Press, Cambridge, MA, 1969) available from Pegasus Communications.

Note that the Urban model that shipped with Version 4 of Vensim had a transcription error in it, so be sure to update this model if you want to use it.

wrld3-91 (wrld3-91.vmf)

This is an updated version of the model that was used for the book *The Limits to Growth* by D.H. Meadows et al (Universe Books, New York, 1972). The model was updated for *Beyond the Limits:*

confronting global collapse, envisioning a sustainable future by D.H. Meadows et al (Chelsea Green Publishing Company, Post Mills, VT, 1992).

Index

T

U

V

W